In-Memory Flow-Based Stochastic Computing on Memristor Crossbars using Bit-Vector Stochastic Streams

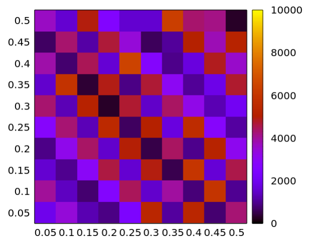
Sunny Raj   Dwaipayan Chakraborty   Sumit Kumar Jha[1]

***Abstract*— Nanoscale memristor crossbars provide a natural fabric for in-memory computing and have recently been shown to efficiently perform exact logical operations by exploiting the flow of current through crossbar interconnects. In this paper, we extend the flow-based crossbar computing approach to approximate stochastic computing. *First*, we show that the natural flow of current through probabilistically-switching memristive nano-switches in crossbars can be used to perform approximate stochastic computing. *Second*, we demonstrate that optimizing the approximate stochastic computations in terms of the number of required random bits leads to stochastic computing using *bit-vector stochastic streams* of varying bit-widths – a hybrid of the traditional full-width bit-vector computing approach and the traditional bit-stream stochastic computing methodology. This hybrid approach based on bit-vector stochastic streams of different bit-widths can be efficiently implemented using an in-memory nanoscale memristive crossbar computing framework.**

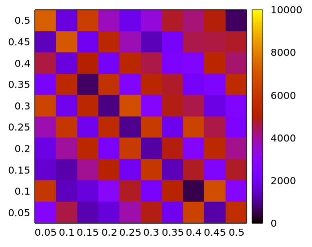
I. Introduction

Stochastic computing [1], [2] has traditionally not enjoyed the same success as the use of stochastic methods in electronic communications despite the common early origins of the two areas [3]. A primary argument against the adoption of stochastic computing has been the length of the stochastic stream required to perform computations with desirable accuracy. Even though each operation in stochastic computation is simple and energetically attractive, the total energy spent in performing this simple energy-efficient operation for a long stochastic stream turns out to be energetically unattractive. Another challenge in building stochastic computing systems has been the difficulty of generating long stochastic streams of random bits in a natural and efficient manner. Hence, an important design objective in stochastic computing is to minimize the number of random bits required for a computation.

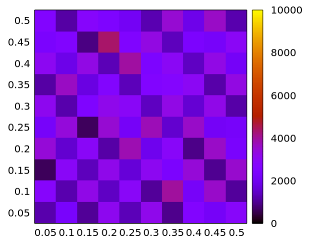
Recent surveys and applications to soft error-tolerant applications [4], [5] are re-visiting the foundational results in stochastic computing [6], [7], [1] obtained by Gaines and John von Neumann. This renewed interest in nanoscale stochastic computing is driven by the end of Dennard scaling [8] and the rise of machine learning workloads requiring only approximate probabilistic correctness guarantees. The discovery of nanoscale memristors [9], [10], [11] as non-volatile memory elements with stochastic switching capability [12] provides us a unique opportunity to revisit the design



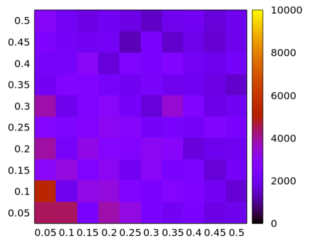
(a) Bit stream



(b) Bit-width = 2



(c) Bit-width = 4



(d) Bit-width = 8

Fig. 1: Number of samples needed by bit-vector stochastic streams of different bit-widths for approximate addition of two numbers. The horizontal and vertical axes denote the two numbers being added and the lighter colors indicate more number of samples than the darker colors. Bit-vector stochastic streams with sufficiently high bit-widths need fewer samples for stochastic computing.

of stochastic computing systems. In this paper, we make two new contributions:

(i) We present a *flow-based in-memory stochastic computing architecture* that exploits the flow of current through probabilistically-switching memristive nano-switches in high-density crossbars [13], [14], [15] to perform stochastic computations. Such an approach has not been studied earlier due to three reasons: (i) low switching speeds of non-volatile memories, (ii) high deterministic switching energies, and (iii) sneak paths in memory crossbars. The first problem has been alleviated by device engineers. We turn the two remaining "problems" – probabilistic switching at low energies and sneak paths in crossbars – into our design features.

