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

Mutsuo Saito[1] and Makoto Matsumoto[2]

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

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 optimised MT using SIMD operations. Moreover, the dimension of equidistribution of SFMT is better than MT.

We also introduce a block-generation function, which fills an array of 32-bit integers in one call. 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. The implemented C-codes are downloadable from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html.

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) [11], 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, such as Pentium and PowerPC, have changed. They have Single 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.

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

In this article, we propose an MT-like pseudorandom number generator that makes full use of these new features: SFMT, a SIMD-oriented Fast 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. 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 in one call. Actually some demanding users re-coded MT to avoid the function call; see the homepage of [11]. 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 a 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 bits $\{0,1\}$ 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

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

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) [11] is an example with

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

where $(\mathbf{w}_0|\mathbf{w}_1)$ denotes the concatenation of the $32 - r$ most significant bits (MSBs) of $\mathbf{w}_0$ and the r least significant bits (LSBs) of $\mathbf{w}_1$, 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 $\mathbf{x}_0T, \mathbf{x}_1T, \mathbf{x}_2T, \ldots$ (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

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

where $\mathbf{w}_0, \mathbf{w}_M, \ldots$ 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 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.
- $\mathbf{w}C := (\mathbf{w} \overset{128}{>>} 8)$.
  This 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 $2^{19937} - 1$.

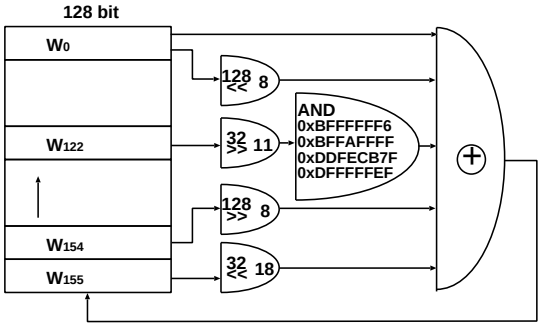

Fig. 1. A circuit-like description of SFMT19937.

2.4 Endianness

Let $\mathbf{x}[0..3]$ be an array of 32-bit integers of size four. There are two natural ways to convert the array to a 128-bit integer. One is to concatenate in the

order of $\mathbf{x}[3]\mathbf{x}[2]\mathbf{x}[1]\mathbf{x}[0]$, from MSBs to LSBs, which is called the little-endian system, adopted in Pentium. The converse is the big-endian system adopted in PowerPC, see [18].

The descriptions in this article is based on the former. To assure the portability for both endian systems, we implemented two codes: one is for little-endian system (SSE2 of Pentium) and the other is for big-endian system (AltiVec of PowerPC), to assure the exactly same outputs as 32-bit integer generators. In the latter code, the recursion (2) is considered as a recursion on quadruples 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). Thus, shift-operations on 128-bit integers in the little-endian system is different from that in the big-endian system. PowerPC supports arbitrary permutations of 16 blocks of 8-bit integers in a 128-bit register, which can emulate the shift in (2).

2.5 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, wL should be a multiple of 128 and no less than $N \times 128$, 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 2, 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.

3 How to select the recursion and parameters.

We wrote a code to compute the period and the dimensions of equidistribution (DE, see §3.2). Then, we searched for a recursion with good DE admitting a fast implementation.

3.1 Computation of the Period

An LFSR that obeys 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 $(\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}) \mapsto (\mathbf{w}_1, \ldots, \mathbf{w}_{N-1}, g(\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}))$. As a w-bit integer generator, the output function is $o : S \to \mathbb{F}_2^w,\ (\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}) \mapsto \mathbf{w}_0$.

