Fast and Accurate Computation using Stochastic Circuits

Armin Alaghi and John P. Hayes
Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science, University of Michigan
Ann Arbor, MI, 48109, USA
{alaghi, jhayes}@eecs.umich.edu

***Abstract*—Stochastic computing (SC) is a low-cost design technique that has great promise in applications such as image processing. SC enables arithmetic operations to be performed on stochastic bit-streams using ultra-small and low-power circuitry. However, accurate computations tend to require long run-times due to the random fluctuations inherent in stochastic numbers (SNs). We present novel techniques for SN generation that lead to better accuracy/run-time trade-offs. First, we analyze a property called progressive precision (PP) which allows computational accuracy to grow systematically with run-time. Second, borrowing from Monte Carlo methods, we show that SC performance can be greatly improved by replacing the usual pseudo-random number sources by low-discrepancy (LD) sequences that are predictably progressive. Finally, we evaluate the use of LD stochastic numbers in SC, and show they can produce significantly faster and more accurate results than existing stochastic designs.**

Keywords—Stochastic computing, Monte Carlo methods, Computer arithmetic, progressive precision.

I. INTRODUCTION

Some potentially beneficial applications of computer technology cannot be realized because they have extreme requirements for small size, high speed, or ultra-low power. Examples include real-time processing of sensory data for biomedical devices. An unconventional digital technology that is re-emerging as well-suited to such tasks is stochastic computing (SC) [5][1]. It has recently been successfully used to process images [2] and error-correcting codes [7], as well as to design artificial neural networks [3].

In its basic form, SC processes data in the form of long pseudo-random bit-streams that are treated as probabilities. It can implement arithmetic operations by means of tiny circuits, a feature it shares with analog computing. This allows SC to be used in parallel configurations to achieve high performance at very low area and power cost. Both analog and stochastic computing suffer from accuracy limitations. The accuracy of SC can—at least in principle—be improved by increasing bit-stream length, which, of course, increases run-time. The goal of both fast and accurate SC has remained elusive and poorly understood. We examine the speed-accuracy trade-off problem of SC, and propose several new solutions.

To recap SC's main features, it represents a number by an n-bit sequence called a *stochastic number* (SN) [1]. In the basic "unipolar" format, the numerical value p_X of the SN X is n_1/n, where n_1 is the number of 1s appearing in X. This

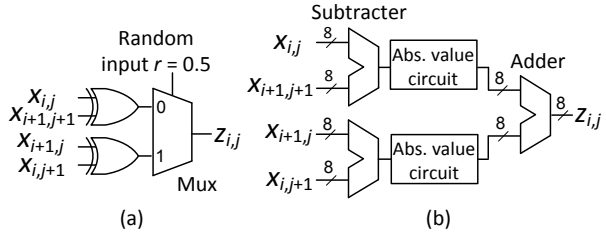

Figure 1. (a) Stochastic and (b) conventional edge detectors.

coding scheme has an obvious stochastic interpretation: p_X is the probability of a 1 appearing in X. It further hints that arithmetic operations can be performed on SNs via logical operations. Figure 1a shows a small stochastic circuit [2] that implements the Roberts cross formula

$$Z_{i,j} = 0.5 \times (|X_{i,j} - X_{i+1,j+1}| + |X_{i,j+1} - X_{i+1,j}|) \qquad (1)$$

for edge detection in visual images. A conventional non-SC or "binary" implementation of (1), as in Figure 1b, requires orders of magnitude more area and power.

The main factor limiting the use of SC is a need for long bit-streams, and hence long run-times, to overcome the random fluctuations inherent in the bit-stream patterns, and to achieve adequate levels of precision and accuracy. The *precision* of an SN X of length n may be defined as $\log_2 n$ bits, while the *accuracy* of X is its closeness to a target value denoted by p_X^*, which may be stated in terms of acceptable error bounds. To increase precision by 1 bit, an SN's length must be doubled. Accuracy targets, however, may demand even longer bit-streams.

Accuracy concerns have usually been addressed by ad hoc means [5][11]. A basic approach has been simply to increase SN length. This lowers performance and provides no guarantee of convergence to an acceptable result. A second approach is to use better stochastic number generators (SNGs) [9]. A typical SNG appears in Figure 2. The "random" number source is usually a linear feedback shift register (LFSR), which produces pseudo-random bit-streams (*m-sequences*) with good randomness properties [6]. A stochastic circuit often needs many SNGs, making them a significant contributor to overall hardware cost

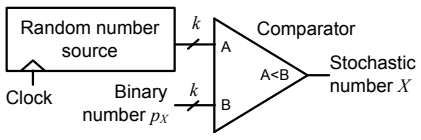

Figure 2. Stochastic number generator: the "random" number source may be a finite-state machine with random-like behavior.