(ii) We introduce a new stochastic computing methodology using *bit-vector stochastic streams* of varying bit-widths (see Figure 9) instead of traditional stochastic streams composed only of individual bits. We show that the use of higher bit-width bit-vector approximations in stochastic computing requires fewer number of random bits than that required by traditional stochastic computing with individual bitstreams.

*This work is supported by the National Science Foundation and the Air Force Office of Scientific Research Young Investigator Award to Sumit Jha.

[1]All authors are with the Computer Science Department, University of Central Florida, 4000 Central Florida Blvd, Orlando, FL 32816-2362. {sraj,dchakra,jha}@eecs.ucf.edu

II. Related Work

As early as 1970, SABUMA or the Safe Bundle Machine [16] implemented principles of stochastic computing and bundled transmission of ternary data on multiple wires to achieve parallel fail-soft computations. Because of the immense success of device physicists in ensuring the longevity of Dennard scaling [8], the science [1] and engineering [6], [7] of stochastic computing was largely forgotten in the subsequent decades. Our work is related to the SABUMA as we also use parallel (nano-)wires for communication and stochastic encodings to facilitate our stochastic computations. Unlike the SABUMA, we use naturally occurring sneak paths in nanocrossbars of memristors to perform approximate stochastic computations.

A flurry of recent activity in stochastic computing has focussed on "soft" applications, such as image processing [17], [18], [19]. Our work is related to these efforts as we only seek to perform approximate computations in parallel. However, we remain firmly tied to the nanocrossbar fabric that is likely to empower future emerging memories and are interested in devising general-purpose in-memory computing architectures that exploit stochasticity. Unlike these approaches that focus on stochastic streams of individual bits, we employ bit-vector stochastic streams of varying bit-widths in our stochastic computations.

III. In-Memory Stochastic Computing

A. Problems and Opportunities

There are two hurdles in the exploitation of high-density nanoscale memristive crossbars as in-memory computing devices: probabilistic switching at sub-threshold voltages and sneak paths in high-density crossbars.

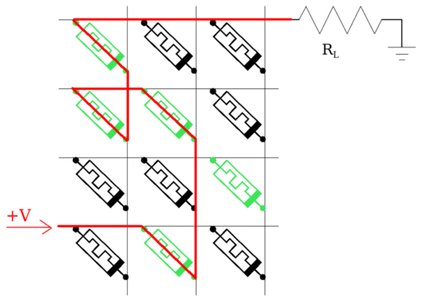


Fig. 2: An illustration [20] of a sneak path through a crossbar of nanoscale memristors. Turned-off memristors are shown in black. The red curve shows how current flows through the turned on (green) memristors from the source to the ground.

1) Probabilistic Switching: The switching of nanoscale memristors at low voltages is probabilistic. Hence, either the storage of data onto the nanoscale device is unreliable or there is a need to use higher programming voltages and larger switching energies. Gaba et al. have shown that stochastic memristive devices can be used for computing [12]. The probability of a memristor being turned-on can be controlled by varying the width of the programming pulse. One set of experimental data and the exact form of the probabilistic switching curves can be obtained from [12]. There is clear experimental evidence that the switching probability of a variety of memristors can be effectively controlled. We will exploit this ability to switch nanoscale memristors with desired probabilities in our flow-based stochastic crossbar computing designs.

2) Sneak Paths: The reading-back of data stored in nanoscale memristor crossbars is subject to the so-called "sneak path" problem. In essence, instead of the current flowing through a turned-off high-resistance memristor representing the bit '0', the current sneaks around alternative paths of turned-on low-resistance memristors around this turned-off memristor (see Figure 2). Hence, an observer measuring this current believes that the memristor being read must be turned-on representing the bit '1'.

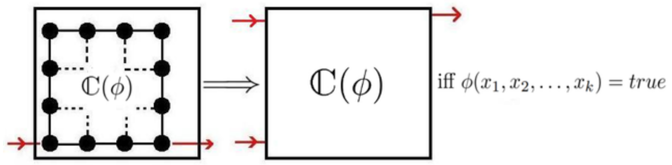


Fig. 3: The current flow manages to reach the top row of the crossbar if and only if there exists a sneak path between the bottom and the top rows. The crossbar $\mathbb{C}(\phi)$ is carefully designed so that there is a sneak path between the top and the bottom row if and only if the Boolean formula ϕ evaluates to true.

