Stochastic Computing for Reliable Memristive In-Memory Computation

Mohsen Riahi Alam
University of Louisiana at Lafayette
Lafayette, Louisiana, USA
mohsen.riahi-alam@louisiana.edu

M. Hassan Najafi
University of Louisiana at Lafayette
Lafayette, Louisiana, USA
najafi@louisiana.edu

Nima TaheriNejad
Heidelberg University
Heidelberg, Germany
nima.taherinejad@ziti.uni-heidelberg.de

Mohsen Imani
University of California, Irvine
Irvine, California, USA
m.imani@uci.edu

Lu Peng
Tulane University
New Orleans, Louisiana, USA
lpeng3@tulane.edu

ABSTRACT

In-Memory Computing (IMC) is a promising computing paradigm to accelerate Big Data applications. It reduces the data movement between memory and processing units, and provides massive parallelism. Memristive technology is one of the promising technologies for IMC. This emerging technology, however, is still in evolution, facing practical challenges. Memristive memories are prone to soft-error while storing the data and during computations. The traditional binary encoding commonly used in memristive IMC is highly sensitive to soft-errors, which makes developing reliable memristive IMC more challenging. Stochastic Computing (SC) is a re-emerging computing paradigm that is highly robust against soft-errors as any bit flip leads to only a least significant bit error. In this work, we study SC as a solution to increase the reliability of memristive IMC. We investigate how and to what extent SC may address or improve the reliability issues of current memristive technology, and memristive IMC. We also evaluate the characteristics yielded by memristive stochastic IMC and compare them with those of the traditional reliability techniques.

CCS CONCEPTS

• **Hardware → Emerging technologies**; *Hardware reliability*.

KEYWORDS

stochastic computing, in-memory computing, memristors, processing in memory, reliability, soft errors.

ACM Reference Format:
Mohsen Riahi Alam, M. Hassan Najafi, Nima TaheriNejad, Mohsen Imani, and Lu Peng. 2023. Stochastic Computing for Reliable Memristive In-Memory Computation. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583781.3590307

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590307

1 INTRODUCTION

Running data-intensive applications on traditional Von Neumann computers with separate memory and processing units results in high energy consumption and slow processing speeds, primarily due to a large amount of data movement between memory and processing units. IMC is introduced to address this issue by processing data in memory where they are stored. Phase Change Memory (PCM), Magnetic Random Access Memory (MRAM), and Resistive Random Access Memory (ReRAM) are emerging non-volatile technologies and referred to under the umbrella term of memristors. Memristive technology is one of the promising technologies for IMC due to providing the most energy benefits, high-density, fast switching speed, and Complementary Metal-Oxide Semiconductor (CMOS)-compatibility. Memristive technology is not a fully mature technology yet and the Memristive devices have various reliability issues such as endurance, durability, and variability. They are prone to *soft-errors* in the logical states. This often causes significant errors in the computation when using the traditional binary representation. The inherent vulnerability of binary methods to fault and noise (e.g., to bit flips) poses a challenge to the reliability of memristive IMC. SC [1, 14] is a promising alternative to conventional binary offering simple execution of complex arithmetic operations and high tolerance to noise. SC uses a redundant representation that can inherently tolerate high rates of noise. Unlike positional binary representation, any bit-flip in a stochastic representation leads to only a least significant bit (LSB) error as all the bits have the same weight. This work combines the complementary advantages of SC and IMC to achieve reliable and fast computation in memristive memory. We study the reliability of memristive IMC for some essential arithmetic operations when executed in both binary and stochastic domains. We show that SC-based IMC is more reliable and a promising solution compared to the baseline in-memory binary techniques and to the state-o-the-art in-memory binary techniques with improved reliability.