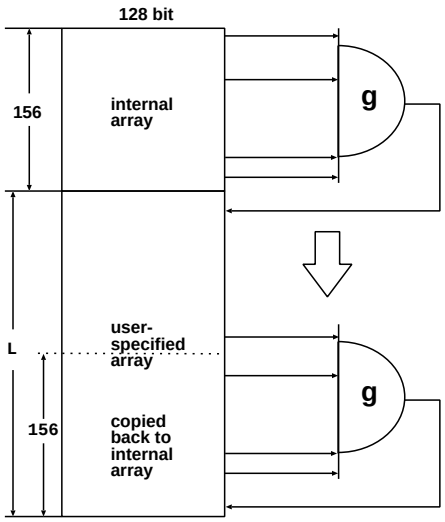

Fig. 2. Block-generation scheme

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 often hard 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[15] discard r specific 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 with a large primitive factor. This idea appeared in [4] [1][2], and applications in the present context are discussed in detail in another article [16], 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 applying the Berlekamp-Massey Algorithm to the output sequence, we obtain $\chi_f(t)$. (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 that it has a (possibly reducible) factor of degree p. In the latter case, the factor is passed to the primitivity test described in [5, §3.2.2].

Suppose that we found a recursion with an irreducible 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, since we assume $p > r$. Let $\mathrm{Ker}(g)$ denote the kernel of a linear transformation g. By putting $V_p := \mathrm{Ker}\,(\phi_p(f))$ and $V_r := \mathrm{Ker}\,(\phi_r(f))$, we have a decomposition into f-invariant subspaces

$$S = V_p \oplus V_r \quad (\dim V_p = p,\ \dim V_r = r).$$

Note that the characteristic polynomial of the restriction f_p of f to V_p is $\phi_p(t)$, and that of the restriction f_r to V_r is $\phi_r(t)$. For any state $s \in S$, we denote $s = s_p + s_r$ for the corresponding decomposition with $s_p \in V_p$ and $s_r \in V_r$. Then, the k-th state $f^k(s)$ is equal to $f_p^k(s_p) + f_r^k(s_r)$. This implies that the automaton is equivalent to the sum of two automata $f_p : V_p \to V_p$ and $f_r : V_r \to V_r$. To combine two linear automata by sum is well-studied as combined Tausworthe generators or combined LFSRs, see [3] [7] [8]. Their purpose is to obtain a good PRNG from several simple generators, which is different from ours.

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 $2^p - 1$. We checked the following.

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

This value of $\mathbf{w}_0$ assures that $s_p \neq 0$, see [16] for a way to find such a value.

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 [15].

3.2 Computation of the dimension of equidistribution

We briefly recall the definition of dimension of equidistribution (cf. [3][7]).

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 generalize 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

$$W_k(\chi) = (m \cdot \mathbb{F}_2^{vk}) \setminus \{\mathbf{0}\},$$

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 $D \subset (m \cdot \mathbb{F}_2^{vk})$ such that*

$$W_k(\chi) = (m \cdot \mathbb{F}_2^{vk}) \setminus D,$$

we say that χ is k-dimensionally equidistributed with defect ratio $\#(D)/\#(m \cdot \mathbb{F}_2^{vk})$, 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 the w-bit integers. If the defect ratio is $1/(P+1)$, then there is an upper bound

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

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 $1 \le v \le 32$ 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. 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 $\mathbf{x}_0[0], \mathbf{x}_0[1], \mathbf{x}_0[2], \mathbf{x}_0[3], \mathbf{x}_1[0], \mathbf{x}_1[1], \ldots$, where $\mathbf{x}[0]$ is the 32 LSBs of $\mathbf{x}$, $\mathbf{x}[1]$ is the 33rd–64th bits, $\mathbf{x}[2]$ is the 65rd–96th bits, and $\mathbf{x}[3]$ is the 32 MSBs.

