A FAST, HIGH QUALITY, AND REPRODUCIBLE PARALLEL LAGGED-FIBONACCI PSEUDORANDOM NUMBER GENERATOR

MICHAEL MASCAGNI
STEVEN A. CUCCARO
DANIEL V. PRYOR
M. L. ROBINSON

Supercomputing Research Center, I.D.A.
17100 Science Drive
Bowie, Maryland 20715-4300 **USA**

SRC-TR-94-115

March 15, 1994

ABSTRACT. We study the suitability of the additive lagged-Fibonacci pseudorandom number generator for parallel computation. This generator has relatively short period with respect to the size of its seed. However, the short period is more than made up for with the huge number of full-period cycles it contains. These different full-period cycles are called equivalence classes. We show how to enumerate the equivalence classes and how to compute seeds to select a given equivalence class. In addition, we present some theoretical measures of quality for this generator when used in parallel. Next, we conjecture on the size of these measures of quality for this generator. Extensive empirical evidence supports this conjecture. In addition, a probabilistic interpretation of these measures leads to another conjecture similarly supported by empirical evidence. Finally we give an explicit parallelization suitable for a fully reproducible asynchronous MIMD implementation.

1991 *Mathematics Subject Classification.* 65C10, 65Y05.

Key words and phrases. random number generation, parallel computation, Fibonacci generator, MIMD, SIMD, reproducible.

1. Introduction.

In Knuth's well known exposition on pseudorandom number generation [8], several methods of generation are considered. Among these is the additive lagged-Fibonacci pseudorandom number generator:

$$x_n = x_{n-k} + x_{n-l} \pmod{M}, \quad l > k. \tag{1}$$

This generator is defined by the modulus, M, the register length, l, and the lag, k. When M is prime, periods as large as $M^l - 1$ are possible. However, it is more common to consider lagged-Fibonacci generators with $M = 2^m$, for some m. These generators with power-of-two moduli are considerably easier to implement than general prime moduli; however, their periods are much smaller than in the prime-modulus case.

In Marsaglia's empirical study of pseudorandom number generators [12], the additive lagged-Fibonacci generator with power-of-two modulus was one among many considered. Overall, this generator did well on all of Marsaglia's "stringent" tests, save the "non-overlapping birthday spacing test." However, Marsaglia noted that by choosing a generator with a large register length, l, improvements are seen in the "non-overlapping birthday spacing test."

There are several other compelling reasons to study this generator, [14]. This generator is used by Thinking Machines Corporation in their "Connection Machine Scientific Subroutine Library" (CMSSL) as a parallel pseudorandom number generator.[1] In addition, Brent has recently added to the understanding of this generator in both theory and practice, [1, 2]. Aside from clarifying the conditions for obtaining the maximum possible period, Brent carefully analyzed the use of the additive lagged-Fibonacci integer generator and its floating-point counterpart. In most Monte Carlo applications, uniformly distributed floating-point numbers, not integers, are desired. The floating-point counterpart of equation (1) is $\omega_n = \omega_{n-k} + \omega_{n-l} \pmod 1$. Here the $\omega_n \in [0, 1)$ are floating-point numbers. Besides being able to directly compute floating-point pseudorandom numbers, this formulation and the integer counterpart in equation (1) are amenable to efficient vectorization, [1]. Thus we see that this generator in both the integer and floating-point versions is very versatile indeed. There are several very good reasons for exploring the additive lagged-Fibonacci generator with power-of-two modulus to find an effective parallel implementation. These include the strong empirical evidence of pseudorandomness for this generator, its computational simplicity, and its highly efficient implementation.

First, we must understand some of the properties that are desirable in a parallel pseudorandom number generator. Besides efficiency and pseudorandomness, which are properties of the generator when used in serial, we require that:

(1) the generator must be easy to parallelize (this question is the primary concern of this paper).
(2) the generator must be reproducible in a "strong" sense.

Property 2 is very important to computational scientists. When doing Monte Carlo calculations on new machines, exact agreement with previous and trusted calculations is

[1] It is the careful study of their (lack of a) seeding algorithm that was the prime motivation for this work.

essential. This is not an easy task, as many sophisticated Monte Carlo calculations can be quite complicated. By reproducibility in the "strong" sense we require reproducibility both on the same machine with a different partitioning of the processing resources and between machines. This demanding definition of reproducibility ensures the portability of a parallel generator to any parallel machine–a rather lofty goal, but one that we show is accessible to the additive lagged-Fibonacci generator.

The plan of the paper is as follows. In §2 we review the conditions equation (1) and the seed for obtaining the maximal-period for these generators. We next introduce an equivalence relationship on the set of seeds that are in a maximal-period cycle. The large number of equivalence classes (ECs) that result is the basis for our parallelization. We then describe an enumeration of all of the ECs for a generator. This leads to an algorithm for the computation of a seed in a given EC. We conclude §2 with a discussion on the quality of this generator in terms of exponential sums. Next comes a conjecture on the full-period exponential sums in question. No proof is given, but we present empirical evidence that supports the validity of the conjecture. In addition, we use a probabilistic interpretation of the full-period exponential sums to conjecture on the magnitude of their associated partial-period sums. This conjecture is also supported by empirical evidence. In §3 we use an enumeration of the ECs as the basis for parallelization and analyze the parent-child generators in terms of their exponential sum quality. Finally, in §4 we summarize the results and propose directions for further study.

2. Properties of the Generator.

2.1 Cycle Structure.

Let us begin with computing the period of these generators. If M is prime, then a period of $M^l - 1$ is possible provided that the characteristic polynomial $f(x) = x^l - x^k - 1$ is primitive modulo M. In the case of interest, when $M = 2^m$, the maximum possible period is $(2^l - 1)2^{m-1}$. In general, a linear recurrence modulo 2^m has period $(2^l - 1)2^{m-1}$ if and only if the following three conditions hold:

(1) modulo 2 the sequence has period $(2^l - 1)$,
(2) modulo 2^2 the sequence has period $(2^l - 1)2$,
(3) modulo 2^3 the sequence has period $(2^l - 1)2^2$.

For a proof see [13] or exercise 11 of §3.2.2 in [8]. We are only interested in simple additive lagged-Fibonacci generators, so our characteristic polynomial, $f(x) = x^l - x^k - 1$, is a trinomial. In this case Brent recently proved, [2], that when $f(x)$ is a trinomial, primitivity of $f(x)$ modulo 2 and $l > 2$ suffice to give the maximum possible period of $(2^l - 1)2^{m-1}$.

