SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator *

Mutsuo Saito[1] and Makoto Matsumoto[2]

[1] `saito@math.sci.hiroshima-u.ac.jp` Dept. of Math. Hiroshima University
[2] `m-mat@math.sci.hiroshima-u.ac.jp` Dept. of Math. Hiroshima University

Summary. Mersenne Twister (MT) is a widely-used fast pseudorandom number generator (PRNG) with a long period of $2^{19937}-1$, designed 10 years ago based on 32-bit operations. In this decade, CPUs for personal computers have acquired new features, such as Single Instruction Multiple Data (SIMD) operations (i.e., 128-bit operations) and multi-stage pipelines. Here we propose a 128-bit based PRNG, named SIMD-oriented Fast Mersenne Twister (SFMT), which is analogous to MT but making full use of these features. Its recursion fits pipeline processing better than MT, and it is roughly twice as fast as optimized MT using SIMD operations. Moreover, the dimension of equidistibution of SFMT, as a 32-bit integer generator, is better than MT.

We also introduce a block-generation function, which fills an array of 32-bit integers at one time. It speeds up the generation by a factor of two. A speed comparison with other modern generators, such as multiplicative recursive generators, shows an advantage of SFMT.

Key words: Mersenne Twister, SFMT, SIMD, 128-bit, block generation

1 Introduction

Recently, the scale of simulations is getting larger, and faster pseudorandom number generators (PRNGs) are required. The power of CPUs for usual personal computers are now sufficiently strong for such purposes, and the necessity of efficient PRNGs for CPUs on PCs is increasing. One such generator is Mersenne Twister (MT) [8], which is based on a linear recursion modulo 2 over 32-bit words. An implementation MT19937 has the period of $2^{19937}-1$.

MT was designed 10 years ago, and the architectures of CPUs for Personal Computers, such as Pentium and PowerPC, have changed. They have Single

* This study is partially supported by JSPS/Ministry of Education Grant-in-Aid for Scientific Research No.18654021, No. 16204002, and JSPS Core-to-Core Program No.18005.

Instruction Multiple Data (SIMD) operations, which may be regarded as operations on 128-bit registers. Also, they have more registers and automatic parallelisms by multi-stage pipelining. These are not reflected in the design of MT.

In this article, we propose an MT-like pseudorandom number generator that makes full use of these new features: SFMT, SIMD-oriented Mersenne Twister. We implemented an SFMT with the period a multiple of $2^{19937}-1$, named SFMT19937, which has a better equidistribution property than MT. SFMT is much faster than MT, even without using SIMD instructions.

There is an argument that the CPU time consumed for function calls to PRNG routines occupies a large part of the random number generation, and consequently it is not so important to improve the speed of PRNG (cf. [11]). This is not always the case: one can avoid the function calls by (1) inline-expansion and/or (2) generation of pseudorandom numbers in an array at one call. Actually some demanding users re-coded MT to avoid the function call; see the homepage of [8]. In this article, we introduce a block-generation scheme which is much faster than using function calls.

2 SIMD-oriented Fast Mersenne Twister

We propose SIMD-oriented Fast Mersenne Twister (SFMT) pseudorandom number generator. It is a Linear Feedbacked Shift Register (LFSR) generator based on a recursion over $\mathbb{F}_2^{128}$. We identify the set of bit $\{0,1\}$ with the two element field $\mathbb{F}_2$. This means that every arithmetic operation is done modulo 2. A w-bit integer is identified with a horizontal vector in $\mathbb{F}_2^w$, and $+$ denotes the sum as vectors (i.e., bit-wise exor), not as integers. We consider three cases: w is 32, 64 or 128.

2.1 LFSR generators

A LFSR method is to generate a sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ of elements $\mathbb{F}_2^w$ by a recursion

$$\mathbf{x}_{i+N} := g(\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+N-1}), \tag{1}$$

where $\mathbf{x}_i \in \mathbb{F}_2^w$ and $g : (\mathbb{F}_2^w)^N \to \mathbb{F}_2^w$ is an $\mathbb{F}_2$-linear function (i.e., the multiplication of a $(wN \times w)$-matrix from the right to a wN-dimensional vector) and use it as a pseudorandom w-bit integer sequence. In the implementation, this recursion is computed by using an array `W[0..N-1]` of N integers of w-bit size, by the simultaneous substitutions

$$\mathtt{W[0]} \leftarrow \mathtt{W[1]},\ \mathtt{W[1]} \leftarrow \mathtt{W[2]},\ \ldots, \mathtt{W[N-2]} \leftarrow \mathtt{W[N-1]},\ \mathtt{W[N-1]} \leftarrow g(\mathtt{W[0]}, \ldots, \mathtt{W[N-1]}).$$

The first $N-1$ substitutions shift the content of the array, hence the name of LFSR. Note that in the implementation we may use an indexing technique to avoid computing these substitutions, see [5, P.28 Algorithm A]. The array

`W[0..N-1]` is called the state array. Before starting the generation, we need to set some values to the state array, which is called the initialization.

