Multiplicative, Congruential Random-Number Generators with Multiplier $\pm 2^{k1} \pm 2^{k2}$ and Modulus $2^p - 1$

PEI-CHI WU
National Chiao Tung University

The demand for random numbers in scientific applications is increasing. However, the most widely used multiplicative, congruential random-number generators with modulus $2^{31} - 1$ have a cycle length of about 2.1×10^9. Moreover, developing portable and efficient generators with a larger modulus such as $2^{61} - 1$ is more difficult than those with modulus $2^{31} - 1$. This article presents the development of multiplicative, congruential generators with modulus $m = 2^p - 1$ and four forms of multipliers: $2^{k1} - 2^{k2}$, $2^{k1} + 2^{k2}$, $m - 2^{k1} + 2^{k2}$, and $m - 2^{k1} - 2^{k2}$, $k1 > k2$. The multipliers for modulus $2^{31} - 1$ and $2^{61} - 1$ are measured by spectral tests, and the best ones are presented. The generators with these multipliers are portable and very fast. They have also passed several empirical tests, including the frequency test, the run test, and the maximum-of-t test.

Categories and Subject Descriptors: G.3 [**Mathematics of Computing**]: Probability and Statistics—*random number generation*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Cycle length, efficiency, multiplicative congruential random-number generators, portability, spectral test

1. INTRODUCTION

The demand for random numbers in many scientific applications is increasing due to the use of high-performance computer systems for large-scale scientific problems. Even a low-end workstation can consume 10^7 random numbers in a few minutes. However, the most widely used multiplicative, congruential random-number generators with modulus $2^{31} - 1$ have a cycle length of about 2.1×10^9. Kirkpatrick and Stoll [1981] presented a *lagged-Fibonacci generator* (cf. Anderson [1990]), called $R250$, which is

Author's address: Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan, Republic of China; email: pcwu@csie.nctu.edu.tw.

very fast and has a cycle length of $2^{250} - 1$. Although $R250$ passed a number of statistical tests, a recent experience [Selke 1993] showed that $R250$ gives wrong results for a clustered Monte Carlo simulation, while the multiplicative congruential generator

$$x_i = 16807 \cdot x_{i-1} \bmod (2^{31} - 1),$$

the minimal standard generator suggested by Park and Miller [1988], worked very well.

Multiplicative congruential generators have been widely used and tested. It would be better if this kind of generator provides a longer cycle length to meet the current needs of large-scale simulations. Payne et al. [1969] predicted that, due to increases in computer speed and the next Mersenne prime of $2^{31} - 1$ being $2^{61} - 1$, multiplicative congruential generators with modulus $2^{61} - 1$ would be needed. The time has now arrived. The cycle length of these generators, up to $2^{61} - 2 \approx 2.3 \times 10^{18}$, is long enough for most current scientific applications. In the future, generators with modulus $2^{127} - 1$ may be required to provide a longer cycle length of up to 1.7×10^{38}. However, developing portable and efficient multiplicative congruential generators is not straightforward [Park and Miller 1988; Schrage 1979]. Moreover, developing generators with larger moduli such as $2^{61} - 1$ is more difficult than developing those with modulus $2^{31} - 1$.

This article presents the development of multiplicative congruential generators with prime modulus $m = 2^p - 1$ and four forms of multipliers: $2^{k1} - 2^{k2}$, $2^{k1} + 2^{k2}$, $m - 2^{k1} + 2^{k2}$, and $m - 2^{k1} - 2^{k2}$, $k1 > k2$. The multipliers for modulus $2^{31} - 1$ and $2^{61} - 1$ are measured by spectral tests, and the best ones are presented. The generators with these multipliers are portable and very fast. The performance results compared with other implementation techniques are also presented. These generators have also passed several empirical tests, including the frequency test, the run test, and the maximum-of-t test [Knuth 1981].