We have devised efficient techniques [20], [21], [22], [23] to encode Boolean formulas into memristive crossbars. Sneak paths, usually perceived as a design flaw, are leveraged to perform Boolean computations using flow of current through nanoswitches and nanowires in memristor crossbars. Given a certain Boolean formula ϕ and the crossbar $\mathcal{C}(\phi)$ designed to compute this formula, the relation is expressed as $\mathcal{C} \models \phi$. The shorthand notation indicates that the current flow introduced at the bottom row of the crossbar reaches the top row if and only if the Boolean formula $\phi(x_1, x_2, \cdots, x_k)$ evaluates to true for the inputs $x_1, x_2, \cdots, x_k$. This general idea is depicted in Fig. 3. The approach of using sneak paths to perform computation [21] has been used to implement a compact, fast, and energy-efficient one-bit adder, as shown in Fig. 4.

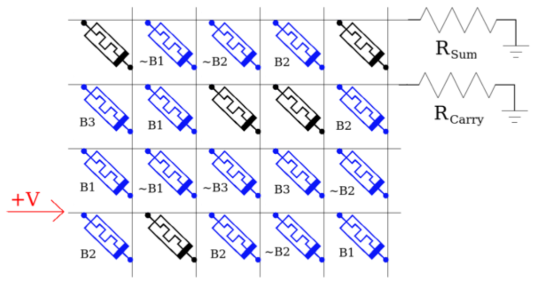


Fig. 4: The design [21] of a one-bit adder using sneak paths.

Our stochastic computing approach relies on these efficient crossbar design approaches for computing Boolean formula.

B. Motivating Example

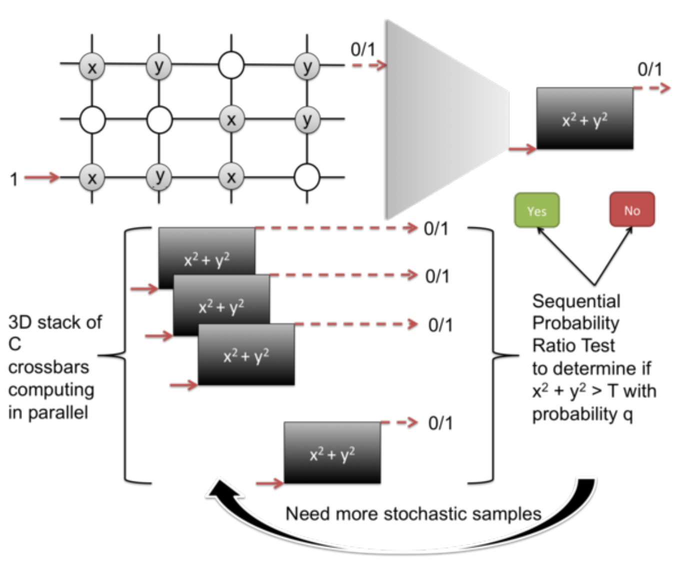

x
y
y
x
y
x
x
y
x
1
0/1
0/1
x² + y²
Yes
No
0/1
0/1
0/1
0/1
x² + y²
x² + y²
x² + y²
x² + y²
3D stack of C crossbars computing in parallel
Sequential Probability Ratio Test to determine if x² + y² > T with probability q
Need more stochastic samples

Fig. 5: A motivating example describing the stochastic approximate parallel computation of the expressions $x^2+y^2 > T$ using three-dimensional stacks of memristor crossbars.

Consider a simple program fragment that decides if the squared distance of a point (x, y) from an origin exceeds a fixed threshold T i.e. it computes the expression $x^2+y^2 > T$ where T is a constant threshold. Fig. 5 (top left) shows how a nanoscale memristor crossbar with 3 rows and 4 columns can be used to compute the expression x^2+y^2 approximately. The 3×4 crossbar contains three types of memristors: (i) memristors that are always turned off, (ii) memristors whose probability of being turned on is given by the input variable x, and (iii) memristors whose probability of being turned on is given by the input variable y. Each memristor is configured randomly by turning on memristors according to the input probabilities of the variables x and y. This can be achieved by various methods such as subjecting each memristor to a sub-threshold voltage pulse width of adequate duration (see Sec. III-A.1 and [12]).

Then, a small voltage is provided to the lowest nanowire of the memristor crossbar. The output current of the nanoscale memristor crossbar is observed on the topmost nanowire of the 3×4 crossbar. If a significant current is observed on the topmost nanowire, the crossbar is said to have produced the logical output 1. If no significant current is observed on the topmost nanowire, the logical output is determined to be 0. Each crossbar thus produces a single Boolean value - 0 or 1, and contributes to the unary stochastic encoding of the output.