978-3-9815370-2-4/DATE14/©2014 EDAA

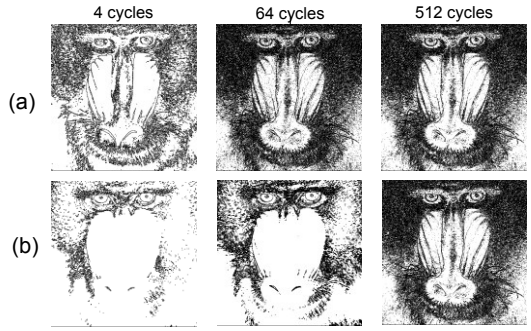

Figure 3. Progressive precision in edge detection at 4, 64 and 512 cycles with (a) LD sequences (good PP); (b) bad PP sequences.

[11]. These costs can be mitigated by sharing SNGs among different circuits, e.g., a few SNGs can be shared by many pixel processors in a vision chip [2].

This paper introduces a new method of accurate number generation for SC. It also analyzes, for the first time, an important property of SNs called *progressive precision* (PP), which enables accuracy to be traded for speed: if the first $k < n$ result bits of an n-bit stochastic computation provide a sufficiently good approximation to a desired result, the computation can be stopped early.

To exploit PP, we introduce a new class of random number sources that produce *low-discrepancy* (*LD*) sequences, in which 1's and 0's are uniformly spaced, so they do not suffer from random fluctuations. They are widely used in quasi-Monte Carlo (QMC) sampling [10][4], but have not been previously applied to SC. Other benefits of LD include deterministic error bounds and fast convergence. We present circuits employing LD sequences that are faster and more accurate than existing SC designs. For example, in image edge detection using LD sequences with good PP, as shown in Figure 3a, many edges are detected after only 4 clock cycles, so the computation can stop early in most cases, as opposed to full-scale computation with 512 clock cycles. But if bit-streams with bad PP are used, we get the output edges shown in Figure 3b, where many edges remain undetected even after 64 clock cycles. LFSR-based *m*-sequences often have poor PP because they have uneven spacing of 1's and 0's [6].

II. Accuracy of Stochastic Numbers

We first quantify the concepts of accuracy and error introduced informally above.

Definition 1. Let X be a stochastic number of length n with value p_X and let p_X^* denote its exact value. Then the *bit-error* of X is defined as $\varepsilon_X = n.|p_X - p_X^*|$.

The bit-error indicates how many bits p_X is away from p_X^* for the precision level corresponding to n. Note that $1/n$ is the smallest non-zero SN that can be represented exactly. SNs can only rational represent numbers of the form n_1/n exactly; irrational numbers are approximated by the closest rational number. The exact value p_X^* is often known from the context. For instance, the output of a stochastic multiplier with inputs X and Y has the exact value $p_Z^* = p_X \times p_Y$, which serves to measure the output error ε_Z.

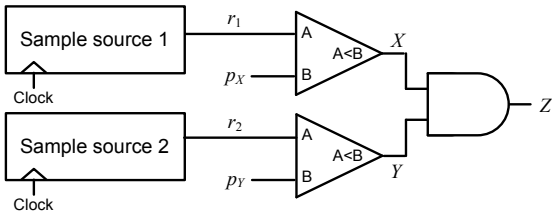

Figure 4. SC multiplication viewed as a Monte Carlo problem.

For example, let X = 0111 0100 and p_X^* = 6/8. Because p_X = 4/8, we have $\varepsilon_X = 8|4/8 - 6/8| = 2$. This error can be reduced to zero by changing two 0s of X to 1s, making p_X = 6/8. For the same X and p_X^* = 7/16, the error is $\varepsilon_X = 8|4/8 - 7/16| = 1/2$, implying that bit-flipping alone cannot eliminate the error. It can only be fixed by increasing n to 16; e.g., by extending X to 0111 0100 0101 0100.

A key insight of our approach is establishing a link between SC and Monte Carlo (MC). To illustrate this, we interpret an SC circuit as a small MC problem. Consider the circuit C in Figure 4 which is a stochastic multiplier supplied by two inputs X and Y derived from SNGs like that of Figure 2. The MC formulation of C is the following: given p_X and p_Y, estimate $p_Z^* = p_X p_Y$ by applying independent uniform random samples to r_1 and r_2. The direct MC approach [8] to solving this problem is to generate n independent random samples at r_1 and r_2. It can be shown that the expected value of p_Z is indeed $p_Z^* = p_X p_Y$ and its variance is $p_Z^*(1 - p_Z^*)/n$. The variance reflects the random fluctuations around p_Z^*, and implies that p_Z converges toward p_Z^* at the rate $O(1/\sqrt{n})$.