There is a conceptual benefit from working modulo a power-of-two: a qualitative description of the cycle structure is possible. Let us assume that we have a lagged-Fibonacci generator with the maximum possible period of $(2^l - 1)2^{m-1}$, i.e. equation (1) satisfies Brent's conditions. Taken modulo 2, equation (1) defines a shift-register sequence. With $f(x)$ primitive modulo 2, we obtain a maximal-period shift-register sequence of period $2^l - 1$, [6, 17]. It is well known that this sequence cycles over all possible nonzero contents of its l-bit state. Next consider equation (1) taken modulo 4. The least-significant bits of the register are just the maximal-period shift-register sequence from the modulo 2 case. The most-significant bits are the superposition of two sequences: (a), the maximal-period

shift-register sequence from the initial values of the most-significant bits, and (b), the impulse response of the carries from the least-significant bits.[2] Sequence (a) cycles with period $2^l - 1$ if it is nonzero and adds bit-wise to sequence (b). Thus, without loss of generality we can assume (a) is zero and concentrate on (b) to analyze the cycle structure. Sequence (b) is the superposition of impulse responses forced by carries from the least-significant bit. This impulse response has period $2^l - 1$. Since we have the maximum-possible period, the carries must also have period $2^l - 1$. Thus a particular carry will force a period $2^l - 1$ impulse response, after which the periodic repeat of the first carry will zero the impulse response. Thus each carry will produce a period $(2^l - 1)2$ response made up of a $2^l - 1$ length maximal-period sequence followed by $2^l - 1$ zeros. Because we obtain the maximum possible period, the superposition of these sequences also has period $(2^l - 1)2$. This explains the doubling of the period when a new most-significant bit is added and gives an understanding of the cycle structure of the additive lagged-Fibonacci generator.

We now understand how each new most-significant bit doubles the period of this generator; however, this adds l bits of seed to the generator, not just one. The maximum possible period of these generators is extremely short, given the size of the seed. In the prime modulus case, the maximum possible period is equal to the number of nonzero fills in the register. With $M = 2^m$, the maximum possible period of $(2^l - 1)2^{m-1}$ is considerably smaller than the number of nonzero fills, $2^{lm} - 1$. Where has all this state gone?

The answer to this question comes by considering the condition on the seed for obtaining the maximum possible period. Since we see that the lesser significant bits perturb the more significant bits through period-doubling carries, starting the generator with an all zero least-significant bit must reduce the period. In fact, the only condition to obtain the maximum possible period is that the seed must not all be zeros in the least-significant bit. In terms of residues modulo 2^m, this means that the seed cannot all be even. It is easy to calculate that the number of seeds that give the maximum possible period is $(2^l - 1)2^{l(m-1)}$. Since each of these seeds is in a maximum possible period cycle, there must be

$$E = \frac{(2^l - 1)2^{l(m-1)}}{(2^l - 1)2^{m-1}} = 2^{(l-1)(m-1)} \tag{2}$$

cycles with maximum possible period. If we define an equivalence relationship among seeds as being in the same cycle, then we see that these generators have E distinct ECs.

2.2 Equivalence Class Canonical Form.

The use of these ECs will be the key to parallelizing this generator. Thus we must be able to enumerate the ECs and to calculate a seed from each of the ECs given this enumeration. To derive an explicit enumeration, we must decide on one seed from the full period to serve as the representative for the given EC. We call this representative seed the EC's canonical form. Since the least-significant bit of this generator is a maximal-period shift-register sequence, we can choose some given nonzero fill for the least-significant bits of the canonical form. To transform an arbitrary seed into a seed with the canonical form's

[2] Recall that the impulse response in a shift-register is the sequence obtained by starting from the "unit" fill $[1, 0, 0, \ldots, 0]$, e.g. see [6].

least-significant bits, one need only advance the seed at most $2^l - 2$ times to match the least-significant bits.[3]

Now that we have placed the least-significant bits of a seed in canonical form, we must decide what to do with the more-significant bits. Given that we want simultaneously to fix the least-significant bits and remain in the same EC, we must leap ahead in the generator's cycle some multiple of $2^l - 1$, the period of the least-significant bits.

At this point we must define some notation to simplify the subsequent discussion. Let us first recast equation (1) into a matrix recursion modulo 2^m. First we write $\mathbf{x}_n = [x_n, x_{n-1}, \ldots x_{n-l+1}]^T$ for the contents of the register at the nth step. We may then write equation (1) as $\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} \pmod{2^m}$ with the $l \times l$ matrix $\mathbf{A}$ defined by:

$$\mathbf{A} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ k \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \ell \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 & 1 \\ 1 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 1 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 1 & 0 \end{pmatrix} \tag{3}$$

Let us also define an alternative notation for $\mathbf{x}_n$ so that we can refer to a particular set of bits across its l elements. Thus let $\mathbf{x}_n = [b_{m-1}^{(n)}, b_{m-2}^{(n)}, \ldots, b_0^{(n)}]$ be a column vector with $b_0^{(n)}$ holding $\mathbf{x}_n$'s least-significant bits and $b_{m-1}^{(n)}$ its most-significant bits.[4]

With the above notation we can say that since $\mathbf{A}$ is a recursion matrix for a maximum possible period additive lagged-Fibonacci generator, $\mathbf{A}^{2^l-1} \equiv \mathbf{I} \pmod 2$. Let us call $\mathbf{J} = \mathbf{A}^{2^l-1}$, so we may write $\mathbf{J} \equiv \mathbf{I} \pmod 2$ and $\mathbf{J}^2 \equiv \mathbf{I} \pmod 4$. Thus applying $\mathbf{J}$ to $\mathbf{x}_n$ leaves the least-significant bits fixed, while $\mathbf{J}^2$ leaves the two least-significant bits fixed. If we assume that we have changed a given seed's least-significant bits into the canonical form, application of $\mathbf{J}$ yields two possible b_1's, call them b_1 and $b_1^{\mathbf{J}}$. We choose one of these to be the b_1 for our EC representative. An unambiguous choice is to choose the smallest of b_1 and $b_1^{\mathbf{J}}$ viewed as l-bit integers. Note that b_1 and $b_1^{\mathbf{J}}$ can never be equal as that would contradict achieving the maximum possible period. Next we use $\mathbf{J}^2$ in a similar manner to choose between b_2 and $b_2^{\mathbf{J}^2}$. This procedure continues until we have our EC representative. By construction this algorithm produces the same seed for an EC when given any seed in the same EC. Additionally, application of this procedure to seeds from different ECs will produce different canonical form seeds.

[3] A table of size j of least-significant bits spaced equally around the cycle will reduce this to no more than $\lceil \frac{2^l-2}{j} \rceil$ steps.

[4] We will usually not use the superscript in subsequent discussion as we rarely need to refer to the bits from one time step to another.

Using this algorithm, we can produce a single seed that is the representative for its EC and is in a canonical form. How do we now enumerate the different ECs? The number of bits in the seed is $l \times m$, while the number of ECs is $E = 2^{(l-1)(m-1)}$. Thus a set of $(l-1)(m-1)$ bits specifies a unique EC. Our canonical form has already specified the l least-significant bits, so it could be hoped that the canonical form gives the following explicit enumeration:

(4)

m.s.b				l.s.b.	
b_{m-1}	b_{m-2}	...	b_1	b_0	
$\square$	$\square$	...	$\square$	$b_{0\,l-1}$	x_{l-1}
$\square$	$\square$	...	$\square$	$b_{0\,l-2}$	x_{l-2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\square$	$\square$	...	$\square$	$b_{0\,1}$	x_1
0	0	...	0	$b_{0\,0}$	x_0

This enumeration leaves exactly $(l-1)(m-1)$ bits to be specified in the canonical form and yields exactly $E = 2^{(l-1)(m-1)}$ different possibilities.