2. RELATED WORK

A linear congruential generator can be defined by the following equation:

$$x_i = a \cdot x_{i-1} + c \bmod m. \tag{1}$$

Computing the above equation requires one integer addition, one multiplication, and one division. Usually $m = 2^e$ is chosen to avoid the division operation and the portability problem in getting product $a \cdot x_{i-1}$ in high-level languages. Knuth [1981, pp. 22–24] analyzed multiplier $a = 2^k + 1$ for generators with modulus 2^e, $k < e$, and $c = 1$. The equation is shown below:

$$x_i = ((2^k+1) \cdot x_{i-1}) + 1 \bmod 2^e \tag{2}$$

The above equation can be computed by merely shifting and adding, and it avoids the multiplication and division operation. However, Knuth [1981, p. 22] showed that this kind of multiplier should be avoided based on the concept of potency. Knuth [1981, p. 24] suggested that (for multiplier $a = 2^k + 1$) multiplicative congruential generators with a prime modulus should be used instead.

A *multiplicative congruential generator* is a special case of Eq. (1) taking $c = 0$:

$$x_i = a \cdot x_{i-1} \bmod m. \tag{3}$$

To get a period of maximal length $m - 1$, m must be a prime; a is a primitive root of m; and $x_0 \in [1, m - 1]$ (cf. Knuth [1981, p.19]). Developing portable and efficient multiplicative congruential generators is not straightforward, because simply taking $m = 2^e$ does not get an "almost correct" implementation. Park and Miller [1988] gave a sampling of simple but "bad" multiplicative congruential generators, many of which use modulus 2^e, e.g., $e = 16$, $e = 31$, and $e = 32$.

Some research has been devoted to portable implementations of multiplicative congruential generators with $m = 2^{31} - 1$. Schrage [1979] presented a Fortran program that uses two integer multiplication operations and several bit operations to get the product $16807 \cdot x$. This technique is more costly to apply for generators with a larger modulus. Park and Miller [1988] presented a portable program based on simple integer arithmetic. The technique also works for generators with modulus $2^{61} - 1$, if the compiler used supports 64-bit integer arithmetic. All these portable implementations allow only small multipliers and are slower than those using assembly codes directly.

3. THE $\pm 2^{k1} \pm 2^{k2}$ MULTIPLIERS

The division operation (mod m) in multiplicative congruential generators with (prime) modulus $m = 2^p - 1$ can be performed by shifting and addition [Payne et al. 1969]. To further replace multiplication with shifting and addition, multiplier a must be in the form of simple expressions of 2^k. To guarantee that the resulting generators are good, we need to meet the following criteria: A multiplier a is adequate if

(1) The number a is a primitive root of m (or *primitive element modulo* m): a is a primitive root of m if $a^n \bmod m \neq 1$ for $n = 1, 2, \ldots, m - 2$ [Knuth 1981, p. 10]. Knuth [1981, p. 20] presented a method to test whether a number is a primitive root of m:

The number a is a primitive root of m if and only if

$a \neq 0 (\bmod m)$, and $a^{(m-1/q)} \neq 1 (\bmod m)$,

for any prime divisor q of $m - 1$.

(2) The generator with multiplier a and modulus m should pass a number of statistical tests. Knuth [1981, p. 89] and Fishman and Moore [1986] all suggested the spectral test.

The multipliers with the simplest form are $\pm 2^k \bmod m$, i.e., multipliers 2^k and $m - 2^k$, $0 \le k \le \lfloor \lg m \rfloor$, where $\lg m$ denotes $\log_2 m$. Arithmetic modulo $2^p - 1$, for any $p > 1$, is well known as "one's complement" arithmetic. For this arithmetic, multiplication by 2^k results in a k-place left cyclic shift of the binary digits. That is, if integer x has binary digits

$$b_{p-1} \ldots b_0$$

then the integer $2^k \cdot x \bmod (2^p - 1)$ has digits

$$b_{p-k-1} \ldots b_0 \, b_{p-1} \ldots b_{p-k}.$$

Also, $-x \bmod (2^p - 1) = 2^p - 1 - x$ results in bitwise negation. Thus, with $m = 2^p - 1$, a multiplier of form $a = 2^k$ must cause the sequence to cycle after p steps, as p cyclic k-place shift restores the original digits. If $a = -2^k$, each step is a k-place cyclic shift followed by negation, and the sequence will cycle after p steps if p is even, or $2p$ steps otherwise. Hence, no such multiplier can be a primitive root.

Now, we pay attention to multipliers with the form of $\pm 2^{k1} \pm 2^{k2} \bmod m$, $k1 > k2$, i.e., four kinds of multipliers: $2^{k1} - 2^{k2}$, $2^{k1} + 2^{k2}$, $m - 2^{k1} + 2^{k2}$, and $m - 2^{k1} - 2^{k2}$. There is an efficient algorithm for generators with these multipliers.

