Machine Learning at the Edge Using Spintronic Stochastic Computing

Wojciech Romaszkan, Tianmu Li, Jiyue Yang, Albert Lee, Di Wu, Kang Wang, Sudhakar Pamarti, Puneet Gupta
Electrical and Computer Engineering Department
University of California, Los Angeles
Los Angeles, USA
Email: {litianmu95, wromaszkan, jyang669, alee0618, physicalwu, wang, spamarti, puneetg}@ucla.edu

***Abstract*—Stochastic computing (SC) has seen a renaissance in recent years as a means for machine learning acceleration due to its compact arithmetic and approximation properties. Still, SC accuracy remains an issue, with prior works either not fully utilizing the computational density or suffering from significant accuracy losses. In this work, we propose a set of optimizations targeting stream generation and execution components of SC, that bridges the accuracy gap between stochastic computing and fixed-point neural networks. Our techniques improve accuracy by coupling controlled stream sharing with training and balancing OR and binary accumulations. They further optimizes the SC execution through progressive shadow buffering, spintronic non-volatile memory and architectural optimizations. We show that they can improve accuracy compared to state-of-the-art SC by 2.2-4.0% points while being up to 4.4X faster and 6.8X more energy efficient. Our optimizations eliminate the accuracy gap between SC and fixed-point architectures while delivering up to 5.6X higher throughput and 2.9X lower energy.**

I. Introduction

The rapid growth of deep learning in the past decade has created an immense demand for computing power at both the cloud and edge. Multiple algorithmic, architectural, and circuit approaches have been proposed to meet this demand. Among those, stochastic computing (SC) has been enjoying a renaissance in deep learning acceleration for latency-, energy-, and cost-constrained devices [1]–[5]. It offers a very compact computing footprint, enabling high levels of parallelism and data reuse not achievable using conventional floating- or fixed-point architectures [5]. Its approximate nature synergizes well with neural networks' inherent error-tolerant properties, enabling new axes of accuracy and performance tradeoffs [3], [5], [6].

Stochastic computing represents numbers using the proportion of ones in a random bitstream and enables multiplication and addition using simple logic gates. To improve accuracy, the majority of prior works opted for approximate parallel counter-based accumulation fabric [4], [6]–[8] or fixed-point accumuation [3], [9], losing computational density. Custom SC addition circuits have also been used [10], [11]. [5] showed OR-accumulation using split-unipolar stochastic streams to be a viable approach, but it suffers from significant accuracy loss. In this work, we show that those sacrifices are not necessary.

Stochastic bitstream generation has also received much attention. A typical stochastic number generator (SNG) has a random number generator (RNG) and a comparator that compares the target value with the random number [12]. Streams generated from a true random number generator (TRNG) have a predictable but high error [13]. Less expensive TRNGs [14]–[17] as well as quasi-random sequences [3], [4], [9] have been explored to reduce error and cost of stream generation. Correlation of the random sources in stream generation can severely impact accuracy and has forced most prior works to limit the amount of computation performed in the stochastic domain, sacrificing potential performance benefits [3], [4], [9].

Most SC literature focuses on SC "component" improvements [3], [9], [18] or implement dedicated network-specific accelerators [7], [19]. Programmable, full-system SC implementations [2], [5], like the focus of our work are rare. We account for overheads of generalizability of programmable accelerators and generate power, performance, and accuracy numbers for the entire compute+memory system.

In this work, we propose an ensemble of optimization techniques that can bridge the accuracy gap between stochastic and fixed-point accelerators while improving performance even when compared to state of the art stochastic accelerators. Our contributions are as follows:

- We show that, with appropriate training, neural networks can learn the biases caused by the use of pseudo-RNGs and extensive sharing of them in SNGs and *improve* accuracy compared to using non-shared TRNGs by as much as 6.1% points while reducing energy and area.
- We propose a progressive stream generation and shadow buffering scheme that reduces required memory bandwidth by up to 4X while improving latency by as much as 2X.
- We propose using a balanced mix of stochastic OR and fixed-point accumulation to improve accuracy by up to 9.4% points. The increase in accuracy allows us to reduce stream length by 4X while still maintaining 2.2-4.0% points accuracy advantage.
- We leverage aggressive pipelining and near memory computation to enable high throughput, maximal reuse,

Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

and efficient compute utilization regardless of layer parameters.
- We utilize on-chip, high-density VCMA MTJ storage for further area and energy reduction.