2 FUNDAMENTALS

In-Memory Computation: As an umbrella term, IMC refers to any computing architecture where the data travels a smaller distance compared to its von-Neumann equivalent before processing. This includes processing data on the memory module (also called near-memory computing) and processing data on the memory chip. The

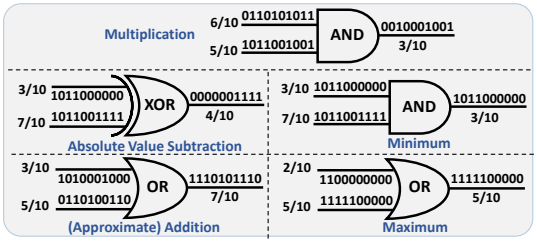

Figure 1: Basic SC Operations.

latter can be divided into processing data on the periphery of the memory array, or in-array computing. Processing data inside the array leads to minimum data travel and avoids unnecessary read and write processes. Thus it can maximize efficiency. This work focuses on *in-array processing*. In the realm of memristive IMC, stateful logics are the primary method for in-array computing. Stateful logics refer to logical operation in which the input and output values are represented as the state (resistance) of memristors [3]. This is in contrast to non-stateful logics, where inputs or outputs are represented as voltage or current levels.

Stochastic Computing: SC [1, 14] is an unconventional computing paradigm processing random bit-streams. Unlike conventional binary, all digits of a stochastic bit-stream have the same weight. The data value is determined by the probability of observing a '1' (0011010010 is representing 0.4). This redundant representation provides a high tolerance to noise, as any bit-flip can cause only an LSB error. The paradigm is also known for its ability for simple execution of complex arithmetic operations. For example, multiplication can be implemented using bit-wise logical AND [17]. Figure 1 shows some examples of basic SC operations.

SC in Memristive Memory: IMC has been employed to address the long-time challenges in cost-efficient design of SC systems [18]. In prior work, the intrinsic non-deterministic properties of memristors have been exploited to generate random bit-streams in memory. In [9], input data in an analog format were directly converted to random bit-streams by a stochastic group writing into the memristive memory. The stochastic nature of ReRAM devices was exploited in [6] to generate random bit-streams in memory. These in-memory methods eliminate the significant overhead of off-memory CMOS-based bit-stream generation [14]. A challenge with these methods is that the bit-stream generation is inaccurate. The bit-streams suffer from random fluctuations error [1]. Relatively long bit-streams (e.g., 1024 bits or more) must be generated for acceptable accuracy. SC with exploiting low-discrepancy (LD) bit-streams can be as accurate as conventional binary computing with shorter bit-streams. An in-memory method for generating LD deterministic bit-streams is proposed in [17]. LD bit-streams quickly and monotonically converge to the target value, achieving high accuracy with much shorter bit-streams and free of random fluctuations error.

Memristive IMC also provides massive bit-level parallelism. This is ideal for SC systems with many bit-level operations. By applying the same voltage along bit-lines/word-lines, we may induce a logical operation in all rows/columns of the memristive crossbar simultaneously. Further, crossbar arrays can be dynamically divided into multiple partitions to support simultaneous but different in-row (in-column) operations in the same row (column) [2, 5].

Soft-Error in Memristive Memory: Memristive technology is prone to soft-errors, i.e., unwanted non-deterministic transient changes in the logical states of memristors. These errors can be categorized into three categories:

Intrinsic errors depend mainly on the technological details of the device. For instance, variability in resistance (device-to-device and cycle-to-cycle), threshold voltage, and switching speed are known phenomena in memristors [20, 22]. These kinds of variability may lead to unwanted states or state changes in memristors. Another intrinsic effect is the retention [15, 20], which refers to a state drift and change in the absence of external stimuli and due to leakage. Some of these effects may be alleviated through good circuit design.

Operational errors mainly depend on the execution of certain functions and operations on memristors. For instance, even though the read-out can be designed such that it has a negligible effect on the (microscopic) state of a memristor [15], the accumulative effect of state drifts due to the read-outs can eventually lead to an unwanted state change [15]. Another example is the incomplete state change due to Material Implication (IMPLY) operation [4]. It is known that the state of the output memristor after an IMPLY operation in which both inputs are logical 0 (high resistance state) cannot reach to the low resistance state [4], even though the resistance may be low enough to be considered as logical 1. When a memristor that has not experienced a full state change (here, the output memristor holding the result of the IMPLY operation) is used in consecutive operations, it may or may not cause an error in one or more consecutive operations [20].