Let $a = \pm 2^{k1} \pm 2^{k2}$, $m = 2^p - 1$.

$$\begin{aligned} x_i &= a \cdot x_{i-1} \bmod m. \\ &= x_{i-1}(\pm 2^{k1} \pm 2^{k2}) \bmod m. \\ &= (\pm 2^{k1} \cdot x_{i-1} \pm 2^{k2} \cdot x_{i-1}) \bmod m. \\ &= (\pm (2^{k1} \cdot x_{i-1} \bmod m) \pm (2^{k2} \cdot x_{i-1} \bmod m)) \bmod m. \end{aligned}$$

Multiplication by 2^k is a k-place left cyclic shift and can be rewritten as follows:

Let x be a p-bit number, $k < p$.
$2^k \cdot x \bmod (2^p - 1) = x$ (high-order k bits) + x (low-order $p-k$ bits) $\cdot 2^k$.

Thus, the multiplication and division can be reduced to shifting and addition (Figure 1).

Let $w_1 = 2^{k1} \cdot x_{i-1} \bmod m$, $w_2 = 2^{k2} \cdot x_{i-1} \bmod m$. Table I lists the equations for all these multipliers. Because $0 < w_1, w_2 < m$, the range of x' is $-m < x' < m$. When $x' > 0$, $x_i = x'$; otherwise, $x_i = m + x'$. Algorithm RAND shows the code.

Algorithm RAND

Input: multiplier a (p bits) and seed x_{i-1} (p bits), a is one of the forms

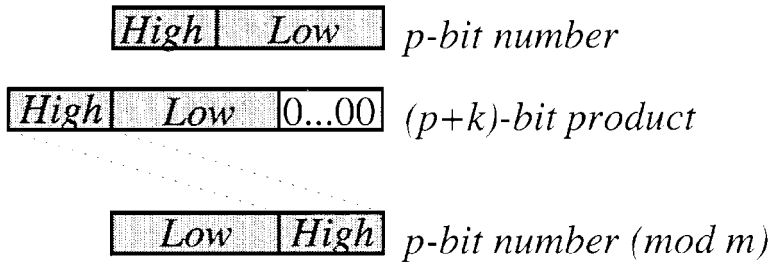

Fig. 1. Exchanging the low-order and high-order bits of x.

Table I. Equations of Four Forms of Multipliers and Their Equations

Form	Equation
$2^{k1} - 2^{k2}$	$x' = w_1 - w_2$
$2^{k1} + 2^{k2}$	$x' = w_1 + w_2 - m$
$m - 2^{k1} + 2^{k2}$	$x' = -w_1 + w_2$
$m - 2^{k1} - 2^{k2}$	$x' = m - w_1 - w_2$

of Table I.

Output: a random number $x_i = a \cdot x_{i-1} \bmod 2^p - 1$.
Method:

Compute x' using the corresponding equation in Table I.
IF $(x' < 0)$
THEN Output $x' + m$;
ELSE Output x';
End of algorithm.

4. SPECTRAL TEST AND EMPIRICAL TEST RESULTS

This section first presents spectral test results for generators with multipliers of $\pm 2^{k1} \pm 2^{k2}$ and modulus $2^{31} - 1$ and $2^{61} - 1$. The notation used here follows that of Anderson [1990]. The relative quality measure is

$$q_k = \nu_k/(\gamma_k \cdot m^{1/k}),$$

where ν_k is the spectral test result on k-dimensional space, and m is the prime modulus. The constants γ_k, $2 \le k \le 8$ were shown in Knuth [1981, p. 105] and Anderson [1990, Table 4.1]. The measure q_k is a real number in [0, 1]. Large values of q_k are best. We applied spectral test to all multipliers $\pm 2^{k1} \pm 2^{k2}$ for $m = 2^{31} - 1$ and $m = 2^{61} - 1$. These multipliers are rated by the minimum of q_k, $2 \le k \le 8$. Based on this measurement, the best multipliers for $m = 2^{31} - 1$ are $m - 2^{16} - 2^{11}$ and $2^{15} - 2^{10}$ (see Table II), and the best multipliers for $2^{61} - 1$ are $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ (see Table III). The measure $\beta_k = \lfloor \lg v_k \rfloor$ is interpreted