Mersenne Twister (MT) [8] is an example with

$$g(\mathbf{w}_0, \dots, \mathbf{w}_{N-1}) = (\mathbf{w}_0|\mathbf{w}_1)A + \mathbf{w}_M,$$

where $(\mathbf{w}_0|\mathbf{w}_1)$ denotes the concatenation of $32-r$ MSBs of $\mathbf{w}_0$ and r LSBs of $\mathbf{w}_1$, and A is a (32×32)-matrix for which the multiplication $\mathbf{w}A$ is computable by a few bit-operations, and M is an integer $(1 < M < N)$. Its period is $2^{32N-r}-1$, chosen to be a Mersenne prime. To obtain a better equidistribution property, MT transforms the sequence by a suitably chosen (32×32) matrix T, namely, MT outputs $\mathbf{x}_0T, \mathbf{x}_1T, \mathbf{x}_2T, \dots$ (called tempering).

2.2 New features of modern CPUs for personal computers

Modern CPUs for personal computers (e.g. Pentium and PowerPC) have new features such as (1) fast integer multiplication instructions (2) fast floating point operations (3) SIMD operations (4) multi-stage pipelining. These were not common to standard PC CPUs, when MT was designed.

An advantage of $\mathbb{F}_2$-linear generators over integer multiplication generators (such as Linear Congruential Generators [5] or Multiple Recursive Generators [6]) was high-speed generation by avoiding multiplications. This advantage is now smaller, since 32-bit integer multiplication is now quite fast.

Among the new features, (3) and (4) fit $\mathbb{F}_2$-linear generators. Our idea is simple: to design a 128-bit integer PRNG, considering the benefit of such parallelism in the recursion.

2.3 The recursion of SFMT

We choose g in the recursion (1) as

$$g(\mathbf{w}_0, \dots, \mathbf{w}_{N-1}) = \mathbf{w}_0A + \mathbf{w}_MB + \mathbf{w}_{N-2}C + \mathbf{w}_{N-1}D, \tag{2}$$

where $\mathbf{w}_0, \mathbf{w}_M, \dots$ are $w(= 128)$-bit integers (= horizontal vectors in $\mathbb{F}_2^{128}$), and A, B, C, D are sparse 128×128 matrices over $\mathbb{F}_2$ for which $\mathbf{w}A, \mathbf{w}B, \mathbf{w}C, \mathbf{w}D$ can be computed by a few SIMD bit-operations. The choice of the suffixes $N-1$, $N-2$ is for speed: in the implementation of g, `W[0]` and `W[M]` are read from the array `W`, while the copies of `W[N-2]` and `W[N-1]` are kept in two 128-bit registers in the CPU, say `r1` and `r2`. Concretely speaking, we assign `r2` $\leftarrow$ `r1` and `r1` $\leftarrow$ "the result of (2)" at every generation, then `r2` (`r1`) keeps a copy of `W[N-2]` (`W[N-1]`, respectively). The merit of doing this is to use the pipeline effectively. To fetch `W[0]` and `W[M]` from memory takes some time. In the meantime, the CPU can compute $\mathbf{w}_{N-2}C$ and $\mathbf{w}_{N-1}D$, because copies of $\mathbf{w}_{N-2}$ and $\mathbf{w}_{N-1}$ are kept in the registers. This selection was made through experiments on the speed of generation.

By trial and error, we searched for a set of parameters of SFMT, with the period being a multiple of $2^{19937}-1$ and having good equidistribution properties. The degree of recursion N is $\lceil 19937/128 \rceil = 156$, and the linear transformations A, B, C, D are as follows.

- $\mathbf{w}A := (\mathbf{w} \overset{128}{<<} 8) + \mathbf{w}$.
  This notation means that $\mathbf{w}$ is regarded as a single 128-bit integer, and $\mathbf{w}A$ is the result of the left-shift of $\mathbf{w}$ by 8 bits. There is such a SIMD operation in both Pentium SSE2 and PowerPC AltiVec SIMD instruction sets (SSE2 permits only a multiple of 8 as the amount of shifting). Note that the notation $+$ means the exclusive-or in this article.
- $\mathbf{w}B := (\mathbf{w} \overset{32}{>>} 11) \& (\texttt{BFFFFFF6 BFFAFFFF DDFECB7F DFFFFFEF})$.
  This notation means that $\mathbf{w}$ is considered to be a quadruple of 32-bit integers, and each 32-bit integer is shifted to the right by 11 bits, (thus the eleven most significant bits are filled with 0s, for each 32-bit integer). The C-like notation & means the bitwise AND with a constant 128-bit integer, denoted in the hexadecimal form.
  In the search, this constant is randomly generated as follows. Each bit in the 128-bit integer is independently randomly chosen, with the probability to choose 1 being 7/8. This is because we prefer to have more 1's for a denser feedback.
- $\mathbf{w}C := (\mathbf{w} \overset{128}{>>} 8)$.
  The notation $(\mathbf{w} \overset{128}{>>} 8)$ is the right shift of an 128-bit integer by 8 bits, similar to the first.
- $\mathbf{w}D := (\mathbf{w} \overset{32}{<<} 18)$.
  Similar to the second, $\mathbf{w}$ is cut into four pieces of 32-bit integers, and each of these is shifted by 18 bits to the left.

