SCRIMP: A General Stochastic Computing Architecture using ReRAM in-Memory Processing

Saransh Gupta*, Mohsen Imani*, Joonseop Sim*, Andrew Huang*, Fan Wu*, M. Hassan Najafi†, Tajana Rosing*
*University of California San Diego, La Jolla, CA 92093, USA
{sgupta, moimani, j7sim, anh162, f2wu, tajana}@ucsd.edu
†University of Louisiana, Lafayette, LA 70504, USA
najafi@louisiana.edu

***Abstract*—Stochastic computing (SC) reduces the complexity of computation by representing numbers with long independent bit-streams. However, increasing performance in SC comes with increase in area and loss in accuracy. Processing in memory (PIM) with non-volatile memories (NVMs) computes data in-place, while having high memory density and supporting bit-parallel operations with low energy. In this paper, we propose SCRIMP for stochastic computing acceleration with resistive RAM (ReRAM) in-memory processing, which enables SC in memory. SCRIMP can be used for a wide range of applications. It supports all SC encodings and operations in memory. It maximizes the performance and energy efficiency of implementing SC by introducing novel in-memory parallel stochastic number generation and efficient implication-based logic in memory. To show the efficiency of our stochastic architecture, we implement image processing on the proposed hardware.**

I. Introduction

The era of Internet of Things (IoT) is expected to create billions of inter-connected devices which are expected to be doubled every year [1], [2]. To ensure network scalability, security, and system efficiency, much of IoT data processing need to run at least partly on the devices at the edge of the internet [3]. However, running data intensive workloads with large datasets on traditional cores results in high energy consumption and slow processing speed due to the large amount of data movement between memory and processing units. Interestingly, new computing paradigms have shown the capability to perform complex computations at lower area and power costs [4], [5], [6]. Stochastic Computing (SC) [7] is one such paradigm, which represents each data point in the form of a bit-stream, where the probability of having '1's corresponds to the value of the data [8], [9]. Representing data in such a format does increase the size of data, with SC requiring 2^n bits to precisely represent an n-bit number. However, it comes with the benefit of extremely simplified computations and tolerance to noise [8], [10]. However, with all its positives, SC comes with some disadvantages. (i) Generating stochastic numbers is expensive and is a key bottleneck in SC designs, consuming as much as 80% [11] of total design area. (ii) Increasing the accuracy of SC requires increasing the bit-stream length, resulting in higher latency and area. (iii) Increasing the speed of SC comes at the expense of more logic gates, resulting in larger area. These pose a big challenge which cannot be solved with today's CMOS technology.

Processing In-Memory (PIM) is an implementation approach that uses high-density memory cells as computing elements [12]. Specifically, PIM with non-volatile memories (NVMs) like resistive random accessible memory (ReRAM) has shown great potential for performing in-place computations and hence, achieving huge benefits over conventional computing architectures [13], [14], [15], [16], [17], [18]. ReRAM boast of (i) small cell sizes, making it suitable to store and process large bit-streams [19], (ii) low energy consumption for binary computations, making it suitable for huge number of bitwise operations in SC, (iii) high bit-level parallelism, making it suitable for bit-independent operations in memory, and (iv) stochastic nature at sub-threshold level, making it suitable for generating stochastic numbers.

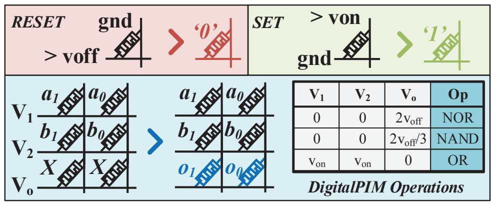
SCRIMP combines the basic properties of ReRAM and SC to make implementation of SC on PIM highly efficient. First, SCRIMP exploits the stochastic nature of ReRAM devices to propose a new stochastic number generation scheme. This completely eliminates the use of stochastic number generators which can consume up to 80% area on a SC chip. It implements, for the first time, implication logic in regular crossbars. This enables SCRIMP to combine various logic families to execute logic operations more efficiently. SCRIMP implementation of basic SC operators using implication logic are faster and more efficient than state-of-the-art. We evaluate SCRIMP over six general image processing applications.