Table II. Two Generators with Multipliers $m - 2^{16} - 2^{11}$ and $2^{15} - 2^{10}$ for Modulus $m = 2^{31} - 1$

$a = 2147416063 = m - 2^{16} - 2^{11}$. min $q_k = 0.6211$				$a = 31744 = 2^{15} - 2^{10}$. min $q_k = 0.5703$			
k	q_k	β_k	v_k	k	q_k	β_k	v_k
2	0.6394	14	31840	2	0.6375	14	31744
3	0.8307	10	1203	3	0.7720	10	1118
4	0.7734	7	198	4	0.5703	7	146
5	0.6740	5	61	5	0.6740	5	61
6	0.6904	5	32	6	0.7335	5	34
7	0.6211	4	18	7	0.7246	4	21
8	0.6265	3	13	8	0.5783	3	12

Table III. Two Generators with Multipliers $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ for Modulus $2^{61} - 1$

$a = 4395899027456 = 2^{42} - 2^{31}$. min $q_k = 0.3780$				$a = 1073217536 = 2^{30} - 2^{19}$. min $q_k = 0.3653$.			
k	q_k	β_k	v_k	k	q_k	β_k	v_k
2	0.6580	30	1073741824	2	0.6577	29	1073217536
3	0.4442	19	658761	3	0.3653	19	541656
4	0.3780	14	17519	4	0.7533	15	34910
5	0.3789	11	2195	5	0.5003	11	2898
6	0.6128	9	909	6	0.5737	9	851
7	0.4440	7	251	7	0.6315	8	357
8	0.6161	7	172	8	0.5803	7	162

Table IV. Two of the Best Multipliers from Fishman and Moore [1986] for Modulus $2^{31} - 1$

$a = 1754050460$. min $q_k = 0.7229$				$a = 742938285$. min $q_k = 0.6211$			
k	q_k	β_k	v_k	k	q_k	β_k	v_k
2	0.9257	15	46095	2	0.8672	15	43186
3	0.8266	10	1197	3	0.8604	10	1246
4	0.8125	7	208	4	0.8594	7	220
5	0.8176	6	74	5	0.8286	6	75
6	0.8414	5	39	6	0.8198	5	38
7	0.7591	4	22	7	0.6211	4	18
8	0.7229	3	15	8	0.6747	3	14

as the number of bits that are "random" when k-tuples are considered [Anderson 1990].

How good are these generators? Fishman and Moore [1986] presented an exhaustive spectral test of multipliers for modulus $2^{31} - 1$ (see Table IV). The 410 multipliers selected as best (as corrected by Park and Miller [1988]) all meet the quality measure $q_k \geq 0.8$, for $2 \leq k \leq 6$. By extending k to 8 in spectral tests on all these multipliers, we find that the largest min

Table V. Summary of Spectral Tests on Various Multipliers for Modulus $2^{31} - 1$

Multiplier	min q_k, $2 \le k \le 8$
1754050460	0.7229
742938285	0.6211
$m - 2^{16} - 2^{11}$	0.6211
$2^{15} - 2^{10}$	0.5703
397204094	0.5520
630360016	0.4316
16807	0.3375

$q_k = 0.7229$ when $a = 1754050460$. The multiplier 742938285, which was considered the best, has min $q_k = 0.6211$. On the other hand, in Table II, the multiplier $m - 2^{16} - 2^{11}$ also has min $qk = 0.6211$. Table V summarizes the quality measure of these multipliers. Based on the spectral test results, multipliers $m - 2^{16} - 2^{11}$ and $2^{15} - 2^{10}$ are better than the three widely used multipliers 16807, 397204094, and 630360016.

Because there are no well-known multipliers for modulus $m = 2^{61} - 1$, we compare the multipliers $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ with two multipliers found by the following methods:

(1) Search for the best multiplier $a = 37^b \bmod m$, where b is relatively prime to $m - 1$. Because 37 is the minimal primitive root of m, all these multipliers are primitive roots of m [Fishman and Moore 1986]. This search is done in the range $b \epsilon [1,10^6]$.