II. Stochastic Stream Generation Optimizations

This section proposes two methods optimizing the stream generation process. Combining shared stream generation and training improves accuracy, while progressive generation relieves memory bottleneck.

A. Co-optimized Shared Generation and Training

RNG Sharing has been shown to be detrimental to stochastic computing accuracy [20], [21] due to the correlation between streams. However, we hypothesize that a partially-shared generation with stream-based training leads to higher accuracy. Deterministic and repeatable (using a pseudorandom RNG) stream generators guarantees obtaining the same outputs from the same inputs, enabling the model to train for a fixed, instead of random error. We achieve determinism using maximal-length linear feedback shift registers (LFSR) as RNG. When generating streams of length 2^n, an n-bit maximal-length LFSR is used with a cycle of $2^n - 1$, and different initial seeds can be used to generate multiple uncorrelated streams. Sharing stream generation simplifies the error profile caused by SC. Assuming that all kernels in a layer share the same set of seeds, training only needs to deal with an error associated with one set of seeds.

To test this hypothesis, we implement three levels of sharing for a 4-layer CNN [22] on the SVHN dataset. Streams are represented using the split-unipolar format, and OR is used for accumulation, similar to [5]. In the "no sharing" case, each SNG gets a different seed for its LFSR. The "moderate sharing" case shares the same set of seeds across all kernels in a given layer. Finally, in the "extreme sharing" case, all rows of all kernels in a layer use the same set of seeds. The same is done when a true random number generator (TRNG) is used as an RNG. The results are shown in Figure 1. *At moderate sharing levels, LFSR-based SNGs show a significant uplift in the accuracy (up to 6.1% points compared to unshared TRNGs)*, adhering to the hypothesis. TRNG does not see the accuracy improvement with sharing due to the lack of determinism. However, both TRNG and LFSR suffer from a significant drop in accuracy from correlation when using extreme sharing. We also compared the validation accuracy when using LFSR without modeling it during training. No accuracy can be gained from moderate sharing when the model is not trained for it, and extreme sharing reduces accuracy to about 20%. We use the moderate sharing scheme in our implementation.

B. Progressive Stochastic Stream Generation

Once a set of weights and activations finish computation, the SNG buffers need to be reloaded for the next iteration of generation and computation which can be a significant bottleneck. We propose using a progressive generation scheme

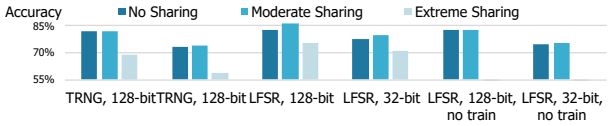

Fig. 1. Accuracy vs. sharing for TRNG and LFSR-based random number generation.

to alleviate this inefficiency, where stream generation begins as soon as the first 2 most-significant bits are loaded into the buffers instead of waiting for all 8 bits, as shown in Figure 2. As stream generation continues, the remaining bits are gradually loaded in groups of 2 bits every two cycles until the number of bits loaded matches the LFSR length used. As the SNG matches the LFSR length to the stream length being used, shorter stream lengths effectively truncate the converted fixed-point values. Our progressive buffering scheme can take advantage of that truncation to reduce the number of required memory accesses, which is not possible when all bits for a given value are being loaded in parallel. Compared to starting generation when all 8-bits are loaded, *progressive generation reduces the latency overhead of reloading by 4X.*

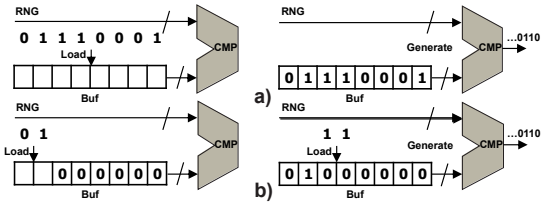

Fig. 2. Normal SNG (a) and progressive stream generation (b).

