PARALLEL PSEUDORANDOM NUMBER GENERATION USING ADDITIVE LAGGED-FIBONACCI RECURSIONS

Michael Mascagni
M. L. Robinson
Daniel V. Pryor
Steven A. Cuccaro

Supercomputing Research Center, I.D.A.

Abstract. We study the suitability of the additive lagged-Fibonacci pseudorandom number generator for parallel computation. This generator has a 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. We call these different full-period cycles equivalence classes. We show how to enumerate the equivalence classes and how to compute seeds to select a given equivalence class. The use of these equivalence classes gives an explicit parallelization suitable for a fully reproducible asynchronous MIMD implementation. To explore such an implementation we introduce an exponential sum measure of quality for the additive lagged-Fibonacci generators used in serial or parallel. We then prove the first non-trivial results we are aware of on this measure of quality.

1. Introduction.

In Knuth's well known exposition on pseudorandom number generation [5], several methods of generation are considered. Among these is the additive lagged-Fibonacci pseudorandom number generator:

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

This generator is defined by the modulus, M, the register length, ℓ, and the lag, k. When M is prime, periods as large as $M^\ell - 1$ 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 [9], the additive lagged-Fibonacci generator with power-of-two modulus was one among that were tested. 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

1991 *Mathematics Subject Classification.* 11L07, 65C10, 65Y05.

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

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, [11]. 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 $\omega_n = \omega_{n-k} + \omega_{n-\ell} \pmod 1$. Here the $\omega_n \in [0, 1)$ are floating-point numbers. Besides being able to compute directly 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 versatile, and that there are several good reasons for exploring the additive lagged-Fibonacci generator with power-of-two modulus to find an effective parallel 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.
(3) The streams generated in parallel must seem "independent."

Property (2) is very important to computational scientists. When doing Monte Carlo calculations on new machines, exact agreement with previous and trusted calculations is 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 different machines. This demanding definition of reproducibility ensures the portability of a parallel generator to any parallel machine. This is a rather lofty goal, but one that we show is accessible to the additive lagged-Fibonacci generator.

Property 3 has not been adequately addressed in a general sense. One partial solution is to use exponential sums as theoretical measures of quality for both the serial and the parallel use of pseudorandom number generators. With additive lagged-Fibonacci generators modulo a power-of-two, this leads to the consideration of full-period exponential sums that have not been previously studied in the literature. In addition, empirical measures of quality are important in our understanding of property 3. Such empirical measures of the parallel use of these generators has already lead to improvements in their implementation, [3].

The plan of the paper is as follows. In §2 we review the conditions on 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 some maximal-period cycle. The

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

large number of equivalence classes (ECs) that result is the basis for our parallelization. We then describe an enumeration of all of the ECs. 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. §3 is devoted to proving some results for the full-period exponential sums measures of quality. These include an exact value for a collection of full-period sums and a non-trivial upper bound for the general case of interest. This last bound is weak in general, but with additional empirical data, it can be used to get much better bounds for specific generators. In addition, 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 $M^\ell - 1$ is possible provided that the characteristic polynomial $f(x) = x^\ell - x^k - 1$ is primitive modulo M. In the case of interest, when $M = 2^m$, the maximum possible period is $(2^\ell - 1)2^{m-1}$. In general, a linear recurrence modulo 2^m has period $(2^\ell - 1)2^{m-1}$ if and only if the following three conditions hold:

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

For a proof see [10] or exercise 11 of §3.2.2 in [5]. We are only interested in simple additive lagged-Fibonacci generators, so our characteristic polynomial, $f(x) = x^\ell - x^k - 1$, is a trinomial. In this case Brent recently proved, [2], that when $f(x)$ is a trinomial, primitivity of $f(x)$ modulo 2 and $\ell > 2$ suffice to give the maximum possible period of $(2^\ell - 1)2^{m-1}$.