All these instructions are available in both Intel Pentium's SSE2 and PowerPC's AltiVec SIMD instruction sets. Figure 1 shows a concrete description of SFMT19937 generator with period a multiple of $2^{19937}-1$.

2.4 How we selected the recursion and parameters.

In typical applications, we need 32-bit integers or 64-bit integers as the output of PRNGs, instead of 128-bit integers. SFMT19937 is a 128-bit integer generator, but we may regard it as a 32-bit (64-bit) integer generator, by partitioning each 128-bit into four 32-bit (two 64-bit, respectively) integers. Let $\mathbf{x}$ be a 128-bit integer. Then we denote by $\mathbf{x}[0]$ the 32 LSBs of $\mathbf{x}$, $\mathbf{x}[1]$ the 33rd–64th bits, $\mathbf{x}[2]$ the 65rd–96th bits, and $\mathbf{x}[3]$ the 32 MSBs. Thus, $\mathbf{x}$ is the concatenation of $\mathbf{x}[3], \mathbf{x}[2], \mathbf{x}[1], \mathbf{x}[0]$, in this order.

SFMT generates a sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ of 128-bit integers. Then, they are converted to a sequence of 32-bit integers $\mathbf{x}_0[0], \mathbf{x}_0[1], \mathbf{x}_0[2], \mathbf{x}_0[3], \mathbf{x}_1[0], \mathbf{x}_1[1], \ldots,$

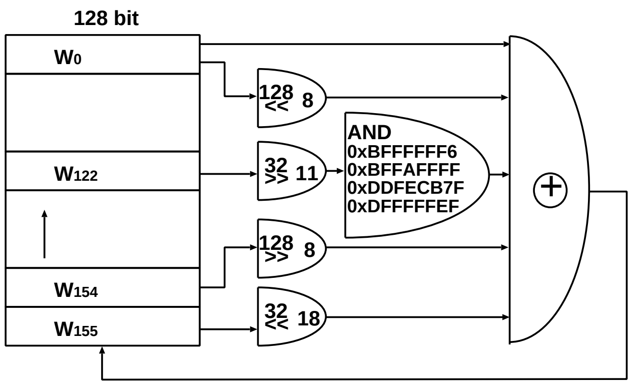

Fig. 1. A circuit-like description of SFMT19937.

or 64-bit integers $\mathbf{x}_0[1]\mathbf{x}_0[0], \mathbf{x}_0[3]\mathbf{x}_0[2], \mathbf{x}_1[1]\mathbf{x}_1[0]$, i.e., by the so-called little-endian system (see §8.2 for a compatible implementation with big-endian systems).

We write codes to compute the lower bounds of the period (see §6), the dimension of equidistribution (DE, see below), and to measure the generation speed for Pentium SSE2 and PowerPC AltiVec, with and without SIMD, as 32-bit or 64-bit integer generators.

We fixed a type of recursion, then using a small model (e.g. the period is a smaller Mersenne prime $2^{607} - 1$), we tested whether it attains the desired period, good DE, and fast generations, by trial and error on the type of recursion, and reached (2). Then, we searched for the parameters with the period being a nonzero multiple of $2^{19937} - 1$. Roughly 10 sets of parameters were found per day, by using four AMD Athlon 64 machines. We selected the best parameter with respect to DE.

We briefly recall the definition of dimension of equidistribution (cf. [3]).

Definition 1. *A periodic sequence with period P*

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

*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 $i = 0, \ldots, P-1$. We allow an exception for the all-zero pattern, which may occur once less often. (This last loosening of the condition is technically necessary, because the zero state does not occur in an $\mathbb{F}_2$-linear generator). The largest value of such k is called the dimension of equidistribution (DE).

We want to generalise this definition slightly. We define the k-window set of the periodic sequence χ as

$$W_k(\chi) := \{(\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+k-1}) | i = 0, 1, \ldots, P-1\},$$

which is considered as a *multi-set*, namely, the multiplicity of each element is considered.

For a positive integer m and a multi-set T, let us denote by $m \cdot T$ the multi-set where the multiplicity of each element in T is multiplied by m. Then, the above definition of equidistribution is equivalent to

$$W_k(\chi) = (m \cdot O^k) \setminus \{\mathbf{0}\},$$

where m is the multiplicity of the occurrences, and the operator $\setminus$ means that the multiplicity of $\mathbf{0}$ is subtracted by one.

Definition 2. *In the above setting, if there exist a positive integer m and a multi-subset $D \subset (m \cdot O^k)$ such that*

$$W_k(\chi) = (m \cdot O^k) \setminus D,$$

we say that χ is k-dimensionally equidistributed with defect ratio $\#(D)/\#(m \cdot O^k)$, where the cardinality is counted with multiplicity.