Figure 5a depicts a set of 64 purely random samples applied to the circuit C of Figure 4. The dashed blue lines indicate p_X and p_Y, and enclose a rectangle B of area $p_Z^* = p_X \times p_Y$. The samples in B produce 1 at the output of C; other samples produce 0. We want to spread the samples evenly over the plane, so that the number of samples in B is proportional to its area. In fact, this property is desired for every possible rectangle with a corner at the origin. Clearly, the samples of Figure 5a are non-uniformly dispersed even though they come from a uniform distribution; such a sample set is said to have *high discrepancy*.

The discrepancy of a sample set measures the uniformity of the distribution of samples, and is directly related to the sample set's accuracy [10]. In general, the sample space has d dimensions, as opposed to the two dimensions in Figure 5a. To measure the discrepancy, we want to check whether, for every d-dimensional hyper-rectangle B with a corner at the origin, the number of points inside B is proportional to the d-dimensional volume of B.

Definition 2: [10] Let S be a set of n samples in $[0,1]^d$, and let $\mathbb{B}$ be the set of d-dimensional hyper-rectangles with a corner at the origin. The *discrepancy* of S is

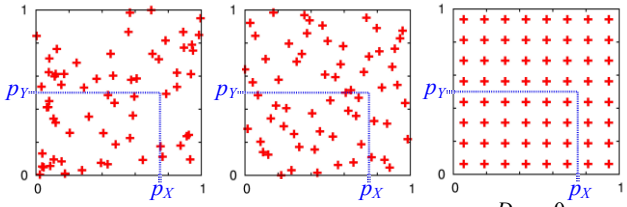

Figure 5. MC formulation of SC multiplication using (a) pure random, (b) pseudo-random, and (c) low-discrepancy samples.

$$D_S = \max_{B \in \mathbb{B}} \left| \frac{n_B}{n} - volume(B) \right|$$

where $volume(B)$ is the d-dimensional volume of B, and n_B is the number of sample point lying in B.

Figure 5a, shows a two-dimensional sample set S_1 with 64 sample points. The blue rectangle B is a member of the set $\mathbb{B}$ of possible rectangles whose area is $volume(B)$. In practice, the sizes of $\mathbb{B}$ and its members are limited by some specified precision. For example, if the maximum precision is $\log_2 n^*$, then $\mathbb{B}$ only includes sub-spaces with sides of the form k^*/n^*. With $n^* = 8$ for S_1, Definition 2 gives $D_{S_1} = 0.17$. This translates to the following statement: by applying sample set S_1 to the circuit C of Figure 4, we expect a maximum absolute error of 0.17 or, equivalently, a maximum bit-error of 11. Figure 5b illustrates a pseudo-random sample set S_2 generated by LFSR m-sequences with discrepancy $D_{S_2} = 0.02$, which is much better than S_1. Finally, the sample set S_3 of Figure 5c is an ideal sample set with discrepancy $D_{S_3} = 0$, implying that, when used with the circuit C, no errors will be seen. These samples are generated by selecting evenly spread points. They are deterministic, but contrary to intuition, they give better accuracy in SC than random or pseudo-random sample sets.

III. Progressive Precision

As noted, there is a strong connection between PP in SC and LD sequences in QMC. We examine this connection by comparing the PP properties of LD sequences with those of random and pseudo-random sequences. Many types of LD sequences are discussed in the QMC literature [10]. Most methods of generating them are software-based; hardware implementations are rare and unsuited to SC [4].

The SNG of Figure 2 produces a variable-length bit-stream X whose value p_X fluctuates initially, but eventually converges near p_X^*. Precision and accuracy tend to improve over time, but random fluctuations in its bit-stream can cause X to temporarily diverge from p_X^*. If the divergence is minimized, then "good" PP behavior is obtained. For example, X = 0101 0101 0101 0101, with $n^* = 16$ has $p_X = p_X^* = 1/2$ and good PP. As X is generated, we see the subsequences 0, 01, 010, 0101, 01010, ... All the even-length subsequences represent 1/2 exactly. At the other extreme, the initial subsequences of Y = 1111 1111 0000 0000 give very poor approximations to p_Y^* = 1/2, so this bit-stream has poor PP.