II. Related Work

A. Stochastic Computing

Stochastic computing (SC) represents numbers in terms of probabilities in long independent bit-streams. Unlike multiplication, stochastic addition, or accumulation, is not a simple operation. Several methods have been proposed which involve a direct trade-off between the accuracy and complexity of operation [20], [21], [22]. Many arithmetic functions like trigonometric, logarithmic, and exponential functions can be approximated in stochastic domain with acceptable accuracy [11], [23]. Stochastic computing is enabled by stochastic number generators (SNGs), which perform binary to stochastic conversion.

SC re-emerged as an active area of research with the introduction of IoT, where devices are small, less complex, and need low latent results. There are recent work in multiple directions. Some try to improve the efficiency of SC operations



V_1	V_2	V_o	Op
0	0	$2v_{off}$	NOR
0	0	$2v_{off}/3$	NAND
v_{on}	v_{on}	0	OR

Fig. 1. Implementing operations using digital PIM.

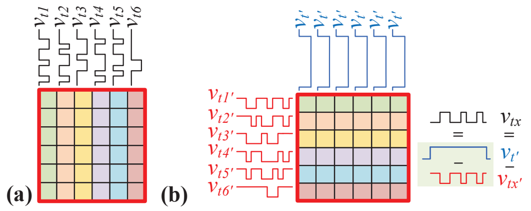


Fig. 2. Generation of stochastic numbers using (a) group write [33], (b) SCRIMP row-parallel generation.

by proposing new approximate implementations [21], [22]. The work in [24], [25], [26] propose new encoding schemes for SC which are more accurate than traditional encoding. Some work also optimize SC for different applications [9], [25], [26], [27], [28].

B. Digital Processing In Memory

A large number of recent designs enabling PIM in ReRAM are based on analog computing [13], [14], [15]. Some recent work has demonstrated ways to implement logic using ReRAM switching [29], [30]. Digital processing in-memory exploits variable switching of memristor to implement a variety of logic functions inside memory [30], [31]. Figure 1 shows how the output of operation changes with the applied voltage [30]. The output device switches whenever the voltage across it exceeds a threshold [29]. As shown, these operations can be implemented in parallel over multi-bits, even the entire row of memory. Digital PIM allows high density operations within memory without reading out the data. In this paper, we utilize digital PIM to implement a majority of stochastic operations. In addition, we also introduce, for the first time, support for an entire class of digital logic, i.e. implication logic, in regular crossbar memory using digital PIM.

III. Stochastic PIM

In this section, we present the hardware innovations which make SCRIMP efficient for SC. First, we present a PIM-B2S conversion technique. Then, we propose a new way to compute logic in memory. Next, we show how we bypass the physical limitations of previous PIM designs to achieve a highly parallel architecture. Last, we show the implementation different SC operations in SCRIMP.

A. Stochastic Number Generation

The ReRAM device switching is probabilistic at sub-threshold voltages, with the switching time following a Poisson distribution [32]. For a fixed voltage, the switching probability of a memristor can be controlled by varying the width of the programming pulse. The group write technique presented in [33] showed that stochastic numbers of large sizes can be generated over multiple bits of a column in parallel It first deterministically programs all the memory cells to zero (RESET) and then stochastically, based on the input number, programs them to one (SET). However, since digital PIM is row-parallel, it is desirable to generate such a number over a row. This can be achieved in two ways:

ON→OFF Group Write: To generate a stochastic number over a row, we need to apply the same programming pulse to the row. As shown before in Figure 1, the bipolar nature of memristor allows it to switch only to '0' by applying a voltage at the wordline. Hence, a ON→OFF group write is needed. Stochastic numbers can be generated over rows by applying stochastic programming pulses at wordlines instead of bitlines. However, a successful stochastic number generation requires us to SET all the rows initially. This results in a large number of SET operations. The SET phase is both slower as well as more energy consuming than the RESET phase, making this approach very inefficient. Hence, we propose a new generation method.