Thus, in Definition 1, the defect ratio up to $1/(P+1)$ is allowed to claim the dimension of equidistribution. If $P = 2^{19937} - 1$, then $1/(P+1) = 2^{-19937}$. In the following, the dimension of equidistribution allows the defect ratio up to 2^{-19937}.

For a w-bit integer sequence, its *dimension of equidistribution at v-bit accuracy* $k(v)$ is defined as the DE of the v-bit sequence, obtained by extracting the v MSBs from each of w-bit integers. If the defect ratio is $1/(P+1)$, then there is an upper bound

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

The gap between the realized $k(v)$ and the upper bound is called the dimension defect at v of the sequence, and denoted by

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

The summation of all the dimension defects at $1 \leq v \leq 32$ is called the total dimension defect, denoted by Δ.

There is a difficulty in computing $k(v)$ when a 128-bit integer generator is used as a 32-bit (or 64-bit) integer generator. We solved this by using a weighted lattice method, generalising [3]. Since the algorithm involves rather complicated mathematics, we omit it here (we plan another article for this).

The algorithm gives a (rather tight) lower bound $k'(v)$ of $k(v)$ for each v, and $k'(v) \leq \lceil 19937/v \rceil$ holds for SFMT19937. Consequently, we redefine the dimension defect for SFMT19937 by

$$d(v) := \lceil 19937/v \rceil - k'(v) \text{ and } \varDelta := \sum_{v=1}^{w} d(v).$$

The meaning of $k'(v)$ and a justification for this definition will be explained in the planned article.

3 Comparison of speed

We compared two algorithms: MT19937 and SFMT19937, with implementations with and without SIMD instructions.

We measured the speeds for four different CPUs: Pentium M 1.4GHz, Pentium IV 3GHz, AMD Athlon 64 3800+, and PowerPC G4 1.33GHz. In returning the random values, we use two different methods. One is sequential generation, where one 32-bit random integer is returned for one call. The other is block generation, where an array of random integers is generated for one call (for detail, see §7 below).

We measured the consumed CPU time in second, for 10^8 generations of 32-bit integers. More precisely, in case of the block generation, we generate 10^5 of 32-bit random integers by one call, and it is iterated for 10^3 times. For sequential generation, the same 10^8 32-bit integers are generated, one per a call. To avoid the function call, the code uses an inline declaration. More precisely: we used the inline declaration `inline` and unsigned 32-bit, 64-bit integer types `uint32_t`, `uint64_t` defined in INTERNATIONAL STANDARD ISO/IEC 9899 : 1999(E) Programming Language-C, Second Edition (which we shall refer to as C99 in the rest of this article). Implementations without SIMD are written in C99, whereas those with SIMD use some standard SIMD extension of C99 supported by the compilers icl (Intel C compiler) and gcc.

Table 1 summarises the speed comparisons. The first four lines list the CPU time (in second) needed to generate 10^8 32-bit integers, for a Pentium-M CPU with the Intel C/C++ compiler. The first line lists the seconds for the block-generation scheme. The second line shows the ratio of CPU time to that of SFMT(SIMD). Thus, SFMT coded in SIMD is 1.94 times faster than MT coded in SIMD, and 3.53 times faster than MT without SIMD. The third line lists the seconds for the sequential generation scheme. The fourth line lists the ratio, with the basis taken at SFMT(SIMD) block-generation (not sequential). Thus, the block-generation of SFMT(SIMD) is 2.78 times faster than the sequential-generation of SFMT(SIMD).

Roughly speaking, in the block generation, SFMT(SIMD) is twice as fast as MT(SIMD), and four times faster than MT without using SIMD. Even in the sequential generation case, SFMT(SIMD) is still considerably faster than MT(SIMD).

Table 2 lists the CPU time for generating 10^8 32-bit integers, for four PRNGs from the GNU Scientific Library and two recent generators. They

CPU/compiler	return	MT	MT(SIMD)	SFMT	SFMT(SIMD)
Pentium-M	block	1.131	0.620	0.721	0.320
1.4GHz	(ratio)	3.53	1.94	2.25	1.00
Intel C/C++	seq	1.402	1.181	1.321	0.891
ver. 9.0	(ratio)	4.38	3.69	4.13	2.78
Pentium-IV	block	0.625	0.390	0.484	0.219
3GHz	(ratio)	2.85	1.78	2.21	1.00
Intel C/C++	seq	1.031	0.906	0.812	0.562
ver. 9.0	(ratio)	4.71	4.14	3.71	2.57
Athlon 64 3800+	block	0.670	0.370	0.330	0.140
2.4GHz	(ratio)	4.79	2.64	2.36	1.00
gcc	seq	0.540	0.390	0.560	0.390
ver. 4.0.2	(ratio)	3.86	2.79	4.00	2.79
PowerPC G4	block	1.240	0.550	1.090	0.220
1.33GHz	(ratio)	5.64	2.50	4.95	1.00
gcc	seq	1.030	0.600	1.150	0.550
ver. 4.0.0	(ratio)	4.68	2.73	5.23	2.50

Table 1. The CPU time (sec.) for 10^8 generations of 32-bit integers, for four different CPUs and two different return-value methods. The ratio to the SFMT coded in SIMD is listed, too.