III. Stochastic Computing Execution Optimizations

This section describes the architecture design and a variety of micro-architectural optimizations to improve performance and accuracy.

A. Architecture Overview

Our accelerator architecture is based on the one proposed in [5], which uses fully-stochastic computation, and is agnostic to the stream generation scheme and supports extensive RNG sharing. The block diagram of the accelerator, including the optimizations described in subsequent sections, is shown in 3 a). The architecture can support a wide variety of activation and kernel sizes, as well as padding and pooling for convolutional layers. It supports fully-connected (FC) layers, although with compute underutilization. It is also fully programmable, with its own ISA and instruction memory.

B. Partial Binary Accumulation

As mentioned in Section I, many recent SC works opt to perform accumulation in the fixed-point domain or completely in SC to save costs. In contrast to these two extremes, we propose to use partial SC-fixed-point accumulation, where the first few levels of accumulation are implemented in SC using

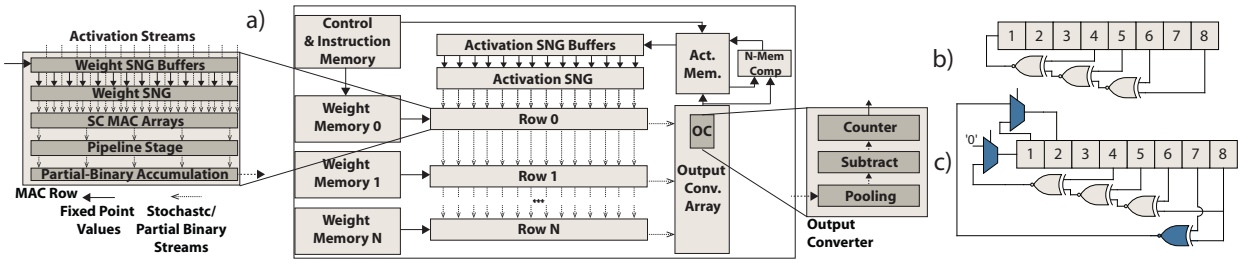

Fig. 3. Overall SC accelerator architecture block diagram. with breakdowns of the MAC row (left) and output converter (right) modules (a). Fixed 8-bit maximum length LFSR (b), and configurable 8- or 7-bit maximum length LFSR (c).

OR gates, before converting the intermediate results to fixed-point and computing the remainder of the accumulation.

The partition between SC and fixed-point accumulation affects both accuracy and performance. Using an approximate parallel counter (APC) [23] only allows one layer of SC accumulation before fixed-point accumulation. Using OR for accumulation allows an arbitrary trade-off between SC and fixed-point accumulation. We tested model accuracy with different fixed-point accumulation levels using the same setup as in Sec. II-A. Assuming weight filters are arranged into (C_{in}, H, W) dimensions, *performing fixed-point accumulation in the W dimension (PBW) improves accuracy by 4.5% and 9.4% respectively for 128-bit and 32-bit streams compared to performing all accumulations using OR*. Extending fixed-point accumulation to H (PBHW) as well improves accuracy by $< 0.5\%$. Using partial binary accumulation increases the dynamic range of outputs. To deal with this, we use an 8-bit fixed-point version of batch normalization (BN) before ReLU activation. While still potentially expensive, BN offers 5.5-6.5% points accuracy improvement.

Figure 4 shows the overhead, in terms of area, of implementing SC-based MAC units with partial binary accumulation stages. The area increase of PBW and PBHW is only 4% and 9% for large ones compared to complete or accumulataiojn. In comparison, complete binary accumulation and APC increase area by 3-5X. Given that PBW is almost identical accuracy-wise, the rest of the paper uses PBW as the default unless otherwise mentioned.