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 $(2^\ell - 1)2^{m-1}$, i.e., equation (1) satisfies Brent's conditions. Because of Brent's recent results the following qualitative description of the cycle structure is rigorous, [2]. 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 $2^\ell - 1$, [4, 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 $2^\ell - 1$ 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

[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 [4].

has period $2^\ell - 1$. Since we have the maximum-possible period, the carries must also have period $2^\ell - 1$. Thus a particular carry will force a period $2^\ell - 1$ impulse response, after which the periodic repeat of the first carry will zero the impulse response. Thus each carry will produce a period $(2^\ell - 1)2$ response made up of a $2^\ell - 1$ length maximal-period sequence followed by $2^\ell - 1$ zeros. Because we obtain the maximum possible period, the superposition of these sequences also has period $(2^\ell - 1)2$. 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 $(2^\ell - 1)2^{m-1}$ is considerably smaller than the number of nonzero fills, $2^{\ell m} - 1$. 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, [10, 2]. 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 $(2^\ell - 1)2^{\ell(m-1)}$. Since each of these seeds is in a maximum possible period cycle, there must be

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

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 least-significant bits, one need only advance the seed at most $2^\ell - 2$ 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 most-significant bits. Given that we want simultaneously

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

to fix the least-significant bits and remain in the same EC, we must leap ahead in the generator's cycle some multiple of $2^\ell - 1$, 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 $\mathbf{x}_n = [x_n, x_{n-1}, \ldots x_{n-\ell+1}]^T$ 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 $\ell \times \ell$ matrix $\mathbf{A}$ defined by:

$$\mathbf{A} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} k \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \ell \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \tag{3}$$

$$\begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 & 1 \\ 1 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 1 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 1 & 0 \end{pmatrix}$$

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, $\mathbf{A}^{2^\ell - 1} \equiv \mathbf{I} \pmod 2$. Let us call $\mathbf{J} = \mathbf{A}^{2^\ell - 1}$, 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 full period. Additionally, application of this procedure to seeds from different ECs will produce different canonical form seeds.

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 $\ell \times m$, while the number of ECs is $E = 2^{(\ell-1)(m-1)}$. Thus a set of $(\ell - 1)(m - 1)$ bits specifies a unique EC. Our canonical form has already

[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.

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	
$\square$	$\square$	...	$\square$	$b_{0\,\ell-1}$	$x_{\ell-1}$
$\square$	$\square$	...	$\square$	$b_{0\,\ell-2}$	$x_{\ell-2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\square$	$\square$	...	$\square$	$b_{0\,1}$	x_1
0	0	...	0	$b_{0\,0}$	x_0

This enumeration leaves exactly $(\ell-1)(m-1)$ bits to be specified in the canonical form and yields exactly $E = 2^{(\ell-1)(m-1)}$ 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 $\mathbf{J} = \mathbf{A}^{2^\ell-1}$ and the period of the shift-register is $2^\ell - 1$, $b_1^{\mathbf{J}}$ 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 $b_1^{\mathbf{J}}$ cannot be 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 $\ell - 1$ 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	$b_{0\,\ell-1}$	$x_{\ell-1}$
0	$\square$	...	$\square$	$b_{0\,\ell-2}$	$x_{\ell-2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\square$	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. A simple proof of this fact can be found in [12].

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 $(\ell-1)(m-1)$ bit equivalence class number into $(\ell-1)(m-1)$ 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). We have implemented a seeding scheme based on these explicit EC computations for the case of the additive lagged-Fibonacci used in Thinking Machines' CMSSL. Here the recurrence is:

$$x_n = x_{x-5} + x_{n-17} \pmod{2^{32}}. \tag{6}$$

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 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 identifier, K. This K is then used to select the Kth EC for this process.[5] This procedure works without difficulty provided the parameters for the generator are chosen so that no K is required in the computation that exceeds $2^{(\ell-1)(m-1)}-1$.

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 problem among different processors is conceptually trivial via the independent trajectories. However, ensuring the reproducibility of this computation on an asynchronous MIMD machine 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 queued 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 all possible 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.

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

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!

2.4 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, [5]. 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, [8, 7, 14]. In turn, the discrepancy appears explicitly in the Koksma-Hlawka bound on numerical integration error. This is very important since numerical integration is **the** fundamental Monte Carlo application.

The exponential sum auto-correlation for a modulo M sequence, $\{x_n\}$, is defined as:

$$A(i,j) = \sum_{n=0}^{i-1} e^{\frac{2\pi\sqrt{-1}}{M}(x_n - x_{n-j})}. \tag{7}$$

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 types of sums in order to calculate or bound them is fundamental to many areas of number theory, [6].