(2) Search for the best multiplier $a = 2^k \pm 1$. The number a must also be a primitive root.

Method (1) performs a time-consuming search to find a rather good multiplier. Method (2) searches for multipliers of a more restricted form, which is easier to compute than those of the general form $\pm 2^{k1} \pm 2^{k2} \bmod m$. The results of these searches are shown in Table VI. The best multiplier in method (1), $a = 37^{458191} \bmod m$, has min $q_k = 0.7129$, and the best multiplier in method (2), $a = 2^{38} - 1$, has min $q_k = 0.0073$. This result shows that $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ are far better than $2^{38} - 1$, and using the form $a = 2^k \pm 1$ is not likely to yield a good multiplier.

Table VII summarizes spectral test results on multipliers for $2^{61} - 1$. Although the min q_k values of $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ are not very high, they are still higher than that of 16807, a commonly used multiplier. Moreover, because q_k is not an absolute measure, multipliers with different m cannot be compared using q_k. A large modulus m is better based on the measure β_k. Generators with multipliers $2^{42} - 2^{31}$ and $2^{30} - 2^{19}$ and modulus $2^{61} - 1$ are better than any generators with modulus $2^{31} - 1$.

We apply three empirical tests on the 61-bit generators with multipliers $2^{30} - 2^{19}$ and $2^{42} - 2^{31}$ and the 31-bit generators with $m - 2^{16} - 2^{11}$,

Table VI. Two Sample Generators with Modulus $m = 2^{61} - 1$

$a = 37^{458191} \bmod m$ = 2137866620694229420. min q_k = 0.7129.				$a = 2^{38} - 1$ = 274877906943. min q_k = 0.0073.			
k	q_k	β_k	v_k	k	q_k	β_k	v_k
2	0.9122	30	1488478930	2	0.0073	23	11863282
3	0.8258	20	1224562	3	0.0221	14	32767
4	0.7745	15	35889	4	0.0247	10	1144
5	0.7316	12	4238	5	0.1387	9	809
6	0.7322	10	1086	6	0.0856	6	127
7	0.7129	8	403	7	0.2247	6	127
8	0.7451	7	208	8	0.4549	6	127

Table VII. Summary of Spectral Tests on Multipliers for Modulus $m = 2^{61} - 1$

Multiplier	min q_k, $2 \le k \le 8$
$37^{458191} \bmod m$	0.7129
$2^{42} - 2^{31}$	0.3780
$2^{30} - 2^{19}$	0.3653
$2^{38} - 1$	0.0073

$2^{15} - 2^{10}$. The frequency test takes x mod 12. The run test examines the length of "run up" sequences, categorized into [1, 6] and > 6. The maximum-of-t test takes the maximum of 5 consecutive numbers and examines whether $\max(x) < 7/8m$. The degrees of freedom of these tests are 11, 6, and 1, respectively. To analyze the test results, we use the chi-square test [Knuth 1981, p. 44] in 6 rounds. Each test consumes 2 million random numbers. In total, our experiment consumes 36 million consecutive random numbers generated by seed 1. The chi-square result V is rejected if V is outside [1%, 99%]. The result is "suspect" if V is in [1%, 5%] or [95%, 99%]. The result is "almost suspect" if V is in [5%, 10%] or [90%, 95%]. The following are the results for the 61-bit generators with $2^{30} - 2^{19}$ and $2^{42} - 2^{31}$ and the 31-bit generators with $m - 2^{16} - 2^{11}$, $2^{42} - 2^{31}$, 16807, and 1754050460. The latter two are included merely for comparison. All these generators are satisfactory.

5. PORTABILITY AND PERFORMANCE RESULTS

Figure 2 shows the C program for the generator $x_i = (2^{30} - 2^{19})\, x_{i-1} \bmod (2^{61} - 1)$. This program can easily be adapted for any generator of the form $\pm 2^{k1} \pm 2^{k2} \bmod m$. `LOG_W` denotes the number of bits in integer types, and `LOG_M` denotes $\lceil \lg m \rceil$, i.e., `LOG_M` = p, letting $m = 2^p - 1$. `K1` and `K2` represent $k1$ and $k2$. The equation from Table I should be plugged into the code. The "`unsigned`" variable `x0` makes the right shift operator ($\gg$) fill zeros on the left. The program assumes that the size of "long long int" type in C is 64 bits wide. For C compilers not supporting such an integer type,