SCRIMP Row-Parallel Generation: The switching of memristor is based on the effective voltage across its terminals. In order to achieve low static energy for initialization, we RESET all the rows like the original group write. However, instead of applying different voltage pulses, $v_{t1}, v_{t2}, ...v_{tn}$, to different bitlines, we apply a common pulse, $v_{t'}$, to all the bitlines. A pulse, v_{tx}, applies a voltage v with a time width of tx. Now, we apply pulses, $v_{t1'}, v_{t2'}, ...v_{tn'}$, to different wordlines such that $v_{tx} = v_{t'} - v_{tx'}$. It generates stochastic numbers over multiple rows in parallel as shown in Figure 2b.

B. Efficient PIM Operations

SC multiplication with bipolar (unipolar, SM-SC) numbers involves XNOR (AND).While the digital PIM discussed in Section II-B implements these functions, they are inefficient in terms of latency, energy consumption, memory requirement, number of device switches. We propose to use a implication-based logic. Implication (→, where $A \rightarrow B = A' + B$) combined with false (always zero) presents a complete logic family. XNOR and AND are implemented using implication very efficiently as described in Table I. Some previous work implemented implication in ReRAM [34]. However, they required additional resistors of specific value to be added to the memory crossbar. Instead, SCRIMP enables, for the first time, implication-based logic in conventional crossbar memory, with the same components as the basic digital PIM. Hence, SCRIMP supports both implication and basic digital PIM operations.

SCRIMP Implication in-memory: As discussed in Section II-B, a memristor requires a voltage greater than v_{off} ($-v_{on}$) to switch from '1' ('0') to '0' ('1') to high resistive state (HRS, logical '0'). We exploit this switching mechanism to implement implication logic in-memory. Consider three cells, two input cells and an output cell, in a row of crossbar memory

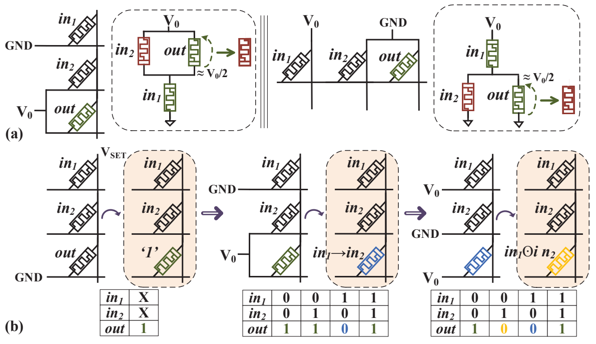


Fig. 3. (a) Implication in a column/row, (b) XNOR in a column.

TABLE I
COMPARISON OF THE PROPOSED XNOR AND AND WITH STATE-OF-THE-ART.

	Latency (cycles)		Energy (fJ)		Memory Req. (# of cells)		Device Sw. (# of cells)	
	XNOR	AND	XNOR	AND	XNOR	AND	XNOR	AND
SCRIMP	2	2	37.1	45.2	1	2	≤2	≤2
FELIX [30]	3	2	53.7	48.8	2	2	≤3	≤2
MAGIC [31]	5	3	120.29	64.1	5	3	≤5	≤5

as shown in Figure 3a. We apply an execution voltage, V_0, at the bitline corresponding to one of the inputs (in_1), while ground the other input (in_2) and the output cell (out). Let out be initialized to '1'. In this configuration, out switches to '0' only when the voltage across it is greater or equal to v_{off}. For all the cases when in_1 is '0,' most of the voltage drop is across in_1, resulting in a negligible voltage across out. In case in_1 is '1,' the voltage across out is ˜$V_0/3$ and ˜$V_0/2$ when in_2 is '1' and '0' respectively. If $2*v_{off} \leq V_0 < 3*v_{off}$, then out switches only when in_1 is '1' and in_2 is '0'. This results in the truth table shown in Figure 3a, corresponding to $in_1 \rightarrow in_2$. To execute $in_2 \rightarrow in_1$, V_0 is applied to in_2 while in_1 and out are grounded.