The remarkable thing is that this is the case! To prove it we first will have to understand the relationship between the bit vectors b_1 and $b_1^{\mathbf{J}}$ from our explicit construction of the EC representative. Recall that the bit vector b_1 is modified into $b_1^{\mathbf{J}}$ by the application of $\mathbf{J}$. The mechanism for this modification is the superposition of the carries from b_0 and the evolution of b_1 viewed as a shift-register. However, since $\mathbf{J} = \mathbf{A}^{2^l-1}$ and the period of the shift-register is $2^l - 1$, $b_1^{\mathbf{J}}$ is only a function of b_0 though the carries. Thus it follows that $b_1^{\mathbf{J}} = b_1 \oplus C_1(b_0)$, where $C_1(b_0)$ is a bit-vector valued function that when vectorially added modulo 2 to b_1 transforms it to $b_1^{\mathbf{J}}$ via the cumulative superposition of the carries. Such a transformation exists for any linear functional on bit-vectors. One consequence of this representation is that $C_1(b_0)$ must be the value of $b_1^{\mathbf{J}}$ when b_1 is all zeros. Since b_1 and $b_1^{\mathbf{J}}$ are not equal, it must be that $C_1(b_0)$ is nonzero and hence has a most-significant one bit, when viewing $C_1(b_0)$ as an l-bit integer. This most-significant one indicates that its bit position changes from b_1 to $b_1^{\mathbf{J}}$. Above we chose the smallest of b_1 and $b_1^{\mathbf{J}}$ viewed as l-bit integers in part of our canonical form. By choosing this position to be a zero, we ensure that choice among all b_1, $b_1^{\mathbf{J}}$ pairs and are free to fill in the remaining $l-1$ as we choose.

This procedure can be repeated for b_2 and $b_2^{\mathbf{J}^2}$ via the calculation of $C_2(\cdot)$, and we can continue to repeat this procedure for each successive b_i to produce the following EC tableau:

(5)

m.s.b				l.s.b.	
b_{m-1}	b_{m-2}	...	b_1	b_0	
$\square$	$\square$	...	0	$b_{0\,l-1}$	x_{l-1}
0	$\square$	...	$\square$	$b_{0\,l-2}$	x_{l-2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\square$	0	...	$\square$	$b_{0\,1}$	x_1
$\square$	$\square$	...	$\square$	$b_{0\,0}$	x_0

The location of the zeros in each x_n is the location of the most-significant one in the $C_i(\cdot)$ bit-vector. It turns out this bit-vector is only a function of the least-significant bit, b_0. We next give a simple proof of this fact.

For simplicity, let us consider $C_2(\cdot)$. Obviously, this bit-vector can be at most a function of b_0 and b_1, so we write $C_2(b_0, b_1)$. We have defined $C_2(b_0, b_1)$ as the most-significant bit of the image under $\mathbf{J}^2$ when $b_2 = \mathbf{0} = [0, 0, \ldots, 0]^T$. We may think of $C_2(b_0, b_1)$ as the most-significant bit of $\mathbf{J}^2[b_0, b_1, \mathbf{0}]$. By linearity we have $C_2(b_0, b_1) = C_2(b_0, \mathbf{0}) \oplus C_2(\mathbf{0}, b_1)$ where $C_2(b_0, \mathbf{0})$ is the most-significant bit of $\mathbf{J}^2[b_0, \mathbf{0}, \mathbf{0}]$ and $C_2(\mathbf{0}, b_1, \mathbf{0})$ is the most-significant bit of $\mathbf{J}^2[\mathbf{0}, b_1, \mathbf{0}]$. The most interesting of these is $C_2(\mathbf{0}, b_1)$. The period of the seed $[\mathbf{0}, b_1, \mathbf{0}]$ in a modulo 2^3 additive lagged-Fibonacci sequence will not be $(2^l - 1)4$. Instead it will be $(2^l - 1)2$. A seed with $b_0 = \mathbf{0}$ will always have $b_0 = \mathbf{0}$, so its top two bits can be thought of as a modulus 2^2 additive lagged-Fibonacci sequence. Recall that $\mathbf{J}^2 = \mathbf{A}^{(2^l-1)2}$. Thus $\mathbf{J}^2[\mathbf{0}, b_1, \mathbf{0}] = [\mathbf{0}, b_1, \mathbf{0}]$, and hence $C_2(\mathbf{0}, b_1) = \mathbf{0}$. This implies that $C_2(b_0, b_1) = C_2(b_0, \mathbf{0}) \oplus C_2(\mathbf{0}, b_1) = C_2(b_0, \mathbf{0}) \oplus \mathbf{0} = C_2(b_0, \mathbf{0})$. This proves that $C_2(\cdot)$ is a function only of the least-significant bit, b_0.

By similar arguments one can show that $C_i(\cdot)$ is a function of only the least-significant bit, b_0. Write

$$C_i(b_0, b_1, \ldots, b_{i-1}) = C_i(b_0, \mathbf{0}, \ldots, \mathbf{0}) \oplus C_i(\mathbf{0}, b_1, \ldots, b_{i-1}). \tag{6}$$

$C_i(\mathbf{0}, b_1, \ldots, b_{i-1})$ is the most-significant bit of $\mathbf{J}^{2^i}[\mathbf{0}, b_1, \ldots, b_{i-1}]$. $\mathbf{J}^{2^i} = \mathbf{A}^{(2^l-1)2^{i-1}}$, and the seed $[\mathbf{0}, b_1, \ldots, b_{i-1}]$ has period at most $(2^l - 1)2^{i-1}$. Thus $C_i(\mathbf{0}, b_1, \ldots, b_{i-1}) = \mathbf{0}$ and

$$C_i(b_0, b_1, \ldots, b_{i-1}) = C_i(b_0, \mathbf{0}, \ldots, \mathbf{0}) \oplus \mathbf{0} = C_i(b_0, \mathbf{0}, \ldots, \mathbf{0}). \tag{7}$$

The fact that $C_i(\cdot)$ is a function only of b_0 has a profound effect in computing a seed in a given EC. Once we have settled on the b_0 for our EC canonical form we can precompute all of the $C_i(b_0)$. This lets us precompute the location of the fixed zeros in the EC representative. Thus we have reduced the problem of producing a seed in a given equivalence class to some precomputation and the translation of a $(l-1)(m-1)$ bit equivalence class number into $(l-1)(m-1)$ open bit locations in seed tableau!

Another consequence of $C_i(\cdot)$'s dependence on b_0 alone is that we may try several b_0's to find one that gives an EC canonical form like equation (4). Proof that there always exists a b_0 such that equation (4) is possible is an open question. However, we have implemented a seeding scheme based on these explicit EC computations for the case of the additive lagged-Fibonacci used in Thinking Machine's CMSSL. Here the recurrence is:

$$x_n = x_{x-5} + x_{n-17} \pmod{2^{32}}. \tag{8}$$

With very little work a particular b_0 was found so that the tableau in (4) could be used. In fact, for all primitive trinomials of degree up to 255, a special b_0 was found that gave a canonical form as in equation (4).

2.3 Quality Issues.

An important issue in pseudorandom number generation is the quality of the numbers produced by a given recursion. There are many desirable randomness properties that a

sequence should possess, and it is important that it does well on empirical tests of statistical randomness. However, empirical testing has practical as well as theoretical limitations, [8]. Thus the inclusion of qualitative theoretical results that impact on pseudorandomness is always important.

A very powerful tool for the theoretical exploration of the quality of pseudorandomness is the exponential sum. Most importantly, the exponential sum is related to the discrepancy through upper and lower bounds, [11, 10, 15]. In turn, the discrepancy appears explicitly in the Koksma-Hlawka bound on integration error. This is very important since numerical integration is the fundamental Monte Carlo application.