Random errors do not show any specific or recognizable pattern or relations with other known phenomena, conditions, inputs, or outputs. Such errors are observed [15] but not explained (at least not yet). A potential cause for such errors, as observed in many electronic devices, circuits, and systems, could be the cosmic radiation. Given that the source and nature of such errors are unknown, practically nothing can be done to remedy them.

3 RELIABILITY STUDY

In this work, we develop a Python-based cycle-accurate simulator for reliability evaluation. The implemented simulator is used to study the reliability of the data representation in binary and stochastic format and also the basic operations such as multiplication, subtraction, and maximum value function when executed in memristive memory in presence of different noise rates. We describe the data formats and the in-memory logic of each operation—consist of basic operations such as NOT and NOR—in this simulator. Our simulator supports both binary and SC IMC operations. We randomly inject noise with different rates of up to 20% into the data and logical operations to evaluate their tolerance to soft-errors.

3.1 Data Value

Binary-radix is a compact representation: it can represent 2^N distinct numbers using N bits. A stochastic representation, on the other hand, is a redundant representation: roughly 2^{2N} bits with *random* and 2^N bits with *deterministic* bit-streams are needed to represent 2^N distinct numbers [17]. This redundancy provides high tolerance to noise (i.e., bit flip) as all bits have the same weight. A single bit-flip results in a small error. Multiple bit-flips produce small and uniform deviations from the nominal value. Soft-errors

Table 1: MAE, MAX, and STD of 8-Bit Data Represented by Binary and Stochastic Format for Different Noise Rates.

SC-256	0%	0.1%	1%	2%	3%	5%	10%	15%	20%	**Binary-8**	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
MAE	0.00	0.39	0.78	1.33	1.73	2.73	5.26	7.82	10.3	MAE	0.00	0.10	0.95	1.96	2.90	4.67	9.06	12.9	16.7
MAX	0.00	0.39	1.17	2.34	3.12	5.07	10.1	15.2	20.3	MAX	0.00	51.5	75.0	81.2	87.5	87.8	88.2	96.8	97.2
STD	0.00	0.00	0.004	0.008	0.010	0.016	0.03	0.04	0.05	STD	0.00	0.019	0.05	0.08	0.09	0.11	0.15	0.18	0.19

in memristive devices can be inherently tolerated with such a representation without using the traditional reliability techniques such as Error Correcting Code (ECC) [10] or Triple Modular Redundancy (TMR) [11]. Table 1 reports the Mean Absolute Error (MAE), maximum absolute error (MAX), and standard deviation (STD) of representing 8-bit precision data using the binary and stochastic format when injecting noise (i.e., flipping bits) with different rates. The data are represented in binary format by 8 bits and in stochastic format by 256 LD bit-streams. We randomly inject noise into the bit positions and find the mean, maximum, and STD of the error in the represented values for 100,000 iterations. The MAE and MAX numbers are multiplied by 100 and shown in percent. As shown in Table 1, the binary format shows on average more than *70% higher MAX rate compared to the stochastic representation.* The maximum error in the binary representation quickly converges to an unacceptable absolute error rate of more than 51% for the 0.1% noise rate and more than 75% error for the 1% noise rate. On the other hand, the stochastic representation provides a much higher tolerance to noise by showing an MAE and a maximum error rate of 0.78% and 1.17% for 1% noise rate, and 2.73% and 5.07% for 5% noise rate, respectively. The maximum error in the stochastic representation increases proportionally with the noise rate, while in the binary format quickly converges to unacceptably high error rates of more than 50%.