CPU	return	mrg	rand48	rand	random256g2	well	xor3
Pentium M	block	3.314	1.522	0.450	0.360	1.792	0.300
	seq	3.425	1.361	0.681	1.002	2.032	1.142
Pentium IV	block	2.594	1.281	0.406	0.203	1.016	0.328
	seq	2.563	1.172	0.610	0.562	1.281	0.812
Athlon	block	2.050	1.580	0.200	0.200	0.950	0.370
	seq	2.080	1.420	0.200	0.210	1.380	0.510
PowerPC	block	2.620	1.200	0.480	0.730	1.820	0.720
	seq	2.420	0.820	0.370	0.990	3.340	1.010

Table 2. The CPU time (sec.) for 10^8 generations of 32-bit integers, by six other PRNGs.

are re-coded with inline specification. Generators examined were: a multiple recursive generator `mrg` [6], linear congruential generators `rand48` and `rand`, a lagged fibonacci generator `random256g2`, a WELL generator `well` (WELL19937c in [12]), and an optimized XORSHIFT generator `xor3` [11] [7]. The table shows that SFMT(SIMD) is faster than these PRNGs, except for the lagged-fibonacci generator `random256g2` with a Pentium IV (but this method is known to have poor randomness, cf. [10]), and `xor3` with a Pentium-M.

4 Dimension of equidistribution

The total dimension defect Δ of SFMT19937 as a 32-bit integer generator is 4188, which is smaller than 6750 of MT19937. Table 3 lists the dimension defects $d(v)$ of SFMT19937 (as a 32-bit integer generator) and MT19937 for $v = 1, 2, \ldots, 32$. SFMT has smaller values of the defect $d(v)$ at 26 values of v. The converse holds for 6 values of v, but the difference is small.

v	MT	SFMT	v	MT	SFMT	v	MT	SFMT	v	MT	SFMT
$d(1)$	0	0	$d(9)$	346	1	$d(17)$	549	543	$d(25)$	174	173
$d(2)$	0	*2	$d(10)$	124	0	$d(18)$	484	478	$d(26)$	143	142
$d(3)$	405	1	$d(11)$	564	0	$d(19)$	426	425	$d(27)$	115	114
$d(4)$	0	*2	$d(12)$	415	117	$d(20)$	373	372	$d(28)$	89	88
$d(5)$	249	2	$d(13)$	287	285	$d(21)$	326	325	$d(29)$	64	63
$d(6)$	207	0	$d(14)$	178	176	$d(22)$	283	282	$d(30)$	41	40
$d(7)$	355	1	$d(15)$	83	*85	$d(23)$	243	242	$d(31)$	20	19
$d(8)$	0	*1	$d(16)$	0	*2	$d(24)$	207	206	$d(32)$	0	*1

Table 3. Dimension defects $d(v)$ of MT19937 and SFMT19937 as a 32-bit integer generator. The mark * means that MT has a smaller defect than SFMT at that accuracy.

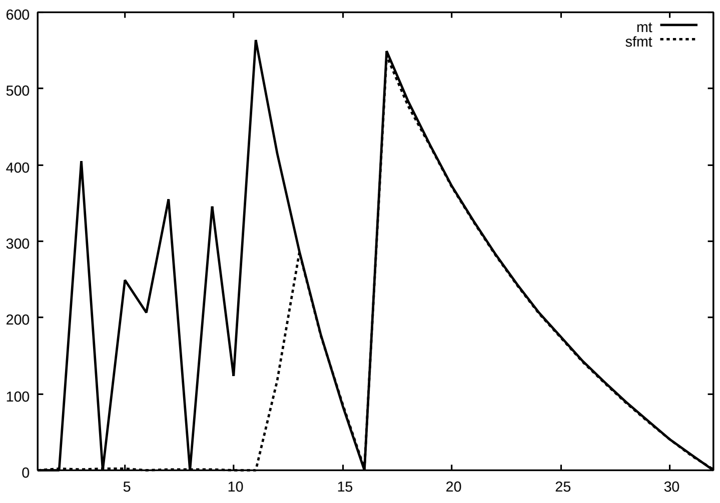

Fig. 2. The dimension defects $d(v)$ of MT (real line) and SFMT (dotted) for $v = 1, 2, \ldots, 32$.