The exponential sums for a modulo M sequence, $\{x_n\}$, are defined as:

$$C(i,j) = \sum_{n=0}^{i-1} e^{\frac{2\pi\sqrt{-1}}{M}(x_n - x_{n-j})}. \tag{9}$$

When $i = \mathrm{Per}(x_n)$, the period length, these are called full-period exponential sums, otherwise they are called partial-period exponential sums. The manipulation of these sums in order to calculate or bound them is fundamental to many areas of number theory, [9].

When x_n is defined by equation (1) we have $\mathrm{Per}(x_n) = (2^l - 1)2^{m-1}$. However, recurrences modulo a power-of-two often defy calculation of exponential sums based on them. This appears to be the case for additive lagged-Fibonacci sequences modulo a power-of-two. However, empirical observation indicates the maximal values of the full-period exponential sums given in equation (9) are well behaved. In fact, our empirical observations have led us to the conjecture that $|C(\mathrm{Per}(x_n), \cdot)| = O(\sqrt{\mathrm{Per}(x_n)})$. Another motivation for our full-period exponential sums comes through probabilistic reasoning. Each term of the sum in (9) is a complex number on the unit circle. In fact, it is one of 2^m possible values on the unit circle. Let us assume that each of $e^{\frac{2\pi\sqrt{-1}}{2^m}j}, j = 0, \ldots, 2^m - 1$ is equally probable, a modest desire for a pseudorandom sequence.[5] Viewing each term as a random variable, a certain scaling of equation (9) will have a limiting normal distribution. The complex number $\Re e^{\theta_n} + \sqrt{-1}\Im e^{\theta_n}$, with $\theta_n = 2\pi\left(\frac{x_n}{2^m}\right)$, can be viewed as the point in $\mathbb{R}^2$, $(\cos\theta_n, \sin\theta_n)$. Under the assumption of equiprobability, the two-dimensional distribution of these points has means $\mu_x = \mu_y = 0$ and covariance matrix

$$\mathbf{C} = \begin{pmatrix} \pi & 0 \\ 0 & \pi \end{pmatrix}. \tag{10}$$

Under the further assumption that the x_n's are independent, equation (9) is sum of i independent, identically distributed random variables. Thus by the multidimensional Lindeberg Central Limit Theorem, [5], the sum $\frac{C(i,j)}{\sqrt{i}}$, viewed as a point in $\mathbb{R}^2$, will have a limiting two-dimensional normal distribution with means $\mu_x = \mu_y = 0$ and covariance matrix equal to $\mathbf{C}$ in equation (10).

[5] It is known empirically that the residues are equiprobable in additive lagged-Fibonacci sequences suitable for use in a pseudorandom number generator, [12]. The empirical test for this is called the equidistribution test, [8].

There are several inferences that we can make from this limiting behavior. The first is that the real and imaginary parts of the sum in equation (9) are independent with distributions:

$$\lim_{i\to\infty} \mathbf{P}\left[\frac{\Re\{C(i,j)\}}{\sqrt{\pi i}} < \beta\right] = \lim_{i\to\infty} \mathbf{P}\left[\frac{\Im\{C(i,j)\}}{\sqrt{\pi i}} < \beta\right] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\beta} e^{-x^2/2}\, dx. \tag{11}$$

This further motivates our conjecture that the full-period exponential sums in equation (9) should be $O(\sqrt{\mathrm{Per}(x_n)})$, as the real and imaginary parts of equation (9) are normal with zero means and standard deviations of $O(\sqrt{\mathrm{Per}(x_n)})$. Another consequence of the asymptotic independent normality of the real and imaginary parts of equation (9) is that $\frac{|C(i,j)|^2}{\pi i}$ will have a limiting distribution that is $\chi^2(2)$, i.e., chi-squared with two degrees of freedom, [7]. In fact, full-period exponential sums formed with other putative pseudorandom sequences should have a similar limiting behavior. This forms the basis for another empirical test of pseudorandomness by checking for agreement between the distribution of $\frac{|C(i,j)|^2}{\pi i}$ and $\chi^2(2)$ among different full-period cycles.

We now present empirical evidence to support our conjecture on full-period exponential sums. We begin with qualitative examples and then provide quantitative evidence obtained through extensive computation. Figures 1, 2, 3, and 4 show the running sum in equation (9) as it marches over a full period. The figures are all for the same recursion, namely $x_n = x_{n-2} + x_{n-5} \pmod{2^m}$ where $m = 2$ in figure 1, $m = 3$ in figure 2, $m = 4$ in figure 3 and $m = 8$ in figure 4. The circles in the figures have radii that are integer multiples of $\sqrt{\mathrm{Per}(x_n)}$. We see that at all times these sums remain smaller than $2 \times \sqrt{\mathrm{Per}(x_n)}$. In Table 1 we have more numerical evidence to confirm our conjecture. Here we have computed the scaled full-period exponential sum, $|C(\mathrm{Per}(x_n), \cdot)|/\sqrt{\mathrm{Per}(x_n)}$, over different ECs. This rectangular table has rows that are indexed by m and columns that are indexed by l as defined in equation (1). Alongside the rows, we have written the (k, l) pair used to define the lagged-Fibonacci recurrence used for that value of l. In the table, bold-faced entries indicate that all ECs of that generator were searched, and the number shown is the maximum full-period exponential sum. Roman-faced entries are the maxima over 1000 ECs; italic entries are the maxima over 100 ECs. Table 2 is identical to Table 1 in format except the quantity tabulated is the maximal value of $|C(i,j)|/\sqrt{\mathrm{Per}(x_n)}$ encountered in the generator's full period.

Two remarkable facts evident from the tables are that the largest entry in each table is smaller than five and that corresponding values from Table 1 and Table 2 are very close. This supports our conjecture. In addition it motivates us to conjecture on the maximal values of the partial-period exponential sums. What we have computed for Table 2 are scaled maxima of partial-period exponential sums. However, we have started at a single initial seed and have not explored all initial seeds. This is not a substantial omission as the triangle inequality tells us that our tabulated partial-period exponential sums can be at most half of the largest partial-period exponential sum obtainable with any initial seed.

For other linear recurrences, when the full-period exponential sum is known, standard theory allows one to bound the partial-period sums based on the full-period values, [11, 15]. Thus if we conjecture that the full-period sum is $|C(\mathrm{Per}(x_n), \cdot)| = O(\sqrt{\mathrm{Per}(x_n)})$, then this