3.2 Multiplication

For reliability evaluation of in-memory multiplication, we implemented the MAGIC-based fixed-point multiplication technique proposed in [7] and the SC-based in-memory multiplication technique developed in [17]. For fixed-point binary, we evaluate in-memory multiplication of two 8-bit precision data. For the SC multiplication, we generate and process LD bit-streams of 256 bits. Exhaustively testing multiplication of two 8-bit precision data on every possible pair of input values gives an MAE rate of 0.19% when processing LD bit-streams of 256 bits.

The binary multiplication method of [7] uses a partial product multiplication algorithm and reuses the memristor cells during execution. A two-input multiplication using this method takes a total of $13N^2 - 14N + 6$ cycles. SC multiplication involves performing bit-wise logical AND on independent bit-streams. To provide independence between bit-streams, we generate bit-streams with different LD distributions [17]. To execute AND operation, MAGIC NOR is performed on the inverted version of bit-streams of each operand. Independent of the data precision (i.e., bit-stream length), SC multiplication with this method is executed in only six cycles.

Table 2 shows the noise evaluation of the implemented binary and SC in-memory multiplication methods. We evaluate the reliability in three cases of injecting noise into 1) input data, 2) logical operations, and 3) both input data and logical operations. As it can be seen, although the SC approach cannot provide a 0.00% error rate in the ideal case of having no noise, it shows a significantly high tolerance to noise in all three cases. For example, for the case of having 1% noise rate in the logical operations, the SC method achieves an MAE of 0.84% and a MAX of 2.12%, whereas the binary method shows an MAE of 6.66% and MAX of more than 87%.

We further implemented the traditional TMR method [11] to improve the reliability of the *logical* operations in the in-memory binary multiplication. Since the voting logic is also vulnerable to noise, we implemented two cases of ideal (Ideal-TMR) and non-ideal (TMR) voting for the modified design. Table 3 reports the MAE and MAX rates. Compared to the numbers in Table 2, the MAE and the MAX reduce up to 2.7% and 31.3%, respectively, with the ideal voting. With non-ideal voting, the MAE and the MAX reduce up to 1.5% and 9.3%, respectively. As it can be seen, the TMR technique is more effective when the noise rate is low. For example, for 0.1%, the MAX reduces from 56.3% in Table 2 to 25%, with the ideal TMR in Table 3. But again, if noise also impacts the voting logic, the MAX quickly increases to high rates. Compared to the SC method, the TMR method provides a lower MAE but a significant MAX for a 0.1% noise rate. For high noise rates, the SC method achieves much lower error rates. The TMR method further costs 3× latency or area [11].

3.3 Addition/Subtraction

Binary addition techniques for memristive IMC are proposed in [21]. A full-adder is implemented with eight NOR and four NOT operations in 13 cycles. Subtraction can be implemented using this addition technique with inputs in two's complement representation. We implemented the latency optimized adder of [21] for reliability evaluation. An OR-based SC addition is implemented in [6]. They generate OR of N bits in a single cycle. The operation is executed in parallel for the entire bit-stream and takes only one cycle to compute the final output. SC subtraction can be realized by performing bit-wise XOR on *correlated* bit-streams [1]. In-memory XOR can be performed by three NOR and two NOT operations, as elaborated in [17]. It can also be implemented using FELIX [5] by executing single cycle OR and NAND in crossbar memory. To be faster, SIXOR [19] can be used, which implements XOR in a single cycle. For reliability evaluation of SC subtraction, we use SIXOR. Table 4 compares the MAE and MAX rate of 8-bit subtraction using XOR operation and 256-bit correlated bit-streams for the SC approach with those of the 8-bit binary subtraction with and without TMR. For the TMR technique, we again evaluate both ideal and non-ideal voting logic. In both SC and binary approaches, noise is only injected into logical operations. As shown, the TMR technique can significantly improve the MAE of the binary subtraction. MAX error, however, is still high for noise rates as small as 0.1%. SC subtraction not only provides high accuracy (0.0% error rate when there is no noise) but also achieves a high tolerance to error. For example, for the noise rate of 5%, the binary approach shows 25.3% MAE with ideal-TMR while the SC approach achieves 2.72% MAE.