We also computed the dimension defects of SFMT19937 as a 64-bit (128-bit) integer generator, and the total dimension defect Δ is 14089 (28676, respectively). In some applications, the distribution of LSBs is important. To check them, we inverted the order of the bits (i.e. the i-th bit is exchanged with the $(w-i)$-th bit) in each integer, and computed the total dimension defect. It is 10328 (21337, 34577, respectively) as a 32-bit (64-bit, 128-bit, respectively) integer generator. Throughout the experiments, $d(v)$ is very small for $v \leq 10$. We consider that these values are satisfactorily small, since they are comparable with MT for which no statistical problem related to the dimension defect has been reported, as far as we know.

5 Recovery from 0-excess states

LFSR with a sparse feedback function g has the following phenomenon: if the bits in the state space contain too many 0's and few 1's (called a 0-excess state), then this tendency continues for considerable generations, since only a small part is changed in the state array at one generation, and the change is not well-reflected to the next generation because of the sparseness.

We measure the recovery time from 0-excess states, by the method introduced in [12], as follows.

1. Choose an initial state with only one bit being 1.
2. Generate k pseudorandom numbers, and discard them.
3. Compute the ratio of 1's among the next 32000 bits of outputs (i.e., in the next 1000 pseudorandom 32-bit integers).
4. Let γ_k be the average of the ratio over all such initial states.

We draw graphs of these ratio γ_k $(1 \leq k \leq 20000)$ in Figure 3 for the following generators: (1) WELL19937c, (2) PMT19937 [13], (3) SFMT19937, and (4) MT19937. Because of its dense feedback, WELL19937c shows the fastest

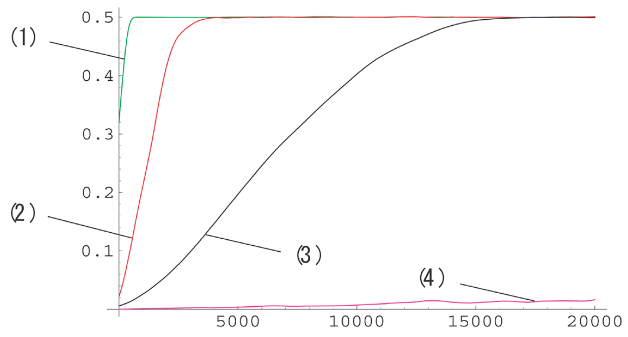

Fig. 3. γ_k $(k = 0, \ldots, 20000)$: Starting from extreme 0-excess states, discard the first k outputs and then measure the ratio γ_k of 1's in the next 1000 outputs. In the order of the recovery speed: (1) WELL19937c, (2) PMT19937, (3) SFMT19937, and (3) MT19937.

recovery among the compared. SFMT is better than MT, since its recursion refers to the previously-computed words (i.e., `W[N-1]` and `W[N-2]`) that acquire new 1s, while MT refers only to the words generated long before (i.e., `W[M]` and `W[0]`). PMT19937 shows faster recovery than SFMT19937, since PMT19937 has two feedback loops (hence the name of Pulmonary Mersenne Twister).

The speed of recovery from 0-excess states is a trade-off with the speed of generation. Such 0-excess states will not happen practically, since the probability that 19937 random bits have less than 19937×0.4 of 1's is about 5.7×10^{-177}. The only plausible case is that a poor initialization scheme gives a 0-excess initial state. In a typical simulation, the number of initializations is far smaller than the number of generations, therefore we may spend more CPU time in the initialization than the generation. Once we avoid the 0-excess initial state by a well-designed initialization, then the recovery speed does not matter, in a practical sense. Consequently, the slower recovery of SFMT compared to WELL is not an issue, under the assumption that a good initialization scheme is provided.

6 Computation of the Period

LFSR by the recursion (1) may be considered as an automaton, with the state space $S = (\mathbb{F}_2^w)^N$ and the state transition function $f : S \to S$ given by $(\mathbf{w}_0, \dots, \mathbf{w}_{N-1}) \mapsto (\mathbf{w}_1, \dots, \mathbf{w}_{N-1}, g(\mathbf{w}_0, \dots, \mathbf{w}_{N-1}))$. As a w-bit integer generator, the output function is $o : S \to \mathbb{F}_2^w,\ (\mathbf{w}_0, \dots, \mathbf{w}_{N-1}) \mapsto \mathbf{w}_0$.

Let χ_f be the characteristic polynomial of $f : S \to S$. If χ_f is primitive, then the period of the state transition takes the maximal value $2^{\dim(S)} - 1$ [5, §3.2.2]. However, to check the primitivity, we need the integer factorization of this number, which is usually impossible for $\dim(S) = nw > 10000$. On the other hand, the primarity test is much easier than the factorization, so many huge primes of the form $2^p - 1$ have been found. Such a prime is called a Mersenne prime, and p is called the Mersenne exponent, which itself is a prime.

MT and WELL discard some fixed r-bits from the array S, so that $nw - r$ is a Mersenne exponent. Then, the primitivity of χ_f is easily checked by the algorithm in [5, §3.2.2], avoiding the integer factorization.