Figure 6 demonstrates how C such nanoscale crossbars can produce C bits in the stochastic bit stream of the output corresponding to x^2+y^2 in parallel. We emphasize that the crossbar computation is both approximate and probabilistic. As we can see from the two plots, the quality of the approximate computation improves as the number of parallel computing elements being employed is increased.

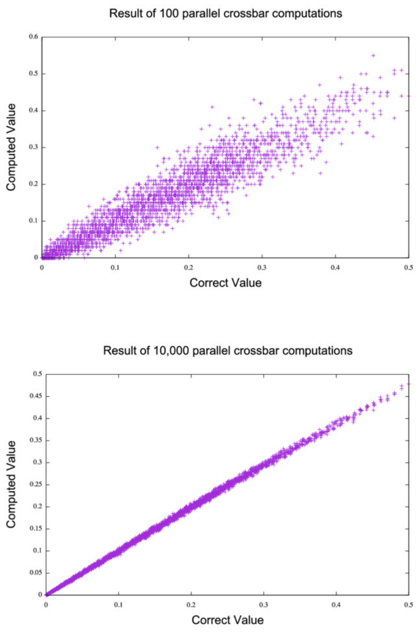


Fig. 6: Simulation results showing that the nanoscale crossbar circuit in Fig. 5 correctly computes an approximation to the expression x^2+y^2 in parallel.

C. Approximate Stochastic Crossbar Computing

Our overall approach to approximate stochastic computing using memristor crossbars is composed of three steps.

First, we compute the Boolean formula corresponding to a desired arithmetic computation. The equivalence between Boolean formula and their arithmetic probabilistic implementation is well established [24], [25], [26], [27]. It has recently been re-visited in [28] together with an elementary proof based on mathematical induction.

Consider the following multivariate polynomial in t variables $x_1, x_2, \ldots, x_t$:

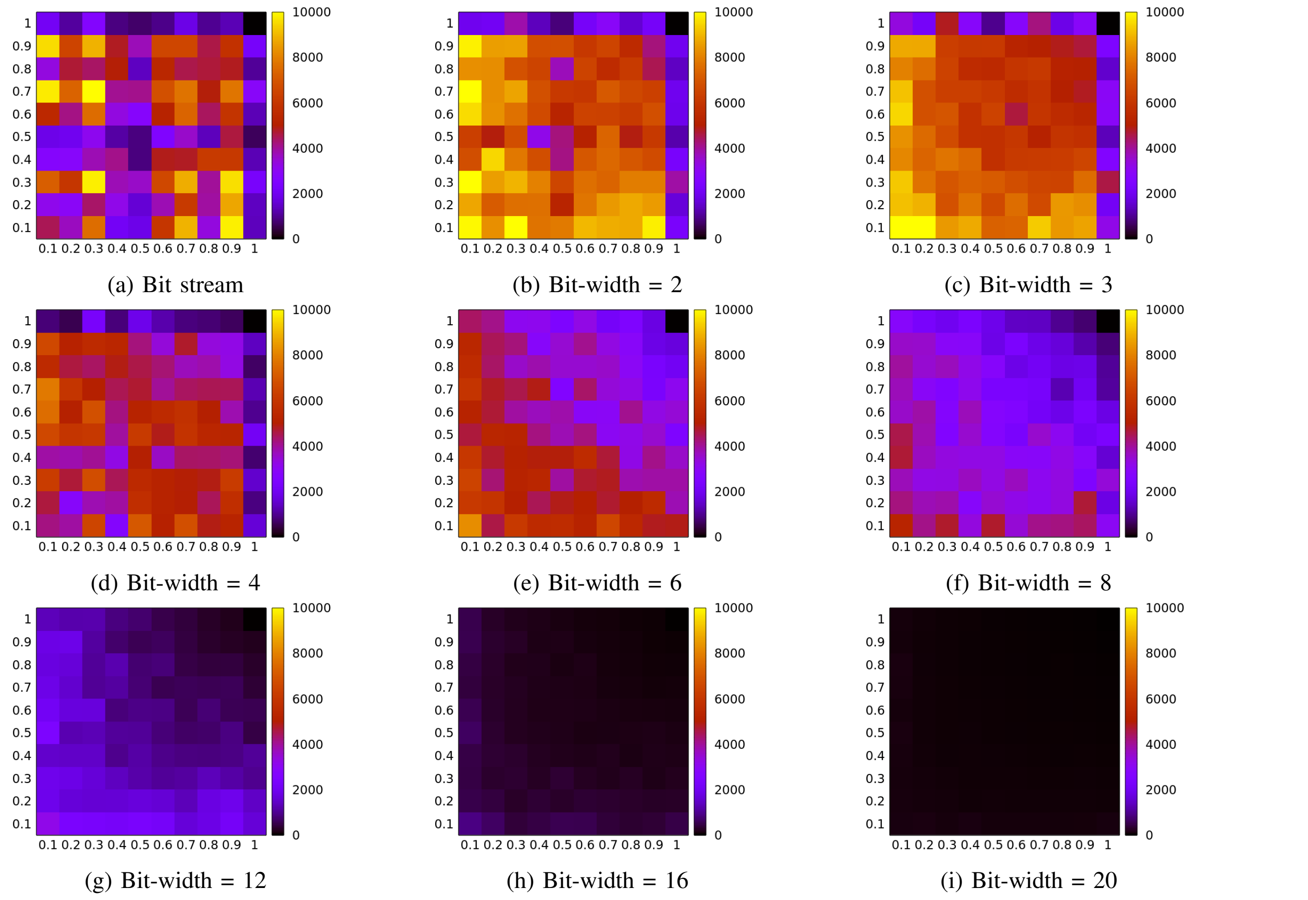
$$f(x_1, x_2, \ldots, x_t) = \sum_{i_1=0, i_2=0, \ldots i_t=0}^{i_1=k, i_2=k, \ldots i_t=k} \alpha_{i_1, i_2, \ldots, i_t} x_1^{i_1} x_2^{i_2} \ldots x_t^{i_t}$$