Table 2: MAE, MAX, and STD of In-Memory Binary and SC 8-Bit Multiplication for Different Noise Rates for three cases of Injecting Noise into *Input* Data, *Logic* Operations, and *Both* Input and Logic.

SC-256	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
Input-MAE	0.19	0.37	0.69	1.17	1.48	2.26	4.26	6.19	8.1
Input-MAX	0.95	1.67	3.05	5.03	6.54	10.3	19.9	28.9	37.5
Input-STD	0.001	0.003	0.005	0.009	0.011	0.017	0.03	0.05	0.06
Logic-MAE	0.188	0.39	0.84	1.54	2.00	3.16	6.19	9.19	12.3
Logic-MAX	1.011	1.34	2.12	3.16	4.13	5.84	10.9	15.9	21.0
Logic-STD	0.001	0.002	0.004	0.008	0.010	0.016	0.03	0.05	0.06
Both-MAE	0.19	0.55	1.28	2.37	3.07	4.80	8.99	12.8	16.1
Both-MAX	1.01	2.06	4.12	7.38	9.42	15.2	29.7	38.3	48.1
Both-STD	0.01	0.004	0.007	0.013	0.017	0.03	0.05	0.07	0.08

Binary-8	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
Input-MAE	0.0	0.10	0.96	1.91	2.76	4.44	8.06	11.1	13.8
Input-MAX	0.0	49.4	73.2	73.4	73.6	77.6	89.5	92.3	94.1
Input-STD	0.0	0.01	0.05	0.06	0.08	0.09	0.12	0.14	0.15
Logic-MAE	0.0	0.87	6.66	10.8	13.9	18.3	24.7	28.1	30.2
Logic-MAX	0.0	56.3	87.7	94.3	97.0	99.8	99.4	99.9	99.9
Logic-STD	0.0	0.03	0.09	0.12	0.13	0.16	0.20	0.22	0.23
Both-MAE	0.0	0.95	7.20	11.6	14.8	19.2	25.4	28.6	30.6
Both-MAX	0.0	59.5	88.8	99.0	98.1	99.8	99.8	99.9	99.8
Both-STD	0.0	0.03	0.10	0.12	0.14	0.16	0.20	0.22	0.23

Table 3: MAE and MAX of In-Memory Binary 8-Bit Multiplication with TMR for Different Noise Rates for case of Injecting Noise into *Logic* Operations

Binary-8	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
Ideal-TMR-MAE	0.0	0.16	4.49	8.43	11.2	15.6	22.0	25.7	28.0
Ideal-TMR-MAX	0.0	25.0	65.5	67.0	79.2	90.8	98.2	99.6	99.8
TMR-MAE	0.0	0.27	5.20	9.66	13.0	17.7	24.3	28.3	31.0
TMR-MAX	0.0	50.0	83.2	85.0	95.1	98.0	99.3	99.4	99.5

Table 4: MAE and MAX of 8-Bit In-Memory SC and Binary Subtraction with and without TMR for Different Noise Rates for case of Injecting Noise into *Logic* Operations

SC-256	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
Logic-MAE	0.0	0.39	0.78	1.33	1.73	2.72	5.24	7.78	10.3
Logic-MAX	0.0	0.39	1.17	2.34	3.12	5.07	10.1	15.2	20.3

Binary-8	0%	0.1%	1%	2%	3%	5%	10%	15%	20%
Logic-MAE	0.0	1.06	9.15	15.8	21.0	27.2	33.5	34.8	34.9
Logic-MAX	0.0	97.9	99.2	99.6	99.6	99.6	99.6	99.6	99.6
Ideal-TMR-MAE	0.0	0.05	3.75	10.3	16.4	25.3	33.1	35.2	35.6
Ideal-TMR-MAX	0.0	75.0	99.2	99.2	99.2	99.6	99.6	99.6	99.6
TMR-MAE	0.0	0.13	4.62	11.8	17.9	26.4	34.1	34.7	35.2
TMR-MAX	0.0	93.7	99.2	99.2	99.2	99.6	99.6	99.6	99.6