Table VIII. Empirical Tests on Two 61-Bit Generators and Four 31-Bit Generators

	Frequency Test	Run Test	Maximum-of-t
$a = 1754050460$ $m = 2^{31} - 1$	12.8195	5.5465	0.9152
	8.7124	9.5065	0.0992
	17.3912 AS	7.2807	1.1192
	12.2226	3.5790	3.0317 AS
	7.8808	3.5037	0.2941
	11.5636	9.6334	0.0298
$a = 16807$ $m = 2^{31} - 1$	17.9788 AS	5.9388	0.0296
	7.2673	5.4556	0.7934
	11.8550	4.6125	1.6692
	6.4879	4.0551	0.0821
	25.8024 R	2.7103	2.5059
	9.9464	4.5692	4.8415 S
$a = 2^{30} - 2^{19}$ $m = 2^{61} - 1$	5.3201 AS	10.0971	0.0430
	4.0330 S	3.2456	0.0115 AS
	13.2601	9.8225	4.0567 S
	16.5406	8.4553	0.4291
	7.4707	2.1997 AS	1.1740
	6.7406	4.3256	0.0263
$a = 2^{42} - 2^{31}$ $m = 2^{61} - 1$	9.0355	4.3436	0.2831
	4.7160 AS	4.0678	0.6844
	16.3919	14.2237 S	1.5277
	6.0833	6.8317	1.5925
	15.4968	12.5323 AS	0.0949
	7.5346	4.6535	0.0273
$a = 2^{15} - 2^{10}$ $m = 2^{31} - 1$	8.0367	7.4715	1.0156
	11.6435	1.9442 AS	2.5679
	8.8121	4.0140	0.9714
	10.2040	8.7158	0.6899
	12.6949	2.4861	0.7757
	12.4820	5.7903	0.2673
$a = m - 2^{16} - 2^{11}$ $m = 2^{31} - 1$	22.7399 S	1.7901 AS	0.2356
	6.4100	2.9220	0.0246
	8.5287	10.5368	0.0021 S
	16.1273	8.3786	1.7448
	18.7574 AS	5.7459	0.0513
	8.5351	5.3021	0.0181

AS = Almost Suspect; S = Suspect; R = Reject

the programmer needs to handle double-word integers by hand. Because the program contains only shifting and addition operations, coding these operations by hand is not difficult.

The code in Figure 2 is general for any $k1$ and $k2$. For a specific generator, the code can be simplified. For example, the recurrence for $a = 2^{30} - 2^{19}$ can be rewritten in the C fragment:

```
#define M 0x1fffffffffffffll
long long int x;
x = (x ≫ 31) + ((x ≪ 30) & M)
  – (x ≫ 42) – ((x ≪ 19) & M);
if (x < 0)
  X + = M;
```

```
#define LOG_W 64
#define LOG_M 61
#define M    0x1fffffffffffffffll
#define K1   30
#define K2   19
static long long int seed = 1;
void rand_seed(long long int s)
{
  seed = s;
}
long long int rand()
{
  unsigned long long int x0 = seed;
  long long int x1, w1, w2;

  w1 = (x0 >> (LOG_M - K1)) /* low-order bits of w1 */
   + ( (x0<<(K1+LOG_W-LOG_M)) >> (LOG_W-LOG_M) ); /* high-order bits */

  w2 = (x0 >> (LOG_M - K2)) /* low-order bits of w2 */
   + ( (x0<<(K2+LOG_W-LOG_M)) >> (LOG_W-LOG_M) ); /* high-order bits */

  x1 = w1 - w2;  /* the equation from Table 1 */
  if (x1<0)
    x1 = x1 + M;
  seed = x1;
  return x1;
}
```

Fig. 2. The C program for the generator $x_i = (2^{30} - 2^{19})x_{i-1} \bmod (2^{61} - 1)$.

The bitwise-and operator (&) is used to get the high-order bits. This gets simpler code; however, compilers may generate the code that loads constant `M` from memory, which is usually less efficient than bit shift.