Then, we need to modify the model automaton as follows. The state space is $S' := S \times \{0,1,2,3\}$, the state transition function $f' : S' \to S'$ is

$$f'(s,i) := \begin{cases} (s, i+1) \ (\text{ if } i < 3), \\ (f(s), 0) \ (\text{ if } i = 3) \end{cases}$$

and the output function is

$$o' : S' \to \mathbb{F}_2^{32}, \ ((\mathbf{w}_0, \dots, \mathbf{w}_{N-1}), i) \mapsto \mathbf{w}_0[i].$$

We fix $1 \le v \le w$, and let $o_k(s,i)$ be the k-tuple of the v MSBs of the consecutive k-outputs from the state (s,i).

Proposition 2. *Assume that f is bijective. Let $k' = k'(v)$ denote the maximum k such that*

$$o_k(-,i) : V_p \to \mathbb{F}_2^{kv}, \quad s \mapsto o_k(s,i) \tag{3}$$

are surjective for all $i = 0,1,2,3$. Take an initial state s satisfying $s_p \neq 0$. Then, the 32-bit output sequence is at least $k'(v)$-dimensionally equidistributed with v-bit accuracy with defect ratio 2^{-p}.

Moreover, if $4 < k'(v) + 1$, then for any initial state with $s = s_p \neq 0$ (hence $s_r = 0$), the dimension of equidistribution with defect ratio 2^{-p} is exactly $k'(v)$.

Proof. Take $s \in S$ with $s_p \neq 0$. Then, the orbit of s by f has the form of $(V_p - \{0\}) \times U \subset V_p \times V_r$, since $p > r$ and $2^p - 1$ is a prime. The surjectivity of the linear mapping $o_{k'}(-,i)$ implies that the image of

$$o_{k'}(-,i) : V_p \times U \to \mathbb{F}_2^{kv}$$

is $m \cdot \mathbb{F}_2^{kv}$ as a multi-set for some m. The defect comes from $0 \in V_p$, whose ratio in V_p is 2^{-p}. Then the first statement follows, since $W_{k'}(\chi)$ is the union of the images $o_{k'}(-,i)((V_p - \{0\}) \times U)$ for $i = 0,1,2,3$.

For the latter half, we define L_i as the multiset of the image of $o_{k'+1}(-,i) : V_p \to \mathbb{F}_2^{(k'+1)v}$. Because of $s_r = 0$, we have $U = \{0\}$, and the union of $(L_i - \{0\})$ $(i = 0,1,2,3)$ as a multi-set is $W_{k'+1}(\chi)$. If the sequence is $(k'+1)$-dimensionally equidistributed, then the multiplicity of each element in $W_{k'+1}(\chi)$ is at most $2^p \times 4/2^{(k'+1)v}$.

On the other hand, the multiplicity of an element in L_i is equal to the cardinality of the kernel of $o_{k'+1}(-,i)$. Let d_i be its dimension. Then by the dimension theorem, we have $d_i \ge p - (k'+1)v$, and the equality holds if and only if $o_{k'+1}(-,i)$ is surjective. Thus, if there is a nonzero element $x \in \cap_{i=0}^3 L_i$, then its multiplicity in $W_{k'+1}(\chi)$ is no less than $4 \times 2^{p-(k'+1)v}$, and since one of $o_{k'+1}(-,i)$ is not surjective by the definition of k', its multiplicity actually exceeds $4 \times 2^{p-(k'+1)v}$, which implies that the sequence is not $(k'+1)$-dimensionally equidistributed, and the proposition follows. Since the codimension of L_i is at most v, that of $\cap_{i=0}^3 L_i$ is at most $4v$. The assumed inequality on k' implies the existence of nonzero element in the intersection.

The dimension of equidistribution $k(v)$ depends on the choice of the initial state s. The above proposition implies that $k'(v)$ coincides with $k(v)$ for the worst choice of s under the condition $s_p \neq 0$. Thus, we adopt the following definition (analogously to t_l in [7]).