SCRIMP XNOR in-memory: XNOR ($\odot$) can be represented as, $A \odot B = (A \rightarrow B).(B \rightarrow A)$. Instead of calculating $in_1 \rightarrow in_2$ and $in_2 \rightarrow in_1$ separately and then ANDing them, we first calculate $in_1 \rightarrow in_2$ and then use its output cell to implement $in_2 \rightarrow in_1$ as shown in Figure 3b. In this way, we eliminate separate execution of AND operation.

SCRIMP AND in-memory: AND (.) is represented as, $A.B = (A \rightarrow B')'$. The inversion uses NOT presented in [34].

C. SC Arithmetic Operations in SCRIMP

Here, we explain how SCRIMP implements SC operations. The operands are either generated using the B2S conversion technique in Section III-A or are pre-stored in memory as outputs of previous operations. They are present in different rows of the memory, with their bits aligned. The output is generated in the output row, bit-aligned with the inputs.

Multiplication: As explained in Section II, multiplication of two numbers in stochastic domain involves a bitwise XNOR (AND) between bipolar (unipolar, SM-SC) numbers across the bit-stream length. This is implemented in SCRIMP using the PIM technique explained in Section III-B.

Conventional Addition/Subtraction/Accumulation: Implementations of different stochastic N-input accumulation techniques (OR, MUX, and count-based) discussed in Section II can be generalized to addition by setting the number of inputs to two. In case of subtraction, the subtrahend is first inverted using a single digital PIM NOT cycle. Then, any addition technique can be used. The OR operation is supported by SCRIMP using the digital PIM operations [30], generating OR of N bits in single cycle. The operation can be executed in parallel for the entire bit-stream, b_l, and takes just one cycle to compute the final output. To implement MUX-based addition in memory, we first stochastically generate b_l random numbers between 1 to N using B2S conversion in Section III-A. Each random number selects one of the N inputs for a bit position. The selected input bit is read using the memory sense amplifiers and stored in the output register. Hence, MUX-based addition takes one cycle to generate one output bit, consuming b_l cycles for all the output bits. To implement parallel count (PC)-based addition in memory, one input bit-stream (b_l bits) is read out by the sense amplifier every cycle and sent to counters. This is done for N inputs sequentially, consuming N cycles. In the end, counters store the number of ones at each bit position.

Other Arithmetic Operations: SCRIMP supports trigonometric, logarithmic, and exponential functions using truncated *Maclaurin Series* expansion [23]. The expansion approximates these functions using a series of multiplications and additions. With just 2-5 expansion terms, it produces more accurate results [23] than most other stochastic methods [11], [20].

IV. Evaluation

A. Experimental Setup

We develop C++-based cycle-accurate simulator which emulates the functionalities of SCRIMP. The simulator uses the performance and energy characteristics of the hardware are obtained from circuit level simulations for a 45nm CMOS process technology using Cadence Virtuoso. We use VTEAM memristor model [29] for our memory design simulation with R_{ON} and R_{OFF} of $10k\Omega$ and $10M\Omega$ respectively. As workload, we consider SCRIMP efficiency on four general image processing applications including: *Sobel, Robert, Prewitt*, and *BoxSharp*. We use images from *Caltech 101* [35] library.