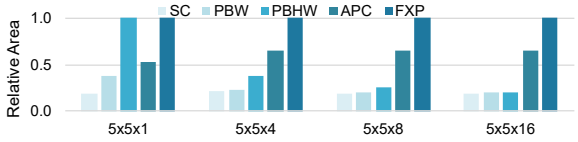

Fig. 4. Area comparison for different hardware implementations of SC-based MAC units for different kernel sizes and different levels of partial binary accumulation.

C. Near-Memory Computation

Organizing the compute hierarchy to mimic a vertically sliding convolutional window means that it naturally yields to the weight-stationary dataflow [24]. While the window iterates through the output tensor, weights can stay unchanged, and only a single row of activations needs to be reloaded between each computational pass. However, this is only true if a strict, weight-stationary implementation can be enforced. It requires that the MAC units' width and the corresponding number of SNGs are sized to fit the entire activation tensor covered by a kernel in a given layer, effectively "merging" weight- and output-stationary dataflows in one. However, it is not uncommon in modern neural networks to find kernels with thousands of weights, which cannot be fully unrolled. When that is not possible, the accelerator needs to store converted partial sums for later accumulation, or implement a strict output-stationary dataflow, accumulating output values in output conversion modules over multiple passes. Both weights and activations need to be swapped between each pass. Such dataflow can increase memory accesses by as much as 10.3X vs. ideal, weight-stationary implementation.

To alleviate this issue, we propose implementing near-memory accumulation, where the activation memory is tightly coupled with an array of adders. We then expand the ISA to support a 2-cycle read-add-write vector instruction that can be used to accumulate partial sums. Since partial sums are stored in large activation memory, there is no need to size it for any specific network or layer.

D. Pipeline Optimizations

On top of the generation optimizations from Section II and execution optimizations listed above, our accelerator includes three microarchitectural enhancements. First, we supplement the progressive generation with shadow buffers to hide loading latency. The overhead of shadow buffers is only about 4% at the whole accelerator level when coupled with progressive loading in Section II-B.

Second, we implement a pipeline stage within our compute engine between the SC and partial-binary accumulation stages. Implementing the pipeline stage in that location allows us to cut down the critical path by over 30% while minimizing the area required by additional flip-flops (<1% overhead on the accelerator level).

Third, we change how spatial split-unipolar computation is being performed. Conventional spatial split-unipolar, as shown in Figure 5 implements two stochastic MAC arrays, one for positive and one for negative weights [5]. However, since activations are assumed to be non-negative, the signs of multiplication results will always match the signs of a given weight. As such, multiplexing between positive and negative

values can be moved after multiplication, cutting down the number of required multipliers (AND gates) by half, and reducing the number of wires withing the MAC array. We estimate this optimization can improve the area and power of the compute itself by up to 26% and 28% respectively.

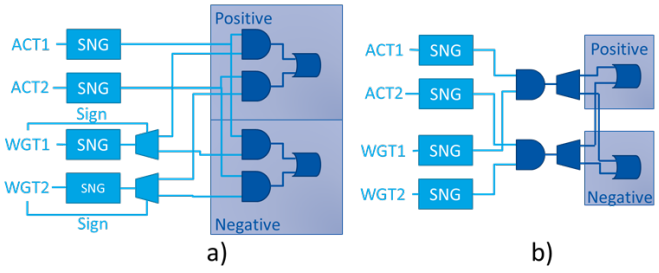

Fig. 5. Conventional spatial split-unipolar (a), and optimized spatial split-unipolar (b).

E. VCMA MTJ On-chip Memory

To further augment the density benefits of SC, we opt to use the emerging, non-volatile VCMA MTJ memory for on-chip storage. STT MTJ numbers are from [25]. VCMA MTJ numbers are based on our own simulations. As shown in Table I, it offers more than 4x density improvement over SRAM and alleviates Spin-Transfer Torque (STT) write energy and latency issues. While slower than eDRAM and SRAM, our architecture's capability of overlapping memory accesses with compute phases minimizes the performance reliance on memory access latency. Finally, it does not require refreshing, like eDRAM, which greatly simplifies scheduling.

TABLE I
READ (RD) AND WRITE (WR) LATENCY AND ENERGY (PER BITLINE) FOR DIFFERENT TYPES OF ON-CHIP MEMORIES.