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 number sequences, $\{x_n\}$ and $\{y_n\}$. Their exponential sum cross-correlation is given by:

$$C(i,j) = \sum_{n=0}^{i-1} e^{\frac{2\pi\sqrt{-1}}{M}(x_n - y_{n-j})}. \tag{8}$$

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, 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 (8) gives us a qualitative tool to explore relationships between related ECs. This approach is discussed in §3.

3. Some Properties of Full-Period Sums.

To analyze these additive lagged-Fibonacci recursions using full-period exponential sums from (7) we must consider such sums with $M = 2^m$. Exponential sums over finite fields have yielded quite remarkable results on the quality of many types of pseudorandom number generators, [8, 13, 14]. The best results for these sums when the recursions are modulo a power-of-two, however, are considerably less informative. For example, the general bound for a ℓ-term recursion modulo 2^m is $O(2^{\frac{\ell \times m}{2}})$, [13]. Given that there are $\mathrm{Per}(x_n) = (2^\ell - 1)2^{m-1}$ terms of magnitude one in this sum, a trivial upper bound is $\mathrm{Per}(x_n) << O(2^{\frac{\ell \times m}{2}})$. To our knowledge, no bounds on these types of exponential sums exist that take into account the relatively short period of these sequences. In the §3.1 we calculate the sum of all of the full-period exponential sums over all ECs. In §3.2 we show how the exponential sum cross-correlations among different ECs depends on the similarity of the two ECs' representative and leads to sums related to full-period sums with smaller powers-of-two. Finally, in §3.3 we prove the first non-trivial upper bounds we are aware of for these exponential sums. The order of magnitude of these sums is substantially inferior to the analogous sums over finite fields. We hope these humble results encourage experts to improve upon them.

3.1 Full-Period Exponential Sums over All Equivalence Classes.

One exact calculation that can be done relating to the full-period exponential sum measure of quality given in (7) is the exact determination of all of the full-period sums over all of the ECs. When $m = 1$ the sequences are just maximal-period shift-register sequences and there is only $E = 2^{(\ell-1)(m-1)} = 1$ equivalence class. It is well known that in this case $C(\mathrm{Per}(x_n), \cdot) = -1$, [4].

To compute these sums in general we will use a well-known trick. Consider the sum:

$$\mathcal{S}_m = \sum_{k \in \mathcal{K}} e^{\frac{2\pi\sqrt{-1}}{2^m} k} \tag{9}$$

where $\mathcal{K}$ is the set of all nonzero ℓ-tuples. This sum has $2^{\ell \times m} - 1$ terms. Thus $\mathcal{S}_m = -1$ since k takes on the value of each residue modulo 2^m exactly 2^ℓ times, except for 0, which is taken on only $2^\ell - 1$ times. We will now show how to rewrite $\mathcal{S}_m$ as a sum over full-periods over all ECs.

First a simple example, $m = 2$. We can rearrange $\mathcal{S}_2$ into a sum over all full-period cycles modulo 4 plus the rest. Recall that a cycle is full-period if and only if at least one of its elements is odd. For a general m there are $E_m \times \mathrm{Per}_m(x_n) = 2^{(\ell-1)(m-1)} \times (2^\ell - 1)2^{m-1} = (2^\ell - 1)2^{\ell(m-1)}$ such elements in these full-period cycles. In particular, for $m = 2$ there are $E_2 = 2^{\ell-1}$ ECs of length $\mathrm{Per}_2(x_n) = (2^\ell - 1)2$. We can characterize the rest of the elements as those that do not have at least one odd element, i.e., they have zero least-significant-bit. There are $2^\ell - 1$ of these accounting for $E_2 \times \mathrm{Per}_2(x_n) + 2^\ell - 1 = 2^{2\ell} - 1$ terms, all the nonzero 2-tuples. Thus:

$$\mathcal{S}_2 = -1 = \sum_{E_2} \sum_{\mathrm{Per}_2(x_n)} e^{\frac{2\pi\sqrt{-1}}{4} x_n} + \sum_{E_1} \sum_{\mathrm{Per}_1(x_n)} e^{\frac{2\pi\sqrt{-1}}{4} x_n}. \tag{10}$$

The last sum is over all ECs modulo 2, $E_1 = 1$, over the full period, $Q_1(x_n) = 2^\ell - 1$, over a $x_n = 2 \times y_n$ where y_n is a maximal-period shift-register sequence modulo 2. This

is easily recognized as $C(\mathrm{Per}(x_n),\cdot) = -1$, and thus $\sum_{E_2}\sum_{\mathrm{Per}_2(x_n)} e^{\frac{2\pi\sqrt{-1}}{4}x_n} = 0$. Let us denote a sum over all ECs and over all full-period cycles modulo 2^m as $S(m)$. Thus the first term in (10) is $S(2)$ and the second term is $S(1)$.