3.4 Minimum/Maximum

In-memory MAGIC-based execution of minimum and maximum functions in binary and stochastic formats are proposed in [2]. Binary execution of each of these two requires one N-bit comparator and one N-bit 2-to-1 multiplexer. If implementing both functions, the output of the comparator can be shared. The N-bit comparator is implemented by $11N-2$ NOR and $7N-2$ NOT logic gates. Each multiplexer is implemented by $3N$ NOR and $3N$ NOT operations. Execution of both functions completes after $6N+15$ cycles. This is 63 cycles for the case of processing 8-bit precision data.

Bit-wise logical AND and OR on *correlated* stochastic bit-streams give the minimum and maximum bit-stream, respectively. The AND operation is realized by first inverting the bit-streams through NOT and then performing bit-wise NOR on the inverted bit-streams, in a total of three cycles. On the other hand, the OR operation is executed by first performing bit-wise NOR on the input bit-streams and then NOT on the outputs of the NOR operations [2]. This is executed in two cycles. Similar to the SC multiplication and addition, the latency of SC minimum and maximum functions is independent of the bit-stream length. Table 5 compares the performance of the binary and SC-based maximum function for different noise rates applied to input data, logic operations, or both. Similar to what we observed for multiplication and addition, overall, SC achieves lower MAE, MAX, and STD compared to those of the binary-based IMC maximum function. Only for the case of having a noise rate equal to 0.1%, the binary technique shows a lower MAE compared to SC when noise is corrupting either inputs or logical operations.

4 DISCUSSION AND CONCLUSIONS

The traditional binary encoding is inherently more vulnerable to soft-errors compared to uniform stochastic representation. With binary encoding, the error (i.e., bit-flip) on the most significant bits of the data is critical, leading to significant errors in the data values and computations. Binary computing in memory involves a chain of operations in which the error in one step can propagate to the following steps. When implementing complex operations such as multiplication in memory based on binary encoding, the errors in different stages can accumulate, resulting in an unacceptable output. MAGIC-based in-memory multiplication, for example, requires many NOR operations executed during a large number of cycles [7]. Even failure in a small number of operations can lead to large overall error rates. IMC based on binary encoding further suffers from endurance and drift issues. Within the lifetime of memristive memories, a limited number of writes is possible. With the complexity of binary computations, which requires a large number of intermediate write operations, relying on memristive IMC is challenging.

SC representation and operations are inherently tolerant of soft-errors as any bit flip leads to only a LSB error [1]. Multiple bit-flips can even compensate each other; a state change from 0 to 1 on one bit position can be compensated by a state change from 1 to 0 on another bit position. Unlike binary computing, the computations in stochastic domain are extremely simple and inherently parallel. The operation on one bit is totally independent of the operation on other bits. Failure in one operation can only change one bit position and so have minimal impact on the output value. Error propagation is significantly lower compared to the propagation in complex binary operations. Simplifying the operations with SC can further reduce the processing latency significantly. For example, employing SC can reduce the latency of multiplying two 8-bit data from 726 cycles [7] to only six cycles [17] with no accuracy loss.

Table 6 compares the latency of basic arithmetic operations with the state-of-the-art binary and SC IMC techniques. As can be seen,

Table 5: MAE, MAX, and STD of In-Memory Binary and SC 8-Bit Maximum Function for Different Noise Rates for three cases of Injecting Noise into *Input* Data, *Logic* Operations, and *Both* Input and Logic.