We assume that $\alpha_{i_1, i_2, \ldots, i_t} \in [0, 1]$, $x_j^i \in [0, 1]$, and $f(x_1, x_2, \ldots, x_t) \in [0, 1]$. We define the Boolean formula $\mathcal{B}(f)$ corresponding approximately to the multivariate polynomial $f(x_1, x_2, \ldots, x_t)$ as follows:

$$\mathcal{B}(f) = \bigvee_{i_1, i_2, \ldots, i_t} A_{i_1, i_2, \ldots, i_t} \left(X_1^1 X_1^2 \ldots X_1^{i_1}\right) \ldots \left(X_t^1 X_t^2 \ldots X_t^{i_t}\right)$$
$$\vee \left(\bigvee_{i_1 \ldots, j_t > i_t} \left(A_{i_1 \ldots, i_t} A_{j_1 \ldots, j_t} \left(X_1^1 \ldots X_1^{i_1}\right) \left(X_1^1 \ldots X_1^{j_1}\right) \ldots \right) \right)$$

Here, the As and the Xs are Boolean variables. The probability $P(X_i^j)$ of the Boolean variable X_i^j having the logical value 1 is x_i. Similarly, the probability $P(A_{i_1, i_2, \ldots, i_t})$ of the Boolean variable $A_{i_1, i_2, \ldots, i_t}$ having the logical value 1 is $\alpha_{i_1, i_2, \ldots, i_t}$. Further, the random variables X_i^j and $A_{i_1, i_2, \ldots, i_t}$ are all independent of one another.

For example, a Boolean formula approximately corresponding to the polynomial $x^2 + y^2$ is $X_1 X_2 + Y_1 Y_2 +$


(a) Bit stream (b) Bit-width = 2 (c) Bit-width = 3

(d) Bit-width = 4 (e) Bit-width = 6 (f) Bit-width = 8

(g) Bit-width = 12 (h) Bit-width = 16 (i) Bit-width = 20

Fig. 7: Number of samples needed by bit-vector stochastic streams of different bit-widths for multiplication of two numbers. Bit-vector stochastic streams with sufficiently large bit-widths need fewer samples.

$X_1X_2Y_1Y_2$. Similarly, a Boolean formula approximately corresponding to the polynomial $x^2 + y^2 + z^2$ is $X_1X_2 + Y_1Y_2 + Z_1Z_2 + X_1X_2Y_1Y_2 + X_1X_2Z_1Z_2 + Y_1Y_2Z_1Z_2$.