Thus we see a pattern, and it is clear that $\mathcal{S}_m = \sum_{k=1}^{m} S(k) = S(m) + \mathcal{S}_{m-1}, m > 1$. We have proven that $\mathcal{S}_m = \mathcal{S}_1 = S(1) = -1$, and so by the previous equation, $S(m) = 0, m > 1$, and we have proven that

$$S(m) = \begin{cases} 0, & \text{for } m > 1 \\ -1, & \text{for } m = 1. \end{cases}$$

This means that for the exponential sum over all full-period ECs of the additive lagged-Fibonacci generator yields zero except when $m = 1$. In this case we have the sum equal one, which follows from more classical results of maximal-period shift-register sequences, [4].

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., $M = 2^m$, with a fixed recursion (i.e., ℓ and k given) and have the full-period exponential sum as a function of m, i.e., $C(\mathrm{Per}(x_n),\cdot) = F(m)$. In addition, assume that $\mathbf{x}_n$ and $\mathbf{y}_{n+j}$ agree only in their least-significant bit. We notice that 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

$$\begin{aligned} \sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^m}\tilde{z}_n} &= \sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^m}2z_n} = \\ \sum_{n=0}^{\mathrm{Per}(x_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} &= \sum_{n=0}^{2\mathrm{Per}(z_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} = \\ 2\sum_{n=0}^{\mathrm{Per}(z_n)-1} e^{\frac{2\pi i}{2^{m-1}}z_n} &= 2F(m-1). \end{aligned} \tag{11}$$

In general, if there is an offset, j, that makes $\mathbf{x}_n$ and $\mathbf{y}_{n+j}$ agree in their r least-significant bits, we have that (11) will equal $2^r \times F(m-r)$. 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 processor 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 **A** to a given seed, but in our implementation [15] we believe we have a more elegant solution.

3.3 Bounds on Full-Period Exponential Sums of Maximal Period.

Throughout this subsection, the notation $\max_{\{x_n\}}$ means that the maximum is taken over all maximal period sequences $\{x_n\}$ satisfying a recursion relation of the form

$$x_n = x_{n-k} + x_{n-\ell} \pmod{2^m}, \quad \ell > k,$$

where ℓ and k are fixed. Our goal in this subsection is to obtain nontrivial upper bounds in certain cases for $|U(m)|$ where

$$U(m) = \max_{\{x_n\}} \sum_{j=0}^{\mathrm{Per}(x_n)} e^{\frac{2\pi\sqrt{-1}}{2^m} x_j}.$$

It is convenient to introduce the quantity

$$Z(m) = \max_{\{x_n\}} \#\{j | x_j = 0 \text{ and } 0 \le j < \mathrm{Per}(x_n)\}.$$

Our first result bounds $|U(m)|$ in terms of $Z(m/2)$ and suggests the possibility of a descent.

Proposition 1. *If $1 < m \in \mathbb{Z}^+$, then*

$$|U(m)| \le 2^{\lceil m/2 \rceil} Z(\lfloor m/2 \rfloor)$$

Proof. Let $\{x_n\}$ be a linear recurrent sequence modulo 2^m that produces the full-period exponential sum $U(m)$. Note that if $\{x_n\}$ is viewed modulo $2^{\lceil m/2 \rceil}$, then it has period $H = (2^\ell - 1)2^{\lceil m/2 \rceil - 1}$. It follows that

$$x_{n+H} - x_n = 2^{\lceil m/2 \rceil} y_n,$$

where $\{y_n\}$ is a maximal period sequence defined modulo $2^{\lfloor m/2 \rfloor}$ that satisfies the same recurrence as $\{x_n\}$.