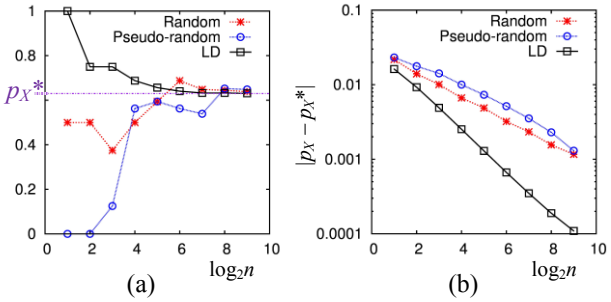

Figure 6. SN generation illustrating PP; (a) numerical value p_X for $p_X^* = 0.63$, and (b) average absolute error $|p_X - p_X^*|$ for all p_X^*s.

Figure 6a illustrates the progressive precision behavior of three different types of stochastic numbers as they converge toward the target p_X^*. The SNs generated from random and pseudo-random (LFSR-based) sample sets converge slowly and fluctuate a lot before settling at or near the target value. These are examples of "bad" PP. The SN generated using low-discrepancy sequences, on the other hand, quickly and monotonically converges to the target value. To further illustrate the convergence behavior of the different SNs, we plot the average error of several SNs as their length increases; see Figure 6b. The error of the LD-based SNs drops much faster than the others. This implies that the initial subsequences of the LD case provide a good early estimate of the target value, implying good PP.

Definition 3: An n-bit SN is k-PP if the bit-error of its initial sub-sequence of length 2^i is at most k for all i.

For example, let X = 0111 1111 1111 0000 and let the exact value $p_X^* = 10/16$. The initial sub-sequences of length 2, 4, 8, and 16 are 01, 0111, 0111 1111, and 0111 1111 1111 0000, respectively. The corresponding bit-errors of the sub-sequences are, from Definition 1: 0.25, 0.5, 2 and 1, respectively. This means that X is 2-PP because the maximum bit-error of its initial sub-sequences is 2. The number Y = 0111 1101 1111 0000, on the other hand, is 1-PP because the bit-errors of the initial subsequences have values 0.25, 0.5, 1 and 0, respectively. We say Y has better PP than X because the errors of its initial sub-sequences are lower.

We propose to use Halton sequences for SC [10]. These are among the less complicated LD sequences, and they show good performance for problems like those posed by stochastic circuits. Figure 7 shows the structure of our Halton sequence generator. It consists of a binary-coded base-b counter, where b is a prime number. The order of the output digits is reversed and the resulting number is converted to a binary number so that it fits into the SNG framework of Figure 2. For example, for b = 3, the (binary-coded ternary) counter generates the sequence:

000, 001, 002, 010, 011, …, 220, 221, 222

Then, the order of the digits is reversed thus:

000, 100, 200, 010, 110, …, 022, 122, 222

(which requires no logic) and the reordered digits are converted to equivalent binary numbers. First, each base-3

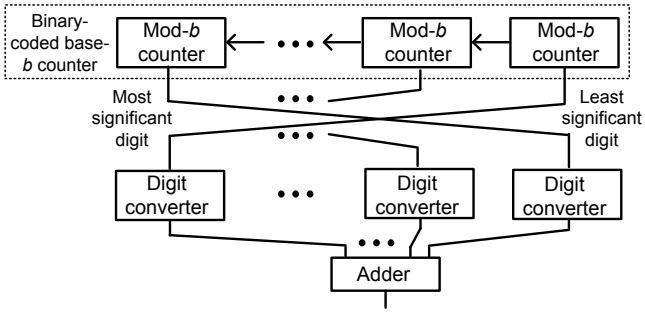

Figure 7. Design of the proposed Halton LD sequence generator.

digit is converted using a digit converter, then the results are summed to generate the binary sequence

00000, 01011, 10101, …, 10100, 11111

When $b = 2$, Figure 7a reduces to a simple binary counter. For d inputs, we need d copies of the circuit of Figure 7a with different prime bases. The 2-input circuit of Figure 4, requires two Halton sequence generators having $b = 2$ and 3. The output Z then becomes a 3-PP SN, in contrast with the 20-PP SNs produced by pseudo-random sequences.

IV. Case Studies

Next, we analyze three applications of SC and use PP to speed them up. Each has a computation step followed by a decision. With appropriate PP, the decision can be made early, thus reducing overall run-time. Higher-precision computations therefore happen only when needed.