Second, the computed Boolean formula is mapped onto the design of a nanoscale memristor crossbar using either a BDD-based approach [22] or a method [23], [21] based on searching the space of possible crossbar designs. Each memristor m in the crossbar $\mathbb{C}$ is mapped to a Boolean variable v or its negation $\neg v$. The Boolean variable v is also associated with a probabilistic variable p in the target arithmetic computation. In our design, the memristor m is turned on with probability p if the memristor m is labeled with variable v; it is turned on with probability $1 - p$ if the memristor m is labeled with the literal $\neg v$.

Continuing with our examples in the previous step, the memristor crossbar corresponding approximately to the expression $x^2 + y^2$ is shown in Figure. 5. Similarly, the memristor crossbar corresponding to the expression $x^2 + y^2 + z^2$ is shown in Fig. 8. It should be noted that the crossbars have been optimized for area and include additional Boolean terms that correspond to very low probability expressions; hence, they do not significantly affect our approximate computations.

Finally, multiple copies of the crossbar designed in the previous step are executed either sequentially or in parallel. Each crossbar performs a stochastic computation and produces a 0 or 1. The fraction of 1s in the stochastic output stream is the output produced by our approximate stochastic computation method.

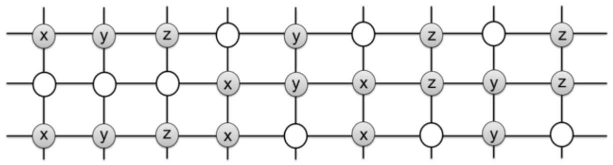
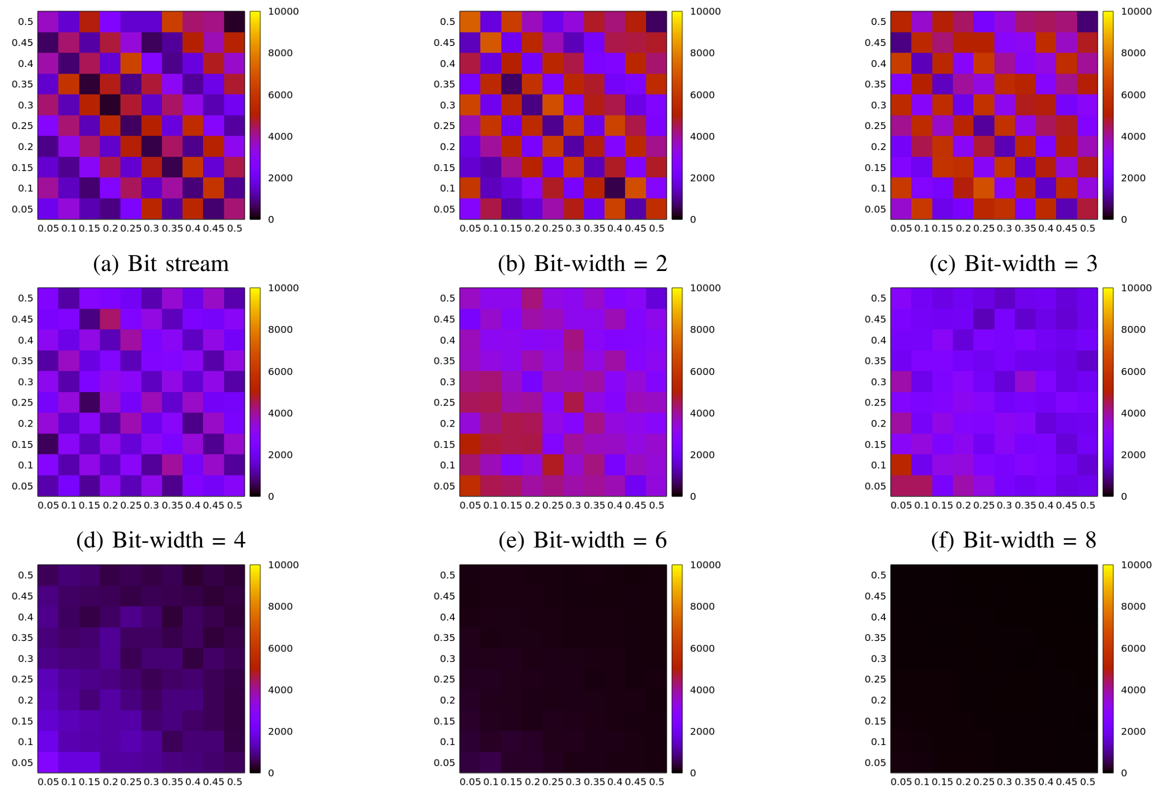


Fig. 8: Crossbar implementing the approximate stochastic computation of $x^2 + y^2 + z^2$. The unlabeled memristors are always turned off while the memristors labelled with variables are probabilistically turned on. The probability of a labeled memristor being turned on is given by the variable labeling the memristor.