Now observe if μ is a non-negative integer, then

$$x_{n+(\mu+1)H} = x_{n+\mu H} + 2^{\lceil m/2 \rceil} y_{n+\mu H} = x_{n+\mu H} + 2^{\lceil m/2 \rceil} y_n,$$

by the periodicity of $\{y_n\}$. It follows by induction that

$$x_{n+\mu H} = x_n + \mu 2^{\lceil m/2 \rceil} y_n.$$

Recall that each non-negative integer j with $0 \le j < (2^\ell - 1)2^{m-1}$ can be written uniquely as $j = N + \mu H$, where $0 \le N < H$, and $0 \le \mu < 2^{\lfloor m/2 \rfloor}$. It follows at once that

$$\sum_{j=0}^{\mathrm{Per}(x_n)} e^{\frac{2\pi\sqrt{-1}}{2^m} x_j} = \sum_{N=0}^{H-1} \sum_{\mu=0}^{2^{\lfloor m/2 \rfloor}-1} e^{\frac{2\pi\sqrt{-1}}{2^m} x_{N+\mu H}}.$$

Hence,

$$U(m) = \sum_{N=0}^{H-1} \sum_{\mu=0}^{2^{\lfloor m/2 \rfloor}-1} e^{\frac{2\pi\sqrt{-1}}{2^m}(x_N + \mu 2^{\lceil m/2 \rceil} y_N)}.$$

Further algebraic manipulation yields

$$U(m) = \sum_{N=0}^{H-1} e^{\frac{2\pi\sqrt{-1}}{2^m} x_N} \sum_{\mu=0}^{2^{\lfloor m/2 \rfloor}-1} \left(e^{\frac{2\pi\sqrt{-1}}{2^{\lfloor m/2 \rfloor}} y_N} \right)^\mu.$$

Now the innermost sum is a geometric series that vanishes unless y_N is zero, in which case it equals $2^{\lfloor m/2 \rfloor}$. Thus

$$U(m) = 2^{\lfloor m/2 \rfloor} \sum_{\substack{N=0 \\ y_N = 0}}^{H-1} e^{\frac{2\pi\sqrt{-1}}{2^m}(x_N)}.$$

Estimating the sum trivially, we have that

$$|U(m)| \le 2^{\lfloor m/2 \rfloor} \#\{N | y_N = 0 \text{ and } 0 \le N < H\}.$$

When m is even, H is the period of $\{y_n\}$. In this case,

$$|U(m)| \le 2^{m/2} Z(m/2),$$

and the proposition follows for even m. When m is odd, H is twice the period of $\{y_n\}$. In this case,

$$|U(m)| \le 2^{\lfloor m/2 \rfloor} 2 Z(m/2),$$

proving the proposition when m is odd. $\square$

Note that the trivial upper bound for $|U(m)|$ is $(2^\ell - 1)2^{m-1}$, the common period of the linear recurrent sequences under consideration. Note that if the values of a maximal period $\{x_n\}$ were approximately uniformly distributed over the congruence classes mod $2^{m/2}$, then Proposition 1 would produce a decidedly nontrivial bound for $|U(m)|$. Indeed, if $|Z(\lfloor m/2 \rfloor)|$ is well-approximated by the expected $(2^\ell - 1)/2$, then Proposition 1 would yield an estimate of the form $|U(m)| = O(2^{m/2+\ell})$. This is much better that the trivial bound $|U(m)| = O(2^{m+\ell})$ by a factor of $2^{m/2}$, although it is still a factor of $2^{\ell/2}$ greater than the empirically conjectured $O(\sqrt{\mathrm{Per}(x_n)})$, [12].

Currently, however, we have only the following estimate for $|Z(m)|$.

Proposition 2. *For any $m \in \mathbb{Z}^+$, we have*

$$|Z(m)| \leq \frac{2^\ell - 1}{2} + \frac{(|U(m)| + \cdots + |U(1)|)}{2}.$$

Proof. Let $\delta(x)$ be the usual Kronecker function which takes the value 1 at the origin and vanishes elsewhere. Let $\{x_n\}$ be the maximal period linear recurrent sequence modulo 2^m that vanishes $Z(m)$ times over a full period. Then

$$Z(m) = \sum_{j=0}^{\mathrm{Per}(x_n)} \delta(x_j) = \sum_{j=0}^{\mathrm{Per}(x_n)} \frac{1}{2^m} \sum_{h=0}^{2^m-1} e^{\frac{2\pi\sqrt{-1}}{2^m} h x_n}.$$

Changing the order of summation yields

$$Z(m) = \frac{1}{2^m} \sum_{h=0}^{2^m-1} \sum_{j=0}^{\mathrm{Per}(x_n)} e^{\frac{2\pi\sqrt{-1}}{2^m} h x_n}.$$