B. SCRIMP Trade-offs

To evaluate the effect at application level, we implement the general applications listed above using SCRIMP with an input dataset of size 1kB. The results shown here use unipolar encoding with AND-based multiplication, SCRIMP addition, and Maclaurin series-based other arithmetic functions. Since all these operations are scalable with the bit-stream length, the latency of the operations doesn't change. The minor increase in the latency at application level with the length is due to the time taken by stochastic-to-binary conversion circuits. However, this change is negligible. Figure 4 shows the impact of bit-stream length on different applications. On an average, both the area and energy consumption of the applications increase by 8×, when the bit-stream length increases from 512 to 4096, with

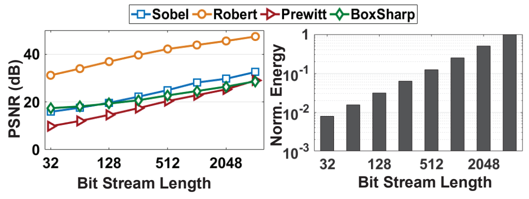


Fig. 4. Effect of bit-stream length on the accuracy and energy consumption for different applications.

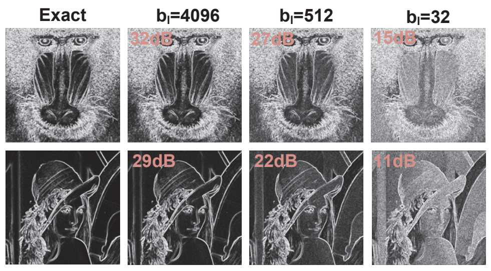


Fig. 5. Visualization of quality of computation in Sobel application.

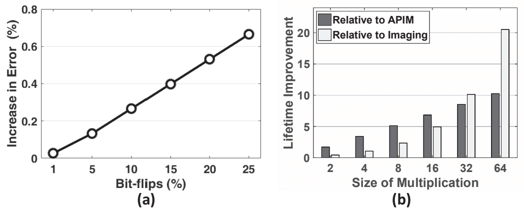


Fig. 6. SCRIMP's resilience to (a) memory bit-flips and (b) endurance.

an average 6.1dB PSNR gain. As shown in Figure 5, with a PSNR of 29dB, the output of Sobel filter with bit-stream length of 4096 is visibly similar to that of the exact computation.

C. SCRIMP and Memory Non-Idealities

Bit-Flips: Here, we evaluate the quality loss in SCRIMP with increase in the number of bit-flips. We evaluate the general applications with the same configuration as in Section IV-B with a bit-stream length of 1024. The quality loss is measured as the difference between accuracy with and without bit-flips. Figure 6a shows that with 10% bit-flips, the average quality loss is meagre 0.27%. When the bit-flips increase to 25%, applications lose only 0.66% in accuracy.

Memory Lifetime: Previous work [15], [34], [36] uses iterative process to implement multiplication and other complex operations. The more the iterations, higher is the number of operations and so is the per cell switching count. SCRIMP reduces this complex iterative process to just one logic gate, in case of multiplication, while it breaks down other complex operations into a series of simple operations. Hence, achieving less switching count per cell. Figure 6b shows that for multiplication, SCRIMP increases the lifetime of memory by 5.9× and 6.6× as compared to [15] and [36] respectively.

V. Conclusion

In this paper, we proposed SCRIMP, a general in memory processing architecture for stochastic computing on ReRAM. SCRIMP is a highly parallel architecture which scales with the size of stochastic computing. To achieve this, SCRIMP proposes novel in-memory stochastic number generation and implication in memory scheme. It supports all SC encoding schemes and operations fully in memory.

Acknowledgements

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration grant, and also NSF grants # 1527034, #1730158, #1826967, and #1911095.