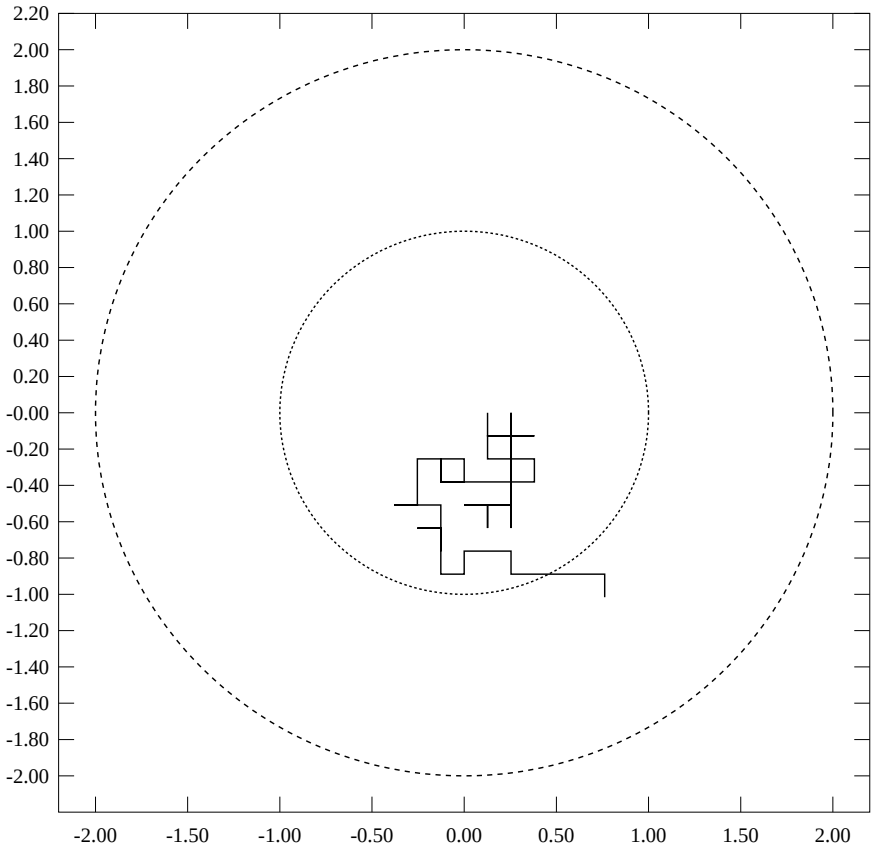

FIGURE 1. Exponential sums, $C(i,j)$, $j = 1$ to $\mathrm{Per}(x_n)$ for $x_n = x_{n-2} + x_{n-5} \pmod{2^2}$. Circles have radii that are integer multiples of $\sqrt{\mathrm{Per}(x_n)}$.

theory would state that the partial-period sums are $|C(i,j)| = O(\sqrt{\mathrm{Per}(x_n)}\log \mathrm{Per}(x_n))$. However, the tables indicate that this is an overestimation. Using probabilistic reasoning, the Central Limit Theorem behavior of the real and imaginary parts of equation (9), we may conclude that the real and imaginary parts of equation (9) behave like independent one-dimensional Brownian motions. Thus the law of the iterated logarithm for Brownian motion motivates a bound on the partial-period exponential sums, [4], i.e., $|C(i,j)| = O\left(\sqrt{\mathrm{Per}(x_n)\log\log \mathrm{Per}(x_n)}\right)$. This is slightly better than expected from upper estimates that follow from our original conjecture. In fact, it appears that the full-period and partial-period exponential sums are of the same magnitude.

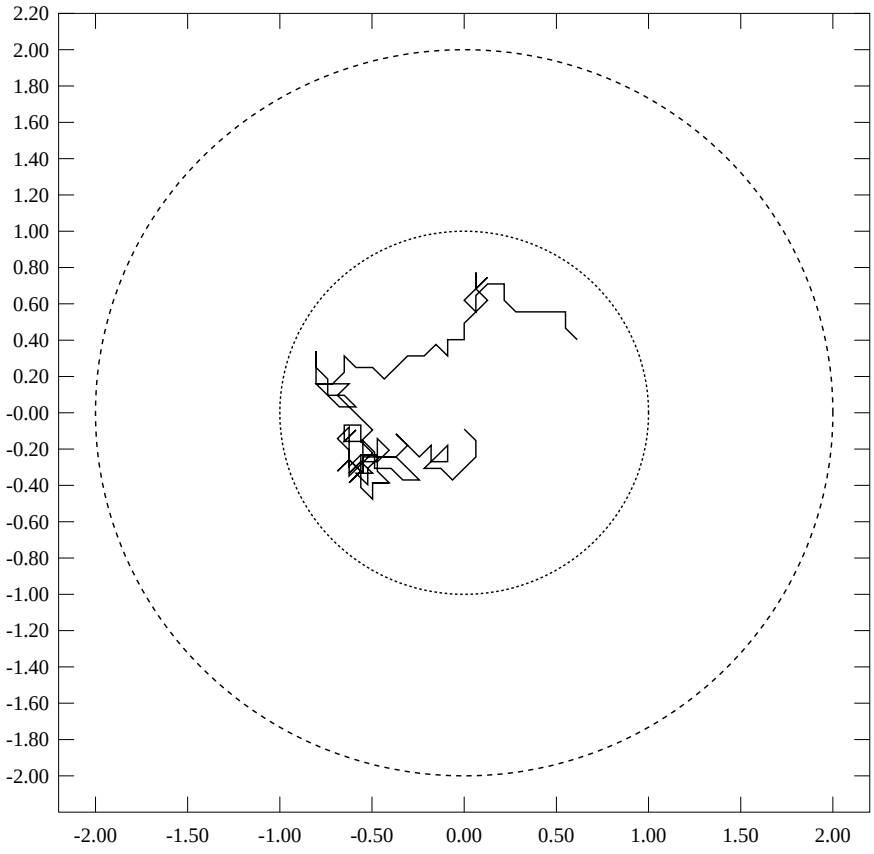

FIGURE 2. Exponential sums, $C(i,j)$, $j = 1$ to $\mathrm{Per}(x_n)$ for $x_n = x_{n-2} + x_{n-5} \pmod{2^3}$. Circles have radii that are integer multiples of $\sqrt{\mathrm{Per}(x_n)}$.

An important use of exponential sums in the case of parallel pseudorandom number generation is to use them as a measure of the exponential sum cross-correlation among different parallel pseudorandom number sequences. Suppose we have two modulo M pseudorandom sequences, $\{x_n\}$ and $\{y_n\}$. Their exponential sum cross-correlation is given by:

$$C(i,j) = \sum_{n=0}^{i-1} e^{\frac{2\pi\sqrt{-1}}{M}(x_n - y_{n-j})}. \tag{12}$$

In our case we are interested in sequences that are both generated by (1) and have seeds in different ECs. Since $C(i,j)$ is a sum over the difference of sequences at a fixed offset,