The contribution from the $h = 0$ term is simply $\mathrm{Per}(x_n)/2^m$, which equals $(2^\ell - 1)/2$. When h is odd, $\{hx_n\}$ is a maximal period sequence modulo 2^m that satisfies the same recurrence as $\{x_n\}$, and there are precisely 2^{m-1} odd values of h. Thus the terms for which h is odd contribute at most $|U(m)|/2$. When h is precisely divisible by 2^k, then the innermost sum can be viewed as 2^k copies of a full-period sum modulo 2^{m-k}, and there are precisely 2^{m-k-1} such values of h. Such terms contribute at most $|U(m-k)|/2$. The proposition is proved. $\square$

Note that applying Propositions 1 and 2 recursively yields the estimate

$$|U(m)| \leq 2^{\lceil m/2 \rceil} \left(\frac{2^\ell - 1}{2} + 1/2 \sum_{j=0}^{\lfloor m/2 \rfloor - 1} |U(\lfloor m/2 \rfloor - j)| \right).$$

Iterating this approach down to the case where $m = 1$ does not in general improve on the trivial bound mentioned earlier. However, suppose the actual value of $Z(m)$ could be obtained by brute force on a computer. Then one could apply Proposition 1 and obtain nontrivial estimates for $|U(2m)|$, even when the size of $2m$ is so large that a brute force attack on $U(m)$ itself is not feasible. For example, we have computed $|Z(2)|$ in the case where $\ell = 17$ and $k = 5$. Applying Proposition 1 then yields the nontrivial result

$$|U(4)| \leq (\mathrm{Per}(x_n))^{.9003},$$

in this special case.

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.

The bounds we have provided for these full-period exponential sums are surely far from the best possible. In fact, if we consider a fixed recursion modulo 2^m, an exhaustive computation of all of the exponential sums can be undertaken. This provides a concrete number to place in this cascade of inequalities that empirically improves what can be proven for a particular generator. For example, in the case where $\ell = 17$ and $k = 5$, the CMSSL generator, we have computed exhaustively that $Z(2) \leq 65790$. This motivates further research into both computational and theoretical improvements in the determination of upper bounds for these full-period exponential sums.

In addition to this work, we have provided an implementation of these ideas, [15]. This implementation is very similar to the ideas presented here, except that certain compromises were made between access to all of the ECs and speed of seeding. 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.

Acknowledgments

The authors would like to thank R. G. E. Pinch of Queen's College at Cambridge University, England, for suggesting the induction used in the main proof of §3.3.

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*, Math. Comput. **63** (1994), 389–401.
3. S. A. Cuccaro, M. Mascagni and D. V. Pryor, *Techniques for testing the quality of parallel pseudo-random number generators*, Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, Pennsylvania, 1995, pp. 279–284.
4. S. W. Golomb, *Shift Register Sequences*, Revised Edition, Aegean Park Press, Laguna Hills, California, 1982.
5. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Second edition*, Addison-Wesley, Reading, Massachusetts, 1981.
6. N. M. Korobov, *Exponential sums and their applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.
7. L. Kuipers and H. Niederreiter, *Uniform distribution of sequences*, John Wiley and Sons, New York, 1974.

8. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, London, New York, 1986.
9. 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.
10. G. Marsaglia and L.-H. Tsay, *Matrices and the structure of random number sequences*, Linear Alg. and Applic. **67** (1985), 147–156.
11. 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.
12. M. Mascagni, S. A. Cuccaro, D. V. Pryor and M. L. Robinson, *A fast, high-quality, and reproducible lagged-Fibonacci pseudorandom number generator*, in the press, Comput. Physics (1995).
13. H. Niederreiter, *Quasi-Monte Carlo methods and pseudo-random numbers*, Bull. Amer. Math. Soc. **84** (1978), 957–1041.
14. H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, SIAM, Philadelphia, Pennsylvania, 1992.
15. D. V. Pryor, S. A. Cuccaro, M. Mascagni and M. L. Robinson, *Implementation and usage of a portable and reproducible parallel pseudorandom number generator*, in Proceedings of Supercomputing '94, IEEE, 1994, pp. 311–319.
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, robinson@super.org pryor@super.org, cuccaro@super.org,