For example, Fig. 6 shows two plots comparing the correct value of the expression $x^2 + y^2$ with the value of the expression computed using the crossbar in Figure 5. The average of 100 crossbar computations has a wider variance than the average of 10,000 crossbar computations. In practice, the number of crossbar computations can be chosen based on the error tolerance of the target application.



(a) Bit stream (b) Bit-width = 2 (c) Bit-width = 3

(d) Bit-width = 4 (e) Bit-width = 6 (f) Bit-width = 8

(g) Bit-width = 12 (h) Bit-width = 16 (i) Bit-width = 20

Fig. 9: Number of samples needed by stochastic bit streams of different bit-widths for addition of two numbers. Bit-vector stochastic streams with sufficiently large bit-widths need fewer samples.

IV. Bit-Vector Stochastic Streams

Given two numbers $m_1, m_2 \in (0, 1)$, a standard approach to stochastic computing uses a unipolar encoding of the inputs into stochastic bit streams and employs an ordinary "AND" gate to implement the multiplication. However, this simplicity comes at tremendous cost in terms of the number of *i.i.d.* samples that must be drawn to obtain a good approximation of the output. See Table I and Figure 7.

TABLE I: Multiplication using stochastic bit-vector streams.

Bit-width	Average #samples	Maximum #samples	Average #bits	Maximum #bits
1	4298.83	9973	4298.83	9973
2	6206.72	11146	12413.40	22292
3	6055.87	11556	18167.60	34668
4	4164.54	7806	16658.10	31224
5	4694.03	8242	23470.10	41210
6	4114.02	8210	24684.10	49260
7	3606.08	7112	25242.50	49784
8	2839.92	5205	22719.30	41640
9	2428.43	4292	21855.90	38628
10	2027.71	3892	20277.10	38920
11	1517.74	3353	16695.20	36883
12	1136.41	3171	13636.90	38052
13	772.74	2144	10045.70	27872
14	527.32	1593	7382.46	22302
15	367.15	1113	5507.24	16695
16	221.30	853	3540.83	13648
17	139.36	409	2369.12	6953
18	88.03	245	1584.56	4410
19	56.32	158	1070.02	3002
20	37.32	100	746.27	2000

Instead of using a long stochastic stream of individual bits or using a single deterministic fixed bit-width representation for computation, we explore a middle path: *stochastic streams of fixed-width bit-vectors*. Given two numbers $m_1, m_2 \in (0, 1)$ and a fixed width t, our approach draws random *i.i.d.* numbers b_1, b_2 from independent binomial distributions with probability of success as m_1, m_2 respectively and $T = 2^t - 1$ as the number of trials.

$$b_1 \sim Binom(m_1, T), \quad b_2 \sim Binom(m_2, T), \quad s = (b_1 \times b_2)$$

The expected value of b_1 is $m_1 T$ and the expected value of b_2 is $m_2 T$, the expected value of s is $(m_1 m_2)T$. A natural question follows: Of all possible bit-widths, *is there a bit-width that minimizes the total number of random bits used by the stochastic bit-vector stream*?

Table I shows how the number of *i.i.d.* samples changes as the bit-width of the bit-vector stochastic stream is increased while maintaining a constant approximation error for multiplication. Table II shows the same information for addition. The row corresponding to the bit-width of 16 is highlighted in both the tables as it needs fewer average number of random bits than the traditional approach based on individual stochastic bits. The results of our experiments are illustrated in Figure 7 for multiplication and Figure 9 for addition. Bit-vector stochastic streams with modest bit-widths (16-20) lead to fewer samples and fewer number of random bits.

TABLE II: Addition using stochastic bit-vector streams.