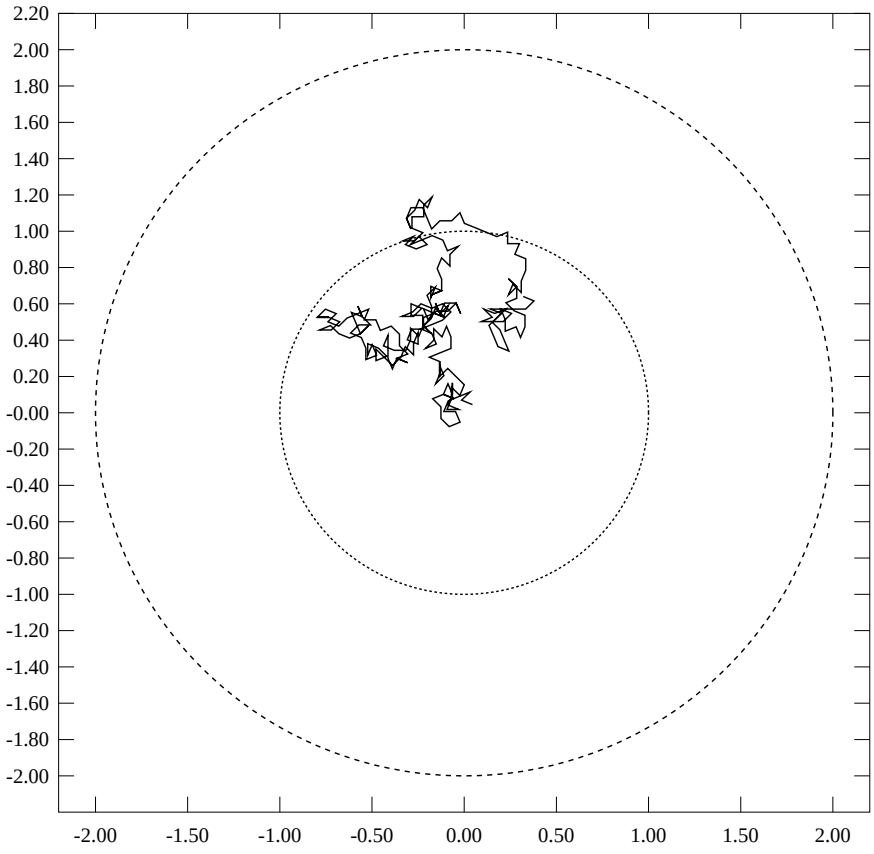

FIGURE 3. Exponential sums, $C(i,j)$, $j = 1$ to $\mathrm{Per}(x_n)$ for $x_n = x_{n-2} + x_{n-5} \pmod{2^4}$. Circles have radii that are integer multiples of $\sqrt{\mathrm{Per}(x_n)}$.

j, we can compute it by considering it as an exponential sum of the same recurrence in a potentially different EC. This is because the difference of two sequences obeying a given recursion will itself obey the recursion. Thus equation (12) gives us a qualitative tool to explore relationships between related ECs. This approach is discussed in §3.2.

3. Parallel Considerations.

3.1 Equivalence Classes for Parallelism.

With this huge number of ECs, parallel implementation is easy. The key is to associate each independent parallel process in the computation with a unique parallel process

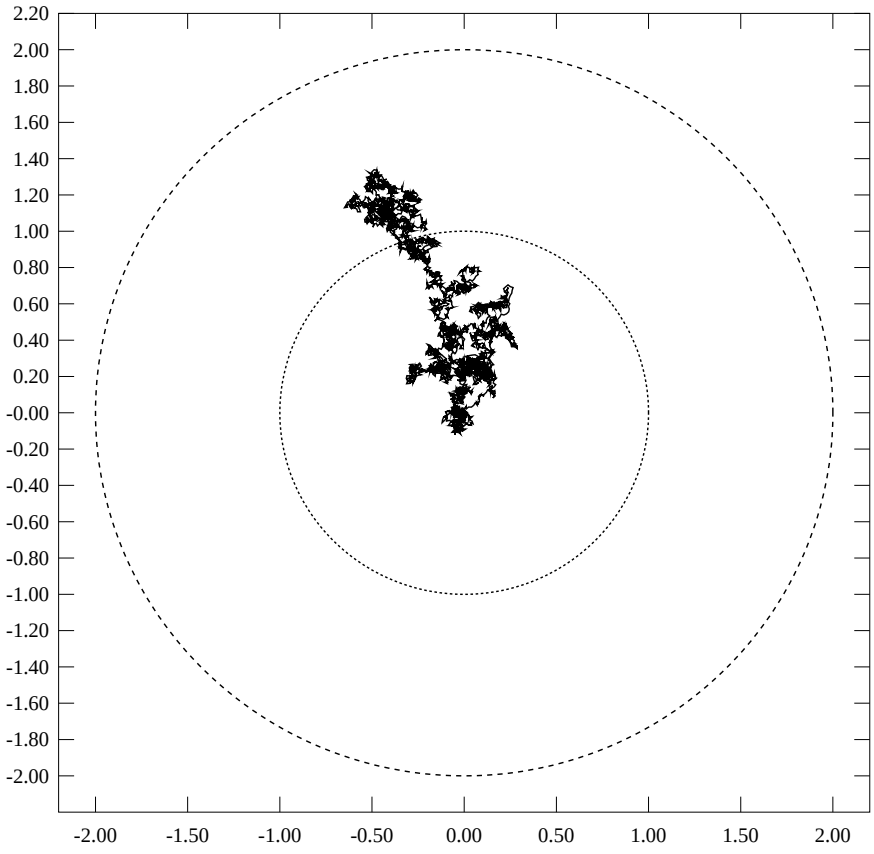

FIGURE 4. Exponential sums, $C(i,j)$, $j = 1$ to $\mathrm{Per}(x_n)$ for $x_n = x_{n-2} + x_{n-5} \pmod{2^8}$. Circles have radii that are integer multiples of $\sqrt{\mathrm{Per}(x_n)}$.

identifier, K. This K is then used to select the Kth EC for this process.[6]

This procedure works without difficulty provided the parameters for the generator are chosen so that no K is required in the computation that exceeds $2^{(l-1)(m-1)} - 1$.

One of the most demanding applications for pseudorandom generators is transport Monte Carlo, [16]. Here the path of a particle is followed and modified via sampling. Particles are emitted, absorbed, and created along a trajectory based on the parameters of the problem and the outcome of the pseudorandom number generator. The overall solution is then the average over many different particle trajectories. The partitioning of this

[6]In practical implementations, the Kth parallel process obtains the $f(K)$th EC, where $f(\cdot)$ is an appropriately chosen function.