SFMT adopted another method to avoid the integer factorization, the reducible transition method (RTM), which uses a reducible characteristic polynomial. This idea appeared in [4] [1][2], and applications in the present context are discussed in detail in another article [13], therefore we only briefly recall it.

Let p be the Mersenne exponent, and $N := \lceil p/w \rceil$. Then, we randomly choose parameters for the recursion of LFSR (1). By the Berlekamp-Massey Algorithm, we compute the minimal polynomial of the (bit) sequence generated by the linear recursion. By taking the least common multiple for several initial values, we have the minimal polynomial of the transition f. With high probability its degree coincides with $\dim(S)$, and if this occurs it is the characteristic polynomial χ_f. (Note that a direct computation of $\det(tI - f)$ is time-consuming because $\dim(S) = 19968$.)

By using a sieve, we remove all factors of small degree from χ_f, until we know that it has no irreducible factor of degree p, or we know that it has a factor of degree p. In the latter case, the factor is passed to the primitivity test in [5, §3.2.2].

Suppose that we found a recursion with a factor of desired degree p in $\chi_f(t)$. Then, we have a factorization

$$\chi_f = \phi_p \phi_r,$$

where ϕ_p is a primitive polynomial of degree p and ϕ_r is a polynomial of degree $r = wN - p$. These are coprime, if we assume $p > r$. By linear algebra, we have a decomposition into f-invariant subspaces

$$S = V_p \oplus V_r,$$

where $f|_{V_p}$ has the minimal polynomial ϕ_p and $f|_{V_r}$ has the characteristic polynomial ϕ_r. For any element $s \in S$, we denote $s = s_p + s_r$ for the corresponding decomposition.

The period length of the state transition is the least common multiple of that started from s_p and that started from s_r. Hence, if $s_p \neq 0$, then the period is a nonzero multiple of $2^p - 1$. By these computations, we checked the following.

Proposition 1. *The period of SFMT19937 as 128-bit integer generator is a nonzero multiple of* $2^{19937} - 1$*, if the 32 MSBs of* $\mathbf{w}_0$ *is set to the value* `6d736d6d` *in hexadecimal form.*

The choice of the value for $\mathbf{w}_0$ is to assure that $s_p \neq 0$.

Remark 1. The number of non-zero terms in $\chi_f(t)$ is an index measuring the amount of bit-mixing. In the case of SFMT19937, the number of nonzero terms is 6711, which is much larger than 135 of MT, but smaller than 8585 of WELL19937c.

7 Block-generation

In the block-generation scheme, the user of the PRNG specifies an array of w-bit integers of the length L, where $w = 32$, 64 or 128 and L is specified by the user. In the case of SFMT19937, L should be a multiple of $128/w$ and no less than $N \times 128/w$, since the array needs to accommodate the state space (note that $N = 156$). By calling the block generation function with the pointer to this array, w, and L, the routine fills up the array with pseudorandom integers, as follows. SFMT19937 keeps the state space S in an internal array of 128-bit integers of length 156. We concatenate this state array with the user-specified array, using the indexing technique. Then, the routine generates 128-bit integers in the user-specified array by recursion (2), as described in Figure 4, until it fills up the array. The last 156 128-bit integers are copied back to the internal array of SFMT19937. This makes the generation much faster than sequential generation (i.e., one generation per one call) as shown in Table 1.

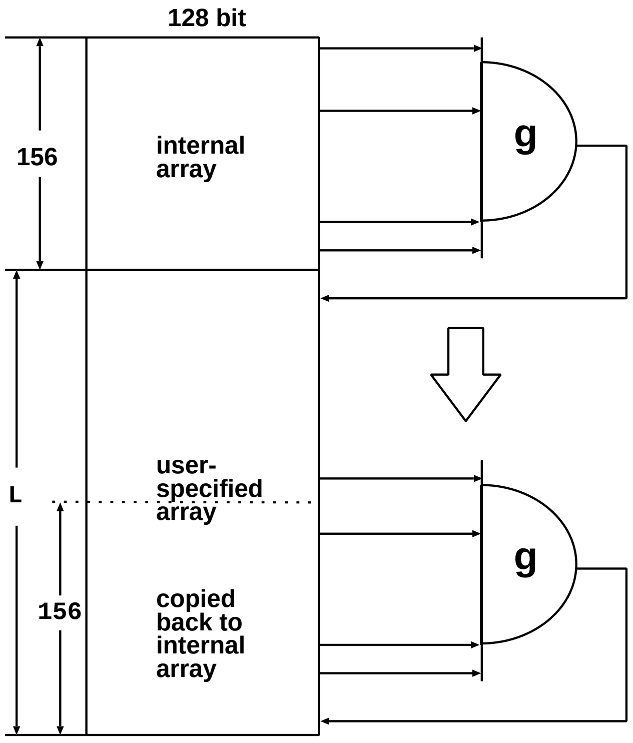

Fig. 4. Block-generation scheme

8 Concluding remarks

We proposed SFMT pseudorandom number generator, which is a very fast generator with satisfactorily high dimensional equidistribution property.