	Unit Cell Area [um^2]	Time [ns] RD	Time [ns] WR	Energy [fJ] RD	Energy [fJ] WR
VCMA MTJ	0.03	4	< 1	27.5	46
STT MTJ	0.06	2.8	20	110	720
SRAM (32nm)	0.13	0.7	< 2	63	> 100
eDRAM (45nm)	0.04	1.3	< 2	48	> 70

IV. Evaluation & Results

A. Evaluation Methodology

We test accuracies on CIFAR-10, SVHN, and MNIST datasets. For CIFAR-10 and SVHN, we use the same 4-layer CNN [22] (CNN-4) as in Section III and VGG-16 [26]. For MNIST we use LeNet-5 [27]. Each model is trained with different stream lengths using split-unipolar implementation, and designated by two stream lengths $\{s_p - s\}$, s_p for layers with pooling and s for layers without.

To estimate the area, power, and latency of the proposed design, we have written individual blocks (SNGs, MAC arrays, buffers, output converters) in Verilog, and then synthesized them using a commercial 28nm HVT library. Memories were modeled using CACTI 6.5 [28], and our own models for the MTJ-based ones. To obtain accurate energy and latency estimates, we used a custom performance simulator, which combines the numbers from individual modules with a compiled code representing the given network model. We create two versions of our architecture: ultra-low power (ULP) and low-power (LP) targeted at different area-points and network sizes. ULP has 25.6K MACs with total on-chip memory of 150KB, while LP variant has 294K MACs and 0.5MB of on-chip memory.

As a fixed-point baseline, we use Eyeriss [24], scaled to 4-bit or 8-bit precision and 28nm node. We simulate the execution of the neural networks using [29]. For SC comparison points, we use the ACOUSTIC [5], Sign-Magnitude SC (SM-SC) [1] and SCOPE [2]. ACOUSTIC configurations are sized to have the same amount of memory and compute as our designs. SM-SC is not a fully programmable accelerator and SCOPE is a massive in-memory, DRAM-based accelerator [2], making full comparison impossible. Numbers are scaled to the 28nm node when necessary, using the models provided in [30]. We further compare the ULP variant with CONV-RAM [31] and MDL-CNN [32] mixed-signal accelerators.

B. Accuracy Comparisons

Table II compares accuracy of our approach with fixed-point and other SC implementations. *Our optimizations offer 2.2-4.0% points better accuracy at quarter stream length compared to [5] and similar accuracy at the same stream length compared to [1].* Compared to fixed-point, the accuracy with CNN-4 is comparable to 4-bit fixed-point when using 32-64 setup on SVHN, but 4% lower on CIFAR-10 when using 32-64 and 1.9% lower when using 64-128. Accuracy with VGG-16 is 2.2% lower than 8-bit fixed-point on CIFAR-10 and comparable on SVHN. Accuracy on MNIST is already comparable to fixed-point in the baseline, and our optimizations don't affect it. Compared to CONV-RAM [31], an in-memory architecture and MDL-CNN [32], a time-domain architecture, they offer superior accuracy even with 16-32 stream length.

C. Performance Impact of Proposed Enhancements

We compare the baseline ULP architecture (without our optimizations and 16-bit LFSRs to emulate TRNG) with three variants::

- GEN-128,128 - uses the generation optimizations from Section II. Progressive shadow buffers are used in this configuration.
- GEN-EXEC-32,64 - uses both the generation and execution (from Section III) optimizations. Further, it reduces the stream lengths being used to remain iso-accuracy with other configurations.
- GEN-EXEC-MTJ-32,64 - same as above, but using VCMA MTJ-based activation and weight memories.

Area, energy, and latency impacts of our optimizations on the ULP architecture are shown in Figure 6. Generation optimizations result in an overall 1% decrease in the accelerator

TABLE II
ACCURACY COMPARISON WITH FIXED-POINT, OTHER SC IMPLEMENTATIONS AND SO ON.