	m								
(l,k)	1	2	3	4	5	6	7	8	9
(2,1)	**.577**	**.817**	**.817**	**.817**	**1.15**	**1.15**	**1.15**	1.15	1.15
(3,1)	**.378**	**1.20**	**1.45**	**1.90**	**2.08**	**2.66**	**2.36**	2.61	3.03
(3,2)	**.378**	**1.20**	**1.71**	**2.06**	**2.21**	**2.22**	**2.83**	2.65	2.84
(4,1)	**.258**	**1.32**	**1.84**	**2.70**	**3.21**	**3.25**	**4.05**	2.69	2.99
(4,3)	**.258**	**1.32**	**2.16**	**2.66**	**2.96**	**3.35**	**3.95**	3.46	3.60
(5,2)	**.180**	**1.63**	**2.46**	**2.92**	**3.49**	**4.01**	**4.57**	2.70	3.03
(5,3)	**.180**	**1.63**	**2.54**	**2.95**	**3.55**	**3.99**	**4.80**	3.12	3.62
(6,1)	**.126**	**1.14**	**2.76**	**4.02**	**3.82**	2.96	2.65	2.66	3.04
(6,5)	**.126**	**1.14**	**2.86**	**4.00**	**3.81**	2.99	3.90	3.03	3.05
(7,1)	**.089**	**1.51**	**2.89**	**3.43**	2.99	2.93	2.75	2.87	2.61
(7,3)	**.089**	**1.51**	**3.13**	**3.86**	2.54	2.36	2.69	2.70	2.86
(7,4)	**.089**	**1.51**	**2.71**	**3.66**	2.74	3.04	2.85	2.63	2.81
(7,6)	**.089**	**1.51**	**3.00**	**3.60**	2.90	2.69	3.06	2.73	2.55
(9,4)	**.044**	**1.46**	**3.27**	2.64	2.53	2.81	3.23	2.85	2.85
(9,5)	**.044**	**1.46**	**3.74**	2.76	3.02	2.63	2.74	2.83	2.68
(10,3)	**.031**	**1.03**	**3.99**	2.61	2.67	2.33	2.70	2.82	2.87
(10,7)	**.031**	**1.03**	**3.91**	2.53	2.60	2.42	3.53	2.62	2.33
(11,2)	**.022**	**1.44**	3.17	2.52	2.61	2.67	2.55	2.66	2.66
(11,9)	**.022**	**1.44**	2.73	2.90	2.67	2.99	2.55	2.90	2.59
(15,1)	**.006**	1.42	2.70	2.69	2.84	2.76	*3.60*	*2.38*	*2.00*
(15,4)	**.006**	1.42	2.82	2.62	2.62	2.99	*2.23*	*1.83*	*2.05*
(15,7)	**.006**	1.42	3.14	3.16	2.39	2.77	*2.02*	*2.03*	*1.78*
(15,8)	**.006**	1.42	2.80	2.56	2.35	2.35	*2.31*	*2.13*	*2.62*
(15,11)	**.006**	1.42	3.10	3.03	2.63	2.69	*2.19*	*2.20*	*2.24*
(15,14)	**.006**	1.42	3.14	2.79	2.69	2.53	*2.32*	*2.38*	*2.17*
(17,3)	**.002**	1.42	2.77	2.63	*2.21*	*2.21*	*2.04*	*2.21*	*2.36*
(17,5)	**.002**	1.42	3.15	2.65	*2.28*	*2.20*	*2.08*	*2.71*	*2.30*
(17,6)	**.002**	1.42	2.71	2.71	*2.67*	*2.33*	*2.71*	*1.94*	*2.28*
(17,11)	**.002**	1.42	3.24	2.62	*2.42*	*2.34*	*2.28*	*1.94*	*2.58*
(17,12)	**.002**	1.42	2.86	2.53	*1.99*	*2.77*	*2.39*	*2.41*	*2.00*
(17,14)	**.002**	1.42	2.55	2.73	*1.91*	*2.23*	*2.19*	*2.20*	*2.39*

TABLE 1. Full-period exponential sums, $|C(\mathrm{Per}(x_n), \cdot)|/\sqrt{\mathrm{Per}(x_n)}$, for various additive lagged-Fibonacci sequences of the form $x_n = x_{n-k} + x_{n-l} \pmod{2^m}$. Rows are indexed by l and columns by m. See text for further details.

problem among different processors is conceptually trivial via the independent trajectories. However, ensuring the reproducibility of this computation on an asynchronous MIMD ma-

	m								
(l,k)	1	2	3	4	5	6	7	8	9
(2,1)	**0.58**	**0.91**	**1.03**	**1.09**	**1.48**	**1.44**	**1.83**	1.75	1.94
(3,1)	**0.76**	**1.20**	**1.49**	**1.90**	**2.11**	**2.66**	**2.61**	2.69	3.06
(3,2)	**0.76**	**1.20**	**1.71**	**2.06**	**2.21**	**2.22**	**3.05**	2.76	2.87
(4,1)	**1.03**	**1.32**	**1.84**	**2.70**	**3.38**	**3.39**	**4.05**	2.73	3.05
(4,3)	**1.29**	**1.39**	**2.16**	**2.66**	**2.96**	**3.39**	**3.97**	3.54	3.83
(5,2)	**0.90**	**1.63**	**2.55**	**2.92**	**3.49**	**4.05**	**4.57**	2.72	3.04
(5,3)	**0.54**	**1.63**	**2.63**	**2.95**	**3.58**	**4.05**	**4.82**	3.12	3.62
(6,1)	**0.88**	**1.64**	**2.76**	**4.02**	**3.90**	3.01	2.65	2.70	3.06
(6,5)	**1.01**	**1.53**	**2.92**	**4.06**	**4.00**	3.04	3.91	3.08	3.05
(7,1)	**1.06**	**1.64**	**2.98**	**3.52**	3.04	3.03	2.82	2.87	2.67
(7,3)	**0.89**	**1.74**	**3.13**	**3.93**	2.63	2.41	2.79	2.88	2.95
(7,4)	**0.80**	**1.65**	**2.86**	**3.78**	2.78	3.20	2.88	2.77	2.95
(7,6)	**0.71**	**2.02**	**3.02**	**3.60**	3.03	2.69	3.06	2.85	2.82
(9,4)	**0.84**	**2.16**	**3.46**	2.72	2.64	2.84	3.39	2.91	2.85
(9,5)	**0.75**	**1.95**	**3.79**	2.83	3.08	3.05	2.83	2.91	2.95
(10,3)	**1.31**	**1.85**	**4.05**	2.75	3.04	2.67	2.94	3.06	2.93
(10,7)	**1.06**	**2.02**	**3.94**	2.67	2.86	2.57	3.55	2.81	2.48
(11,2)	**1.06**	**2.07**	3.28	2.73	2.66	2.72	2.82	2.68	2.94
(11,9)	**0.86**	**2.02**	2.98	2.99	2.68	3.00	2.83	3.01	3.07
(15,1)	**1.35**	2.10	2.86	2.75	2.89	2.90	*3.71*	*2.62*	*2.12*
(15,4)	**1.11**	2.10	3.25	2.76	2.65	3.08	*2.47*	*1.94*	*2.18*
(15,7)	**0.87**	1.99	3.15	3.35	2.62	2.99	*2.06*	*2.06*	*2.18*
(15,8)	**0.77**	2.19	2.89	2.59	2.65	2.46	*2.56*	*2.33*	*2.76*
(15,11)	**1.03**	2.17	3.33	3.07	2.71	2.78	*2.49*	*2.23*	*2.55*
(15,14)	**1.28**	2.16	3.19	2.91	2.77	2.67	*2.34*	*2.49*	*2.45*
(17,3)	**1.36**	2.19	2.83	2.70	*2.28*	*2.29*	*2.21*	*2.28*	*2.39*
(17,5)	**0.75**	2.16	3.23	2.77	*2.37*	*2.30*	*2.42*	*2.72*	*2.57*
(17,6)	**0.61**	2.15	2.81	2.73	*3.10*	*2.37*	*2.72*	*2.28*	*2.33*
(17,11)	**0.65**	2.45	3.33	2.82	*2.49*	*2.37*	*2.54*	*1.99*	*2.64*
(17,12)	**0.71**	2.20	3.10	2.73	*2.10*	*2.83*	*2.52*	*2.48*	*2.06*
(17,14)	**1.32**	2.13	2.55	2.90	*2.28*	*2.27*	*2.50*	*2.54*	*2.64*

TABLE 2. Extremal values of exponential sums, $\max_{0<j\leq \mathrm{Per}(x_n)} \dfrac{|C(i,j)|}{\sqrt{\mathrm{Per}(x_n)}}$, for various additive lagged-Fibonacci sequences of the form $x_n = x_{n-k} + x_{n-l} \pmod{2^m}$. Rows are indexed by l and columns by m. See text for further details.

chine is not nearly as easy. Since a particle may create new particles, this leads to new

trajectories to be followed. It is common practice to place the information from a particle creation into a computational queue. The queued particles are then processed when a free processor becomes available. Reproducibility requires that particles are put into the queue with information sufficient for the pseudorandom number generator to produce the same stream of pseudorandom numbers regardless of what processor or in what order the particle is processed. When using the additive lagged-Fibonacci generator, one need only provide a unique K for each particle to ensure reproducibility in this very general sense.