8.1 Trade-off between speed and quality

It is difficult to measure the generation speed of a PRNG in a fair way, since it depends heavily on the circumstance. The WELL [12] generators have the best possible dimensions of equidistribution (i.e. $\Delta = 0$) for various periods ($2^{1024}-1$ to $2^{19937}-1$). If we use the function call to PRNG for each generation, then a large part of the CPU time is consumed for handling the function call, and in the experiments in [12] or [11], WELL is not much slower than MT. On the other hand, if we avoid the function call, WELL is much slower than MT as seen in Table 1.

Since $\Delta = 0$, WELL has a better quality than MT or SFMT in a theoretical sense. However, one may argue whether this difference is observable or not. In the case of an $\mathbb{F}_2$-linear generator, the dimension of equidistribution $k(v)$ of v-bit accuracy means that there is no constant linear relation among the kv bits, but there exists a linear relation among the $(k+1)v$ bits, where kv bits ($(k+1)v$ bits) are taken from all the consecutive k integers ($k+1$ integers, respectively), i.e., v MSBs of each of the integers. However, the existence of a linear relation does not necessarily mean the existence of some observable bias. According to [9], it requires 10^{28} samples to detect an $\mathbb{F}_2$-linear relation with 15 (or more) terms among 521 bits, by a standard statistical test. If the number of total bits is increased, the necessary sample size is increased. Thus, it seems that $k(v)$ of SFMT19937 is sufficiently large, far beyond the level of the observable bias. On the other hand, the speed of the generator is observable. Thus, SFMT focuses more on the speed, for applications that require fast generations.

8.2 Trade-off between speed and portability

There is a trade-off between the speed and portability. We prepare (1) a standard C code of SFMT, which uses functions specified in C99 only, (2) an optimized C code for Intel Pentium SSE2, and (3) an optimized C code for PowerPC AltiVec. The optimized codes require icl (Intel C Compiler) or gcc compiler with suitable options. We will put the newest version of the codes in a link from the homepage of [8], together with some notes on compiling with SIMD.

There is a problem of the endian when 128-bit integers are converted into 32-bit integers. When a 128-bit integer is stored as a quadruple of 32-bit integers, in a little endian system such as Pentium, the 32 MSBs come first. On the other hand, in a big endian system such as PowerPC, the 32 LSBs come first. The explanation above is based on the former. To assure the exact same outputs for both endian systems as 32-bit integer generators, in the SIMD implementation for PowerPC, the recursion (2) is considered as a recursion on quadraples of 32-bit integers, rather than 128-bit integers, so that the content of the state array coincides both for little and big endian systems, as an array of 32-bit integers (not as 128-bit integers). Then, shift operations on 128-bit integers in PowerPC differs from those of Pentium. Fortunately, PowerPC supports arbitrary permutations of 16 blocks of 8-bit integers in a 128-bit register, which emulates the Pentium's shift by a multiple of 8.

References

1. R.P. Brent and P. Zimmermann. Random number generators with period divisible by a mersenne prime. In *Computational Science and its Applications - ICCSA 2003*, volume 2667, pages 1–10, 2003.
2. R.P. Brent and P. Zimmermann. Algorithms for finding almost irreducible and almost primitive trinomials. *Fields Inst. Commun.*, 41:91–102, 2004.
3. R. Couture, P. L'Ecuyer, and S. Tezuka. On the distribution of k-dimensional vectors for simple and combined tausworthe sequences. *Math. Comp.*, 60(202):749–761, 1993.
4. M. Fushimi. Random number generation with the recursion $x_t = x_{t-3p} \oplus x_{t-3q}$. *Journal of Computational and Applied Mathematics*, 31:105–118, 1990.
5. Donald E. Knuth. *The Art of Computer Programming. Vol.2. Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 3 edition, 1997.
6. P. L'Ecuyer. A search for good multiple recursive random number genarators. *ACM Transactions on Modeling and Computer Simulation*, 3(2):87–98, apr 1993.
7. G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.
8. M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1):3–30, January 1998. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`.
9. M. Matsumoto and T. Nishimura. A nonempirical test on the weight of pseudorandom number generators. In *Monte Carlo and Quasi-Monte Carlo methods 2000*, pages 381–395. Springer-Verlag, 2002.
10. M. Matsumoto and T. Nishimura. Sum-discrepancy test on pseudorandom number generators. *Mathematics and Computers in Simulation*, 62(3-61):431–442, 2003.
11. F. Panneton and P. L'Ecuyer. On the Xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation*, 15(4):346–361, 2005.
12. F. Panneton, P. L'Ecuyer, and M. Matsumoto. Improved long-period generators based on linear reccurences modulo 2. *ACM Transactions on Mathematical Software*, 32(1):1–16, 2006.
13. M. Saito, H. Haramoto, F. Panneton, T. Nishimura, and M. Matsumoto. Pulmonary LFSR: pseudorandom number generators with multiple feedbacks and reducible transitions. 2006. submitted.