First, consider a 2-input multiplier circuit implementing

$$F(p_X, p_Y) = \begin{cases} 1 \text{ if } p_Z^* = p_X \times p_Y \geq 0.5 \\ 0 \quad \text{otherwise} \end{cases}$$

For 8-bit precision, a non-stochastic design needs an 8-bit multiplier and a comparator to implement the decision $p_Z^* \geq 0.5$. It also computes F in one clock cycle. An SC version needs a circuit like that of Figure 4 with an AND-gate multiplier and a decision circuit. Using m-sequences, the SC design would need 1,024 clock cycles to produce a satisfactory result with an error rate of less than 1%.

Applying LD sequences to an SC multiplier produces an output Z with 3-PP, so the initial sub-sequences of Z are at most 3 bits away from the target result. Suppose the initial sub-sequence of length 16 (<< 1,024) is Z_{16} = 0010 0001 0000 1000, denoting $p_{Z_{16}}$ = 3/16. Because Z is 3-PP, the exact target value would be $p_Z^* < 7/16$, yielding $F = 0$. So the computation of F can stop after 16 clock cycles in this case, along with many similar cases. It turns out that for uniformly distributed values of p_X and p_Y, the average run-time per input of the design exploiting PP is only 59 cycles.

We synthesized the proposed design and equivalent stochastic and non-stochastic versions of F using SIS and its generic 0.35μm cell library; see Table I. The area reported does not include conversion units such as SNGs and counters, since their cost can be shared among parallel units. The delay of these units *is* considered in the run-time.

Table I. Comparison of the Three Case Studies.

Design		Area (μm^2)	Ave. run-time (ns)	Area×delay (μm^2 × ps)
Multiplication circuit	Non-stochastic	12,214	48	587
	SC without PP	35	6,871	242
	SC with PP	35	1,375	48
4-input neuron	Non-stochastic	54,727	86	4,714
	SC without PP	273	27,498	7,501
	SC with PP	273	12,598	3,437
Edge detector	Non-stochastic	6,943	26	181
	SC without PP	200	3,855	771
	SC with PP	200	332	66

The average run-time is calculated by multiplying the minimum clock period of the circuit by the average number of cycles needed. The area-delay product is also reported, which shows that our new design is much more efficient than the conventional stochastic and non-stochastic designs. Table I also compares different designs of a stochastic neuron [3] and the edge-detection circuit implementing (1).

V. Conclusions

This paper has addressed the central problem of stochastic computing, viz., accuracy limitations that lead to long run-times. We showed that better accuracy can often be achieved using deterministic number sources instead of the usual pseudo-random ones. We devised a quantitative measure for progressive precision, along with practical ways to exploit it. We also identified useful connections between SC and QMC methods. In particular, we showed that low-discrepancy sequences provide very good PP, leading to fast and accurate stochastic circuits. Finally, we used the proposed ideas to obtain stochastic circuits with PP that significantly outperform conventional stochastic and non-stochastic designs in representative applications.

Acknowledgement

This work was supported by Grant CCF-1318091 from the U.S. National Science Foundation.

References

[1] Alaghi, A. and Hayes, J.P., "Survey of stochastic computing," *ACM Trans. Embedded Computing Sys.*, 12, pp. 92:1-92:19, 2013.

[2] Alaghi, A. Li, C. and Hayes, J.P., "Stochastic circuits for real-time image-processing applications," *Proc. DAC*, paper 136, 2013.

[3] Brown, B.D. and Card, H.C., "Stochastic neural computation I: computational elements," *IEEE Trans. Comp.*, 50, pp.891-905, 2001.

[4] Dalal, I.L.et al."Low discrepancy sequences for MC simulations on reconfigurable platforms," *Proc. ASAP*, pp.108-113, 2008.

[5] Gaines, B.R., "Stochastic computing systems," *Advances in Information Systems Science*, 2, pp. 37-172, 1969.

[6] Golomb, S.W. and Gong, G., *Signal Design for Good Correlation,* New York: Cambridge Univ. Press, 2004.

[7] Gross, W.J., Gaudet, V.C. and Milner, A., "Stochastic implementation of LDPC decoders," *Proc. Asilomar Conf.*, pp. 713-717, 2005.

[8] Hammersley, J.M. and Handscomb, D.C., *Monte Carlo Methods*. London: Methuen, 1964.

[9] Jeavons, P. et al., "Generating binary sequences for stochastic computing," *IEEE Trans. Info. Theory*, 40, pp. 716-720, 1994.

[10] Niederreiter, H., *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM Series in App. Math., 32, Philadelphia, 1992.

[11] Qian, W. et al.,"An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Computers*, 60, pp. 93-105, 2011.