Definition 3. *Let k be the maximum such that (3) is satisfied. We call this the dimension of equidistribution of v-bit accuracy, and denote it simply by $k(v)$. We have an upper bound $k(v) \leq \lfloor p/v \rfloor$.*

We define the dimension defect at v by

$$d(v) := \lfloor p/v \rfloor - k(v) \text{ and } \Delta := \sum_{v=1}^{w} d(v).$$

We may compute $k(v)$ by standard linear algebra. We used a more efficient algorithm based on a weighted norm, generalizing [3]. This will be written somewhere else, because of lack of space.

4 Comparison of speed

We compared two algorithms: MT19937 and SFMT19937, with implementations using and without using 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 used 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 (cf. [5]). For detail, see §2.5 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 this is iterated for 10^3 times. For sequential generation, the same 10^8 32-bit integers are generated, one per call. We used the inline declaration `inline` to avoid the function call, 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 seconds) 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 2.10 times faster than MT coded in SIMD, and 3.77 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 2.00 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).

CPU/compiler	return	MT	MT(SIMD)	SFMT	SFMT(SIMD)
Pentium-M	block	1.122	0.627	0.689	0.298
1.4GHz	(ratio)	3.77	2.10	2.31	1.00
Intel C/C++	seq	1.511	1.221	1.017	0.597
ver. 9.0	(ratio)	5.07	4.10	3.41	2.00
Pentium IV	block	0.633	0.391	0.412	0.217
3GHz	(ratio)	2.92	1.80	1.90	1.00
Intel C/C++	seq	1.014	0.757	0.736	0.412
ver. 9.0	(ratio)	4.67	3.49	3.39	1.90
Athlon 64 3800+	block	0.686	0.376	0.318	0.156
2.4GHz	(ratio)	4.40	2.41	2.04	1.00
gcc	seq	0.756	0.607	0.552	0.428
ver. 4.0.2	(ratio)	4.85	3.89	3.54	2.74
PowerPC G4	block	1.089	0.490	0.914	0.235
1.33GHz	(ratio)	4.63	2.09	3.89	1.00
gcc	seq	1.794	1.358	1.645	0.701
ver. 4.0.0	(ratio)	7.63	5.78	7.00	2.98

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	3.277	1.417	0.453	0.230	1.970	0.296
	seq	3.255	1.417	0.527	0.610	2.266	1.018
Pentium IV	block	2.295	1.285	0.416	0.121	0.919	0.328
	seq	2.395	1.304	0.413	0.392	1.033	0.702
Athlon	block	1.781	0.770	0.249	0.208	0.753	0.294
	seq	1.798	0.591	0.250	0.277	0.874	0.496
PowerPC	block	2.558	1.141	0.411	0.653	1.792	0.618
	seq	2.508	1.132	0.378	1.072	1.762	1.153

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

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 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 [15]), and a XORSHIFT generator `xor3` [14] [10]. The table shows that SFMT(SIMD) is faster than these PRNGs, except for the outdated linear congruential generator `rand`, the lagged-fibonacci generator `random256g2` (which is known to have poor randomness, cf. [13]), and `xor3` with a Pentium-M.

5 Dimension of equidistribution

Table 3 lists the dimension defects $d(v)$ of SFMT19937 (as a 32-bit integer generator) and of 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. The total dimension defect Δ of SFMT19937 as a 32-bit integer generator is 4188, which is smaller than the total dimension defect 6750 of MT19937.

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.

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 deviation related to the dimension defect has been reported, as far as we know.

6 Recovery from 0-excess states

For an LFSR with a sparse feedback function g, we observe 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 many steps, since only a small part is changed in the state array at one step, and the change is not well-reflected to the next setp because of the sparseness.