In general, if one is computing on an arbitrary asynchronous MIMD machine, it is desirable to be able to produce a child process identifier, K, that is guaranteed to be distinct from others created elsewhere in the computation. In addition, the assignment of K should be a local computation based only on the parent's process identifier. This is easily accomplished by associating the parallel processes with a binary tree. When the process for node K is required to create n children, it does so by assigning the n nodes closest and below it on the binary tree. This assures a local computation. In particular, if the process assigned to node K has two children, they receive nodes $2K+1$ and $4K+2$.

This procedure does not totally solve the problem of the reuse of ECs in an asynchronous MIMD computation. It is possible to have each particle in a computation create one child particle and hence rapidly descend down the $(l-1)(m-1)+1$ levels of the process binary tree. However, this seems to be a very unlikely situation, as even the relatively small CMSSL generator has 497 levels!

3.2 Equivalence Classes and Exponential Sums.

We now turn to the analysis of exponential sum cross-correlations among ECs. If $\{x_n\}$ and $\{y_n\}$ come from different ECs, we know we can find an offset, j, so that $\mathbf{x}_n$ and $\mathbf{y}_{n+j}$ agree in their least-significant bit. In fact, if $\mathbf{x}_n$ and $\mathbf{y}_n$ are their respective EC representatives, they already agree in their least-significant bit. Now assume that we are working with recurrences from equation (1), i.e., $M = 2^m$, and we have the full-period exponential sum as a function of m, i.e., $C(\mathrm{Per}(x_n), \cdot) = F(m)$. The difference, $\tilde{z}_n = x_n - y_{n-j}$ is even and may be written as $\tilde{z}_n = 2z_n$, where z_n is a maximum possible period additive lagged-Fibonacci sequence modulo 2^{m-1}. Thus

$$\begin{aligned}
\sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^m}\tilde{z}_n} &= \sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^m}2z_n} = \\
\sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} &= \sum_{n=0}^{2\mathrm{Per}(z_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} = \\
2\sum_{n=0}^{\mathrm{Per}(z_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} &= 2F(m-1).
\end{aligned} \tag{13}$$

From above we conjectured that $F(m) = O(2^{m/2})$, so that if $\mathbf{x}_n$ and $\mathbf{y}_{n+j}$ agree in their r least-significant bits, then the full-period exponential sum cross-correlation, equation (12), is $2^{r/2}$ times bigger that the full-period exponential sum, equation (9).

This gives us a clear understanding of how different ECs are related, since one way to see how many least-significant bits of overlap there are between $\{x_n\}$ and $\{y_n\}$ is to place

them in canonical form. Because of the periods of the different bits, $\{x_n\}$ and $\{y_n\}$ can be made to agree in their r least-significant bits if and only if their EC representatives agree in their r least-significant bits. This means that a local scheme of computing child process numbers based on the mapping of parallel processes onto the binary tree will always produce children with cross-correlations as small as possible. In fact, this analysis allows us to modify the assignment of process numbers in appropriate ways to avoid large cross-correlations if a particular computation chooses related ECs poorly in this respect. In our implementation of this generator, we did not use this assignment procedure for child processes for the reason of correlations in lesser-significant bits in sequences all started from the EC representatives. One solution is to apply some pseudorandom power of $\mathbf{A}$ to a given seed, but in our implementation we believe we have a more elegant solution, [3].

4. Discussion and Conclusions.

We have provided the theoretical background for the use of a two-term additive lagged-Fibonacci pseudorandom number generator in the most general parallel setting. The algorithms are based on the realization that equation (1) produces a vast number of full-period cycles. These cycles can be explicitly chosen through the calculation of an appropriate seed. We have also provided a simple and general local computation to produce child ECs from parents. In addition, we have analyzed the theoretical quality of these sequences and have understood the exponential sum cross-correlations among different ECs.

Still open is a proof that the canonical form given in equation (4) is always attainable through a judicious choice of the least-significant bits. We also have only conjectured as to the value of the principle full-period exponential sum. A proof would be much more satisfying.

In addition to this work, we have provided an implementation of these ideas, [3]. This implementation is very similar to the ideas presented here except that certain compromises between access to all of the ECs and speed of seeding were made. This implementation is based directly on the integer recursion in (1) and produces $[0,1)$ random variables by $u_n = x_n/M$. A more direct approach is to work directly with floating-point numbers and take equation (1) modulo 1. This can be done while still ensuring absolute EC integrity by providing zero valued guard bits in the mantissa. For our simple three term recursions with ± 1 coefficients, a mantissa of length s bits can hold floating-point values in the top $s-1$ bits without risking an operation that changes the EC.

References

1. R. P. Brent, *Uniform Random Number Generators for Supercomputers*, Proceedings Fifth Australian Supercomputer Conference, SASC Organizing Committee, 1992, pp. 95–104.
2. R. P. Brent, *On the periods of generalized Fibonacci recurrences*, in the Press, Math. Comput. (1994).
3. S. A. Cuccaro, D. V. Pryor, M. Mascagni and M. L. Robinson, *Implementation and usage of a portable and reproducible parallel pseudoprandom number generator*, preprint, SRC-TR-94-116.
4. Feller, *An Introduction to Probability Theory and Its Application, Volume I*, Third Edition, John Wiley and Sons, New York, 1968.
5. Feller, *An Introduction to Probability Theory and Its Application, Volume II*, Second Edition, John Wiley and Sons, New York, 1971.
6. S. W. Golomb, *Shift Register Sequences*, Revised Edition, Aegean Park Press, Laguna Hills, California, 1982.
7. R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics*, Fourth Edition, MacMillan Publishing Co., Inc., New York, 1978.

8. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Second edition*, Addison-Wesley, Reading, Massachusetts, 1981.
9. N. M. Korobov, *Exponential sums and their applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.
10. L. Kuipers and H. Niederreiter, *Uniform distribution of sequences*, John Wiley and Sons, New York, 1974.
11. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, London, New York, 1986.
12. G. Marsaglia, *A current view of random number generators*, Computing Science and Statistics: Proceedings of the XVIth Symposium on the Interface, 1985, pp. 3–10.
13. G. Marsaglia and L.-H. Tsay, *Matrices and the structure of random number sequences*, Linear Alg. and Applic. **67** (1985), 147–156.
14. M. Mascagni, S. A. Cuccaro, D. V. Pryor and M. L. Robinson, *Recent Developments in Parallel Pseudorandom Number Generation*, Volume II, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing (D. E. Keyes, M. R. Leuze, L. R. Petzold, D. A. Reed, ed.), SIAM, Philadelphia, Pennsylvania, 1993, pp. 524–529.
15. H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, SIAM, Philadelphia, Pennsylvania, 1992.
16. J. Spanier and E. M. Gelbard, *Monte Carlo Principles and Neutron Transport Problems*, Addison-Wesley, Reading, Massachusetts, 1969.
17. R. C. Tausworthe, *Random numbers generated by linear recurrence modulo two*, Math. Comput. **19** (1965), 201–209.

Supercomputing Research Center; Institute for Defense Analyses; 17100 Science Drive; Bowie, Maryland 20715-4300 USA

E-mail address: mascagni@super.org, cuccaro@super.org, pryor@super.org, robinson@super.org