Bit-width	Average #samples	Maximum #samples	Average #bits	Maximum #bits
1	2881.54	6394	2881.54	6394
2	3659.13	7206	7318.26	14412
3	3949.45	6778	11848.40	20334
4	2338.82	4324	9355.30	17296
5	3847.51	6539	19237.60	32695
6	3500.40	5614	21002.40	33684
7	3120.85	5633	21845.90	39431
8	2374.83	5314	18998.70	42512
9	2041.25	4110	18371.20	36990
10	1538.79	2926	15387.90	29260
11	1146.69	2295	12613.60	25245
12	772.96	2676	9275.53	32112
13	518.13	1423	6735.74	18499
14	316.41	1032	4429.69	14448
15	209.04	618	3135.52	9270
16	132.91	460	2126.54	7360
17	84.27	312	1432.66	5304
18	55.06	181	991.00	3258
19	35.36	134	671.81	2546
20	22.94	74	458.76	1480

V. Conclusions

Our goal in developing the proposed in-memory approximate stochastic computing architecture is to explore the design opportunities provided by novel nanoscale memristor crossbar fabrics and emerging approximate probabilistically-correct computing workloads. In this paper, we have presented

(i) a new flow-based in-memory computing architecture that employs nanoscale memristive crossbars to perform approximate stochastic computing, and

(ii) a new stochastic computing methodology that employs bit-vector stochastic streams of varying bit-widths instead of traditional stochastic streams composed of individuals bits.

Our work builds on earlier foundational results by Gaines [6] and Gaba et al. [12].

VI. Acknowledgment

The authors would like to thank the US Air Force for support provided through the AFOSR Young Investigator Award to Sumit Jha. The authors acknowledge support from the National Science Foundation Software & Hardware Foundations #1438989 and Exploiting Parallelism & Scalability #1422257 projects. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-16-1-0255.

References

[1] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[2] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, pp. 37–172.

[3] K. De Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro, "Computability by probabilistic machines," *Automata studies*, vol. 34, pp. 183–198, 1956.

[4] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, p. 92, 2013.

[5] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 59.

[6] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 149–156.

[7] ——, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, pp. 37–172.

[8] M. Bohr, "A 30 year retrospective on dennard's mosfet scaling paper," *Solid-State Circuits Society Newsletter, IEEE*, vol. 12, no. 1, pp. 11–13, 2007.

[9] G. Gandhi, V. Aggarwal, and L. O. Chua, "The detectors used in the first radios were memristors," in *Memristor Networks*. Springer, 2014, pp. 53–66.

[10] L. O. Chua, "Memristor-the missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[11] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[12] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–5878, 2013.

[13] S. Lee, J. Sohn, Z. Jiang, H.-Y. Chen, and H.-S. P. Wong, "(Invited) graphene plane electrode for low power 3d resistive random access memory," *ECS Transactions*, vol. 72, no. 4, pp. 159–164, 2016.

[14] P. E. Gaillardon, X. Tang, and G. De Micheli, "High-performance low-power near-Vt resistive memory-based FPGA," Jan. 28 2016, US Patent 20,160,028,396.

[15] Q. Liu and J. H. Zhang, "High density resistive random access memory (RRAM)," Apr. 5 2016, US Patent 9,305,974.

[16] D. Coombes, *SABUMA-Safe Bundle Machine*. Department of Computer Science, University of Illinois, 1970.

[17] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. IEEE, 2011, pp. 154–161.

[18] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 136.

[19] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 449–462, 2014.

[20] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016, US Patent US 9,319,047 B2.

[21] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1870–1873.

[22] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Design, Automation and Test in Europe (DATE), Proceedings of 2017 IEEE International Conference on*. IEEE, 2017, pp. 770–775.

[23] ——, "Design of compact memristive in-memory computing systems using model counting," in *Circuits and Systems (ISCAS), Proceedings of 2017 IEEE International Symposium on*. IEEE, 2017, p. in press.

[24] J. L. Shanks, "Computation of the fast walsh-fourier transform," *IEEE Transactions on Computers*, vol. 100, no. 5, pp. 457–459, 1969.

[25] K. V. Palem, L. N. Chakrapani, Z. M. Kedem, A. Lingamneni, and K. K. Muntimadugu, "Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. ACM, 2009, pp. 1–10.

[26] R. O'Donnell, *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[27] Y. Crama and P. L. Hammer, *Boolean models and methods in mathematics, computer science, and engineering*. Cambridge University Press, 2010.

[28] A. Alaghi and J. P. Hayes, "Strauss: Spectral transform use in stochastic circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1770–1783, 2015.