Dataset	Model	Eyeriss 8-bit	Eyeriss 4-bit	ACOUSTIC [5] 256	ACOUSTIC [5] 128	This Work 64-128	This Work 32-64	This Work 16-32	SCOPE [2] 128	CONV-RAM [31] 7a1w	MDL-CNN [32] 4a1w	SM-SC [1] 128
CIFAR-10	CNN-4	85.1%	82.1%	78.0%	74.9%	**80.2%**	**78.1%**	—	—	—	—	80%
	VGG-16	90.9%	—	—	—	**88.7%**	**88.7%**	—	—	—	—	—
SVHN	CNN-4	93.3%	90.5%	89.0%	86.8%	**91.9%**	**90.8%**	—	—	—	—	—
	VGG-16	96.2%	—	—	—	**96.0%**	**95.9%**	—	—	—	—	—
MNIST	LeNet-5	—	99.3%	—	99.3%	—	**99.3%**	**98.9%**	99.3%	96%	98.4%	—

area. At the same time, *the use of progressive shadow buffers to hide memory latency results in a 1.7X speedup and 1.6X reduction in energy*. Energy savings come mainly from SNG optimizations and reduced leakage.

Adding execution optimizations on top of the generation ones increases the area by 2% w.r.t. to baseline. The impact of pipelining and partial binary accumulation is minimal due to its limited application and an overall small contribution of the SC MAC array to the overall area. *The combination of shorter stream lengths, more efficient dataflow enabled by near-memory computation and pipelining coupled with DVFS results in 4.3X and 5.2X reduction in latency and energy w.r.t. baseline*. Finally, using VCMA MTJ on-chip memories can reduce the area by 1.8X and energy by 1.3X. Due to other optimizations decreasing our reliance on memory bandwidth, overall latency is not affected despite lower memory frequency compared to SRAM.

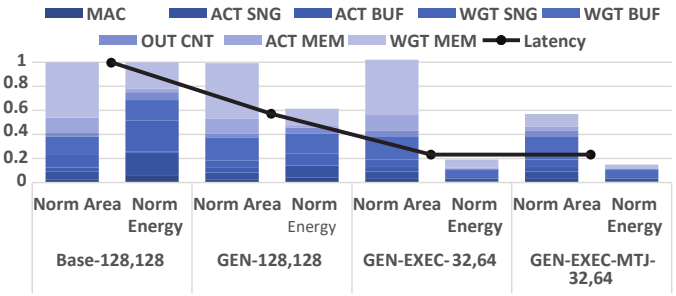

Fig. 6. Area, energy and latency comparison between baseline and different configurations of proposed optimizations (normalized to Base-128,128). The first bar in every group is for area while second is for energy.

D. GEO Performance Compared

Table III shows a comparison of the proposed ULP accelerator with fixed-point and mixed-signal approaches. First, we show that ULP-32,64 configuration outperforms the 4-bit fixed-point baseline in terms of throughput, by 2.7X, and energy efficiency, by 3.4X, with a smaller area footprint. It also outperforms ACOUSTIC-128, by 4.4X and 6.8X, respectively, while achieving higher accuracy. It is also highly-competitive in terms of energy-efficiency with mixed-signal accelerators like Conv-RAM and MDL-CNN. We refrain from comparing the throughput against those implementations due to the large area difference.

On the scale-out end of the spectrum, the LP variant outperforms 8-bit Eyeriss by 5.6X in terms of throughput and 2.9X in terms of energy efficiency. Modest energy reduction is caused by the high cost of external memory accesses - when those are omitted, the LP variant is as much as 6.1X more energy-efficient than Eyeriss. It is also 2.4X faster and 1.8X more energy efficient than ACOUSTIC, while having higher inference accuracy. Despite occupying only 2% of SCOPE area, our accelerator has nearly 24% of its peak throughput and has 2.4X better energy efficiency.

TABLE III
COMPARISON BETWEEN THE ULP VARIANT AND FIXED-POINT AND NEUROMORPHIC IMPLEMENTATIONS. NUMBERS ARE SCALED TO 28NM.