SC-256	**0%**	**0.1%**	**1%**	**2%**	**3%**	**5%**	**10%**	**15%**	**20%**	**Binary-8**	**0%**	**0.1%**	**1%**	**2%**	**3%**	**5%**	**10%**	**15%**	**20%**
Input-MAE	0.0	0.32	0.77	1.42	1.83	2.86	5.39	7.73	9.90	Input-MAE	0.0	0.11	1.25	2.64	3.65	5.98	11.4	16.0	19.5
Input-MAX	0.0	0.78	2.34	4.68	6.25	10.15	20.31	29.29	38.67	Input-MAX	0.0	50.0	62.5	75.0	87.5	92.5	92.1	98.4	98.4
Input-STD	0.0	0.002	0.006	0.011	0.014	0.02	0.04	0.06	0.07	Input-STD	0.0	0.02	0.06	0.09	0.10	0.13	0.17	0.19	0.21
Logic-MAE	0.0	0.39	0.78	1.34	1.73	2.73	5.27	7.80	10.3	Logic-MAE	0.0	0.69	6.18	10.8	14.5	19.7	26.3	29.1	30.7
Logic-MAX	0.0	0.39	1.17	2.34	3.12	5.07	10.2	15.2	20.3	Logic-MAX	0.0	96.5	98.0	98.4	98.8	98.8	99.2	99.2	99.2
Logic-STD	0.0	0.000	0.003	0.008	0.01	0.02	0.03	0.04	0.05	Logic-STD	0.0	0.05	0.14	0.17	0.19	0.21	0.22	0.22	0.23
Both-MAE	0.0	0.50	1.22	2.23	2.88	4.48	8.22	11.5	14.3	Both-MAE	0.0	0.89	7.10	12.3	16.4	21.7	28.2	30.3	31.3
Both-MAX	0.0	1.17	3.51	7.03	9.37	15.2	28.1	38.6	47.2	Both-MAX	0.0	97.7	98.0	98.4	98.8	98.8	98.8	99.2	99.2
Both-STD	0.0	0.003	0.008	0.014	0.019	0.02	0.05	0.07	0.08	Both-STD	0.0	0.05	0.15	0.18	0.20	0.21	0.23	0.23	0.23

Table 6: Latency (# of clock cycles) Comparison of Binary and SC Arithmetic IMC Techniques

Operation	IMC	Ref	# of clock cycles for 4,8,N Data Width		
			4	8	N
Addition.	Bin.	[21]	49	97	$12N+1$
	SC	[6]	1	1	1
Subtract.	Bin.	[21]	49	97	$12N+1$
	SC	[17]	5	5	5
	SC	[5]	2	2	2
	SC	[19]	1	1	1
Multiplic.	Bin.	[7]	158	726	$13N^2-14N+6$
		[8]	195	871	$15N^2-11N-1$
		[13]	77	237	$15N^2+16N-19$
		[16]	101	279	$(10N+2)logN+4N+2$
		[12]	103	211	$NlogN+23N+3$
	SC	[17]	6	6	6
Maximum	Bin.	[2]	39	63	6N+15
Minimum	SC	[2]	5	5	5

the latency of SC-based IMC operations is *constant* and *independent* of data width. The longtime inaccuracy weakness of SC operations is now addressed by generating and processing LD bit-streams. SC with LD bit-streams can provide completely accurate results [17]. For most SC operations, short LD bit-streams can produce acceptable results with low error rates, and hence, the memory space can be used efficiently. The current challenges in reliable fabrication of memrisitive devices and IMC using memrisitive memorises demand for employing unconventional noise-tolerant computing techniques such as SC for reliable IMC. The computation results may not be 100% accurate, but they are achievable with current technology.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants #2127780 and #2019511, the Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, Semiconductor Research Corporation, Office of Naval Research, grant #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Cisco, Xilinx, and Nvidia.

REFERENCES

[1] Armin Alaghi, Weikang Qian, and John P. Hayes. 2018. The Promise and Challenge of Stochastic Computing. *IEEE Trans. on Computer-Aided Design of Integ. Circuits and Systems* 37, 8 (2018), 1515–1531.

[2] Mohsen Riahi Alam, M. Hassan Najafi, and Nima TaheriNejad. 2022. Sorting in Memristive Memory. *J. Emerg. Technol. Comput. Syst.* (Jan 2022).

[3] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and Richard Stanley Williams. 2010. 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature* 464 (April 2010), 873–876.