We measure the recovery time from 0-excess states, by the method introduced in [15], 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 ($1 \le k \le 20000$) in Figure 3 for the following generators: (1) WELL19937c, (2) PMT19937 [16], (3) SFMT19937, and (4) MT19937. Because of its dense feedback, WELL19937c shows the fastest

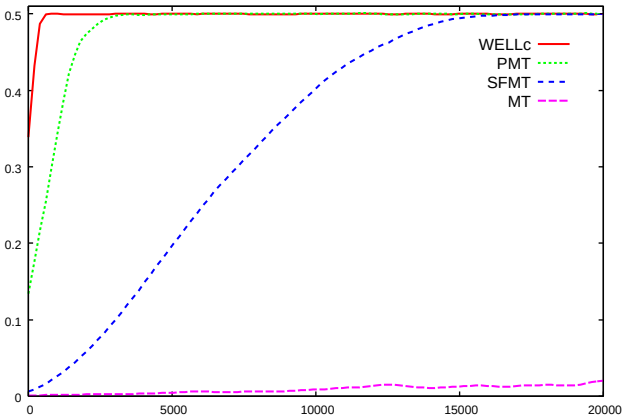

Fig. 3. γ_k ($k = 0, \ldots, 20000$): Starting from extreme 0-excess states, discard the first k outputs and then measure the ratio γ_k of 1's in the next 1000 outputs. In the order of the recovery speed: (1) WELL19937c, (2) PMT19937, (3) SFMT19937, and (3) MT19937.

recovery among the compared generators. SFMT is better than MT, since its recursion refers to two most recently computed words (`W[N-1]` and `W[N-2]`) that acquire new 1s, while MT refers only to the words generated long before (`W[M]` and `W[0]`). PMT19937 shows faster recovery than SFMT19937, since PMT19937 has two feedback loops. 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 would be that a poor initialization scheme gives a 0-excess initial state (or gives two initial states whose Hamming distance is too small). 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. Under the assumption that a good initialization scheme is provided, the slower recovery of SFMT compared to WELL would perhaps not be a great issue.

7 Concluding remarks

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

It is difficult to measure the generation speed of a PRNG in a fair way, since it depends heavily on the circumstances. The WELL [15] 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 the PRNG for each generation, then a large part of the CPU time is consumed for handling the function call, and in the experiments in [15] or [14], WELL is not much slower than MT. On the other hand, if we avoid the function call, WELL is slower than MT for some CPUs, 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) by extracting the v MSBs from each. However, the existence of a linear relation does not necessarily mean the existence of some observable bias. According to [12], it requires 10^{28} samples to detect an $\mathbb{F}_2$-linear relation with 15 (or more) terms among 521 bits, by weight distribution test. If the number of bits is increased, the necessary sample size is increased rapidly. 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. (Note: the referee pointed out that statistical tests based on the rank of $\mathbb{F}_2$-matrix is sensitive to the linear relations [9], so the above observation is not necessarily true.)

There is a trade-off between the speed and portability. We prepared (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 the icl (Intel C Compiler) or gcc compiler with suitable options. We had put and will keep the newest version of the codes in the homepage [17].

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. D. E. Knuth. *The Art of Computer Programming. Vol.2. Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 3rd 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, April 1993.
7. P. L'Ecuyer. Maximally equidistributed combined tausworthe generators. *Math. Comp.*, 65(213):203–213, 1996.
8. P. L'Ecuyer. Tables of maximally equidistributed combined lfsr generators. *Math. Comp.*, 68(225):261–269, 1999.
9. P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 2006. to appear.
10. G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.
11. 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`.
12. 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.
13. M. Matsumoto and T. Nishimura. Sum-discrepancy test on pseudorandom number generators. *Mathematics and Computers in Simulation*, 62(3-6):431–442, 2003.
14. F. Panneton and P. L'Ecuyer. On the Xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation*, 15(4):346–361, 2005.
15. 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.
16. M. Saito, H. Haramoto, F. Panneton, T. Nishimura, and M. Matsumoto. Pulmonary LFSR: pseudorandom number generators with multiple feedbacks and reducible transitions. 2006. submitted.
17. M. Saito and M. Matsumoto. SFMT Homepage. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html`.
18. Endianness from Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Endianness`.