	Eyeriss 4-bit [24]	**ULP -32,64**	Conv-RAM [31]	MDL CNN [32]	ACOUSTIC ULP-128 [5]	**ULP -16,32**
Voltage	0.9	**0.81**	0.9	0.537	0.9	**0.81**
Area [mm^2]	0.59	**0.33**	0.02	0.06	0.57	**0.33**
Power [mW]	20	**48**	0.016	0.02	72	**48**
Clock [MHz]	400	**400**	364	25	400	**400**
Precision	4-bit	—	6b/1b	8b/1b	—	—
CIFAR-10 Fr/s	5.2k	**14k**	———	———	3.2k	**29k**
CIFAR-10 Fr/J	115k	**387k**	———	———	57k	**731k**
LeNet5 CNN Fr/s	47k	**520k**	15k	1k	180k	**780k**
LeNet5 CNN Fr/J	790k	**53M**	117M	50M	11M	**71M**
Peak GOPS	80	**640**	10.7	0.365	160	**1280**
Peak TOPS/W	4	**16.9**	44.2	18.2	2.22	**33.8**

TABLE IV
COMPARISON BETWEEN THE LP VARIANT AND FIXED-POINT AND SC IMPLEMENTATIONS. NUMBERS ARE SCALED TO 28NM.

	Eyeriss 8-bit [24]	**LP -64,128**	SM-SC [1]	SCOPE [2]	ACOUSTIC LP-256 [5]	**LP -32,64**
Voltage	0.9	**0.81**	0.9	—	0.9	**0.81**
Area [mm^2]	9.3	**5.7**	———	273	9	**5.7**
Power [mW]	848	**797**	———	———	1160	**797**
Clock [MHz]	400	**400**	1536	200	400	**400**
CIFAR-10 VGG Fr/s	555	**3.1k**	———	———	1.3k	**5.2k**
CIFAR-10 VGG Fr/J	618	**1.8k**	———	———	1k	**2.5k**
Peak GOPS	204	**1.8k**	1.7k	7.1k	460	**3.6k**
Peak TOPS/W	0.48	**2.5**	0.92	———	0.4	**5.1**

V. CONCLUSION

In this paper, we present generation- and computation-optimized stochastic computing architecture for neural network acceleration. We develop an ensemble of accuracy improvement (co-optimized stream generation and training, partial binary accumulation) and energy/runtime improvement (progressive stream generation, near memory computation, shadow buffering and pipelining) techniques. These optimizations improve accuracy by 2.2-4.0% points compared to state of the art SC-based accelerators while also being 4.4X faster

and 6.8X more energy efficient. Our architecture can compete with fixed-point implementations with similar accuracy and area while delivering up to 5.6X throughput, 2.9X energy-efficiency gains. Despite being an all-digital, programmable accelerator, it can achieve energy efficiency comparable to in-memory/mixed-signal accelerators. Our ongoing work focuses on taping out a silicon prototype of ULP variant and developing fast training approaches for stochastic computing.

ACKNOWLEDGMENT

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-27867. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advance Research Projects Agency (DARPA) or the U.S. Government.

REFERENCES

[1] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-Magnitude SC : Getting 10X Accuracy for Free in Stochastic Computing for Deep Neural Networks ," *DAC*, pp. 1–6, 2018.

[2] Y. Li, Shuangchen and Oliver Glova, Alvin and Hu, Xing and Gu, Peng and Niu, Dimin and T. Malladi, Krishna and Zheng, Hongzhong and Brennan, Bob and Xie, "SCOPE: A Stochastic Computing Engine for DRAM-based In-situ Accelerator," in *IEEE Micro*, 2018, pp. 697–710.

[3] R. Hojabr, K. Givaki, S. M. R. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "SkippyNN : An Embedded Stochastic-Computing Accelerator for Convolutional Neural Networks," in *DAC*, 2019, pp. 1–6.

[4] S. R. Faraji, M. H. Najafi, B. Li, D. J. Lilja, and K. Bazargan, "Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing," in *DATE*, 2019, pp. 1757–1762.

[5] W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "ACOUSTIC : Accelerating Convolutional Neural Networks through Or-Unipolar Skipped Stochastic Computing," in *DATE*, 2020.