[4] Qiao Chen, Xiaoping Wang, Haibo Wan, and Ran Vang. 2017. A Logic Circuit Design for Perfecting Memristor-Based Material Implication. *IEEE TCAD* 36, 2 (2017), 279–284.

[5] Saransh Gupta, Mohsen Imani, and Tajana Rosing. 2018. FELIX: Fast and Energy-Efficient Logic in Memory. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–7.

[6] Saransh Gupta, Mohsen Imani, Joonseop Sim, Andrew Huang, Fan Wu, M. Hassan Najafi, and Tajana Rosing. 2020. SCRIMP: A General Stochastic Computing Architecture using ReRAM in-Memory Processing. In *2020 DATE*. 1598–1601.

[7] Ameer Haj-Ali, Rotem Ben-Hur, Nimrod Wald, and Shahar Kvatinsky. 2018. Efficient Algorithms for In-Memory Fixed Point Multiplication Using MAGIC. In *2018 IEEE Intern. Symp. on Circuits and Systems (ISCAS)*.

[8] Mohsen Imani, Saransh Gupta, and Tajana Rosing. 2017. Ultra-efficient processing in-memory for data intensive applications. In *2017 54th Design Automation Conference (DAC)*. 1–6.

[9] P. Knag, W. Lu, and Z. Zhang. 2014. A Native Stochastic Computing Architecture Enabled by Memristors. *IEEE Trans. on Nanotechnology* 13, 2 (March 2014).

[10] Orian Leitersdorf, Ben Perach, Ronny Ronen, and Shahar Kvatinsky. 2021. Efficient Error-Correcting-Code Mechanism for High-Throughput Memristive Processing-in-Memory. In *58th ACM/IEEE DAC*.

[11] Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky. 2021. Making Memristive Processing-in-Memory Reliable. In *2021 28th IEEE Intern. Conf. on Electronics, Circuits, and Systems (ICECS)*. 1–6.

[12] Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky. 2021. MultPIM: Fast Stateful Multiplication for Processing-in-Memory. arXiv:2108.13378 [cs.AR]

[13] Zhaojun Lu, Md Tanvir Arafin, and Gang Qu. 2021. RIME: A Scalable and Energy-Efficient Processing-In-Memory Architecture for Floating-Point Operations. In *ASP-DAC'21* (Tokyo, Japan).

[14] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. 2011. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *Computers, IEEE Trans. on* 60, 1 (Jan 2011), 93–105.

[15] D. Radakovits and N. TaheriNejad. 2019. Implementation and Characterization of a Memristive Memory System. In *IEEE CCECE*. 1–5.

[16] David Radakovits, Nima TaheriNejad, Mengye Cai, Théophile Delaroche, and Shahriar Mirabbasi. 2020. A Memristive Multiplier Using Semi-Serial IMPLY-Based Adder. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 5 (2020), 1495–1506.

[17] Mohsen Riahi Alam, M. Hassan Najafi, and Nima TaheriNejad. 2021. Exact Stochastic Computing Multiplication in Memristive Memory. *IEEE Design Test* (2021), 1–6.

[18] Mohsen Riahi Alam, M. Hassan Najafi, Nima Taherinejad, Mohsen Imani, and Raju Gottumukkala. 2022. Stochastic Computing in Beyond Von-Neumann Era: Processing Bit-Streams in Memristive Memory. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 5 (2022), 2423–2427.

[19] N. TaheriNejad. 2021. SIXOR: Single-cycle In-memristor XOR. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 29, 5 (2021), 925–935.

[20] N. TaheriNejad and D. Radakovits. 2019. From Behavioral Design of Memristive Circuits and Systems to Physical Implementations. *IEEE Circuit and Systems (CAS) Magazine* 19, 4 (Fourthquarter 2019), 6–18.

[21] Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky. 2016. Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC). *IEEE Transactions on Nanotechnology* 15, 4 (2016), 635–650.

[22] Mohammed A. Zidan, John Paul Strachan, and Wei D. Lu. 2018. The future of electronics based on memristive systems. *Nature electronics* 1 (2018), 22–29.