We compared the performance of our generators with other development techniques. A test was conducted that generated 2 million random integers for all following programs. Program *m61-port* is the generator by Park and Miller [1988] with modulus $2^{61} - 1$. The codes on Sun SPARC and HP PA-RISC were compiled by GNU CC (gcc -O); the codes in Alpha and RS/6000 were compiled by the vendor's C compiler (cc -O). Program *m61-asm* denotes generators with modulus $2^{61} - 1$ developed using assembly languages. We had two implementations: one on the Sun SPARC and one on the DEC Alpha. The SPARC assembly code allows only 32-bit multipliers, while the Alpha assembly code allows any 61-bit multipliers. Program *m61-p3019* denotes the 61-bit generator with $a = 2^{30} - 2^{19}$. All these 61-bit generators generated the same sequence of random numbers. Program *m31-asm* is an assembly version of a 31-bit generator with any multiplier a and modulus $m = 2^{31} - 1$. The time was measured with $a = 16{,}807$. Program *m31-p1611* denotes a 31-bit generator with $a = m - 2^{16} - 2^{11}$. Program *m31-p1510* denotes a 31-bit generator with $a = 2^{15} - 2^{10}$. Finally, the test on the generator with modulus 2^{32} was also performed. The results show that our generators (*m61-p3019, m31-p1611,* and *m31-p1510*) are very fast.

Table IX. Performance Results on Various Machines

Program	SPARC-IPC	SPARC-2	PA-RISC	RS/6000	Alpha
m61-port	53.18s.	31.72s.	57.96s.	10.99s.	1.00s.
m61-asm	13.63s.	8.00s.	n.a.	n.a.	0.92s.
m61-p3019	4.57s.	2.63s.	2.88s.	0.95s.	0.38s.
m31-asm	6.93s.	4.25s.	2.13s.	0.69s.	0.70s.
m31-p1611	2.53s.	1.58s.	1.22s.	0.65s.	0.50s.
m31-p1510	2.28s.	1.43s.	1.23s.	0.66s.	0.42s.
$a \cdot x \bmod 2^{32}$	5.02s.	2.93s.	0.79s.	0.36s.	0.48s.

n.a = data not available; the computing time for 2 million random integers is given in seconds

6. CONCLUSION AND FUTURE WORK

This article has presented the development of multiplicative congruential generators with prime modulus $2^p - 1$ and four forms of multipliers: $2^{k1} - 2^{k2}$, $2^{k1} + 2^{k2}$, $m - 2^{k1} + 2^{k2}$, and $m - 2^{k1} - 2^{k2}$, $k1 > k2$. The multipliers for modulus $2^{31} - 1$ and $2^{61} - 1$ have been measured by spectral tests. The generators with these multipliers are portable and very fast. They have also passed several empirical tests, including the frequency test, the run test, and the maximum-of-t test. In the future, more tests and more experience with these generators are needed.

ACKNOWLEDGMENTS

The author would like to thank the referees and the editor, whose comments helped to improve the overall presentation.

REFERENCES

ANDERSON, S. L. 1990. Random number generations generators on Vector supercomputers and other advanced architectures. *SIAM Rev. 32*, *2* (June), 221–251.

FISHMAN, G. A. AND MOORE, L. R. 1986. An exhaustive analysis of multiplicative congruential random number generators with modulus 2^{31}-1. *SIAM J. Sci. Stat. Comput. 7*, *1*, 24–45.

KIRKPATRICK, S. AND STOLL, E. 1981. A very fast shift-register sequence random number generator. *J. Comput. Phys. 40*, 517–526.

KNUTH, D. E. 1981. *The Art of Computer Programming*. Vol. 2, *Seminumerical Algorithms*. Addison-Wesley, Reading, Mass.

PARK, S. K. AND MILLER, K. W. 1988. Random number generators: Good ones are hard to find. *Commun. ACM 31*, *10* (Oct.), 1192–1201.

PAYNE, W. H., RABUNG, J. R., AND BOGYO, T. P. 1969. Coding the Lehmer pseudo-random number generator. *Commun. ACM 12*, *2*, 85–86.

SCHRAGE, L. 1979. A more portable Fortran random number generator. *ACM Trans. Math. Softw. 5*, *2*, 132–138.

SELKE, W. 1993. Cluster-flipping Monte Carlo algorithm and correlations in "good" random number generators. *JETP Lett. 58*, *8*, 665–668.

Received March 1995; revised September 1996; accepted November 1996