[6] Z. Yawen, Z. Xinyue, S. Jiahao, W. Yuan, H. Ru, and W. Runsheng, "Parallel Convolutional Neural Network (CNN) Accelerators Based on Stochastic Computing," in *SiPS*, 2019, pp. 19–24.

[7] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Others, "HEIF: Highly Efficient Stochastic Computing based Inference Framework for Deep Neural Networks," *IEEE TCAD*, vol. 0070, no. c, pp. 1–14, 2018.

[8] B. Li, M. H. Najafi, and D. J. Lilja, "Low-Cost Stochastic Hybrid Multiplier for Quantized Neural Networks," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, 2019.

[9] H. Sim and J. Lee, "A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks," pp. 1–6, 2017.

[10] D.-a. Nguyen, H.-h. Ho, D.-h. Bui, and X.-t. Tran, "An Efficient Hardware Implementation of Artificial Neural Network based on Stochastic Computing," in *NICS*. IEEE, 2018, pp. 237–242.

[11] B. Li, S. Koester, and D. J. Lilja, "Low Cost Hybrid Spin-CMOS Compressor for Stochastic Neural Networks," in *GLSVLSI*, 2019, pp. 141–146.

[12] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*, 1969, pp. 37—-172.

[13] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *DATE*. Leuven, BEL: European Design and Automation Association, 2014.

[14] X. Ma, L. Chang, S. Li, L. Deng, Y. Ding, and Y. Xie, "In-memory multiplication engine with SOT-MRAM based stochastic computing," *CoRR*, vol. abs/1809.0, 2018.

[15] A. Mondal and A. Srivastava, "Data driven optimizations for MTJ based stochastic computing," *CoRR*, vol. abs/1804.03228, 2018.

[16] M. W. Daniels, A. Madhavan, P. Talatchian, A. Mizrahi, and M. D. Stiles, "Energy-Efficient Stochastic Computing with Superparamagnetic Tunnel Junctions," *Physical Review Applied*, vol. 13, no. 3, p. 1, 2020.

[17] S. Wang, S. Pal, T. Li, A. Pan, C. Grezes, P. Khalili-Amiri, K. L. Wang, and P. Gupta, "Hybrid vc-mtj/cmos non-volatile stochastic logic for efficient computing," in *DATE*, 2017, pp. 1438–1443.

[18] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "Ugemm: Unary computing architecture for gemm applications," in *ISCA*, 2020, pp. 377–390.

[19] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan, and Y. Wang, "SC-DCNN: Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing," *CoRR*, vol. abs/1611.0, 2016.

[20] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," *ICCD*, pp. 361–366, 2014.

[21] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *DSD*, 2017, pp. 1–8.

[22] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *CoRR*, vol. abs/1801.06601, 2018.

[23] K. Kim, J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," in *ISOCC*, 2015, pp. 123–124.

[24] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *Proc. ISSCC*, vol. 52, no. 1, pp. 262–263, 2016.

[25] Q. Dong, Z. Wang, J. Lim, Y. Zhang, Y. Shih, Y. Chih, J. Chang, D. Blaauw, and D. Sylvester, "A 1mb 28nm stt-mram with 2.8ns read access time at 1.2v vdd using single-cap offset-cancelled sense amplifier and in-situ self-write-termination," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 480–482.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, pp. 1–14, 2014.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] H. Labs, "CACTI-6.5 (Cache Access Cycle Time Indicator)." [Online]. Available: http://www.hpl.hp.com/research/cacti/

[29] M. Gao and M. Horowitz, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017, pp. 751–764.

[30] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration, the VLSI Journal*, vol. 58, no. January, pp. 74–81, 2017.

[31] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *ISSCC*, 2018, pp. 488–490.

[32] A. Sayal, S. Fathima, S. S. T. Nibhanupudi, and J. P. Kulkarni, "All-Digital Time-Domain CNN Engine Using Bidirectional Memory Delay Lines for Energy-Efficient Edge Computing," *ISSCC*, vol. 49, no. 4, pp. 228–230, 2019.