References

[1] A. Al-Fuqaha *et al.*, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
[2] S. C. Mukhopadhyay *et al.*, "Internet of things: Challenges and opportunities," in *Internet of Things*. Springer, 2014, pp. 1–17.
[3] P. Schulz *et al.*, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, 2017.
[4] Y. Zhang *et al.*, "A survey on emerging computing paradigms for big data," *Chinese Journal of Electronics*, vol. 26, no. 1, pp. 1–12, 2017.
[5] A. K. Kar, "Bio inspired computing–a review of algorithms and scope of applications," *Expert Systems with Applications*, 2016.
[6] M. R. Jokar *et al.*, "Direct-modulated optical networks for interposer systems," in *NOCS*. ACM, 2019, p. 10.
[7] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 149–156.
[8] A. Alaghi *et al.*, "The promise and challenge of stochastic computing," *TCAD*, vol. 37, no. 8, pp. 1515–1531, 2018.
[9] A. Ren *et al.*, "Sc-dcnn: highly-scalable deep convolutional neural network using stochastic computing," in *ASPLOS*. ACM, 2017.
[10] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *DAC*. ACM, 2015, p. 59.
[11] W. Qian *et al.*, "An architecture for fault-tolerant computation with stochastic logic," *TC*, vol. 60, no. 1, pp. 93–105, 2011.
[12] S. Gupta *et al.*, "Exploring processing in-memory for different technologies," in *GLS-VLSI*. ACM, 2019.
[13] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.
[14] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ISCA*, 2016.
[15] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *DAC*. ACM, 2017.
[16] M. Imani et al, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*. ACM, 2019.
[17] M. R. Jokar *et al.*, "Cooperative nv-numa: prolonging non-volatile memory lifetime through bandwidth sharing," in *Proceedings of the International Symposium on Memory Systems*. ACM, 2018, pp. 67–78.
[18] M. Zhou *et al.*, "Gram: graph processing in a reram-based computational memory," in *ASP-DAC*. ACM, 2019.
[19] M. R. Jokar *et al.*, "Sequoia: A high-endurance nvm-based cache architecture," *IEEE TVLSI*, vol. 24, no. 3, pp. 954–967, 2015.
[20] B. D. Brown *et al.*, "Stochastic neural computation. i. computational elements," *IEEE TC*, vol. 50, no. 9, pp. 891–905, 2001.
[21] K. Kim *et al.*, "Approximate de-randomizer for stochastic circuits," in *ISOCC*. IEEE, 2015, pp. 123–124.
[22] P.-S. Ting *et al.*, "Stochastic logic realization of matrix operations," in *Euromicro*. IEEE, 2014, pp. 356–364.
[23] K. Parhi *et al.*, "Computing arithmetic functions using stochastic logic by series expansion," *TETC*, 2016.
[24] V. Canals *et al.*, "A new stochastic computing methodology for efficient neural network implementation," *IEEE TNNLS*, 2016.
[25] A. Zhakatayev *et al.*, "Sign-magnitude sc: getting 10x accuracy for free in stochastic computing for deep neural networks," in *DAC*. ACM, 2018.
[26] S. Li *et al.*, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *MICRO*. IEEE, 2018, pp. 696–709.
[27] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *DAC*. IEEE, 2016, pp. 1–6.
[28] H. Sim *et al.*, "Scalable stochastic-computing accelerator for convolutional neural networks," in *ASP-DAC*. IEEE, 2017, pp. 696–701.
[29] S. Kvatinsky *et al.*, "Vteam: a general model for voltage-controlled memristors," *TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.
[30] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*. ACM, 2018, p. 55.
[31] S. Kvatinsky *et al.*, "MAGIC – memristor-aided logic," *TCAS II*, vol. 61, no. 11, pp. 895–899, 2014.
[32] S. H. Jo *et al.*, "Programmable resistance switching in nanoscale two-terminal devices," *Nano letters*, vol. 9, no. 1, pp. 496–500, 2008.
[33] P. Knag *et al.*, "A native stochastic computing architecture enabled by memristors."
[34] S. Kvatinsky *et al.*, "Memristor-based material implication (IMPLY) logic: design principles and methodologies," *TVLSI*, 2014.
[35] "Caltech Library," http://www.vision.caltech.edu/Image_Datasets/Caltech101/.
[36] A. Haj-Ali *et al.*, "Efficient algorithms for in-memory fixed point multiplication using magic," in *ISCAS*. IEEE, 2018, pp. 1–5.