uGEMM: Unary Computing Architecture for GEMM Applications

Di Wu
Department of ECE
University of Wisconsin–Madison
Madison, WI, USA
di.wu@ece.wisc.edu

Jingjie Li
Department of ECE
University of Wisconsin–Madison
Madison, WI, USA
jingjie.li@wisc.edu

Ruokai Yin
Department of ECE
University of Wisconsin–Madison
Madison, WI, USA
ryin25@wisc.edu

Hsuan Hsiao
Department of ECE
University of Toronto
Toronto, ON, CA
julie.hsiao@mail.utoronto.ca

Younghyun Kim
Department of ECE
University of Wisconsin–Madison
Madison, WI, USA
younghyun.kim@wisc.edu

Joshua San Miguel
Department of ECE
University of Wisconsin–Madison
Madison, WI, USA
jsanmiguel@wisc.edu

***Abstract*—General matrix multiplication (GEMM) is universal in various applications, such as signal processing, machine learning, and computer vision. Conventional GEMM hardware architectures based on binary computing exhibit low area and energy efficiency as they scale due to the spatial nature of number representation and computing. Unary computing, on the other hand, can be performed with extremely simple processing units, often just with a single logic gate. But currently there exist no efficient architectures for unary GEMM.**

In this paper, we present uGEMM, an area- and energy-efficient unary GEMM architecture enabled by novel arithmetic units. The proposed design relaxes previously-imposed constraints on input bit streams—low correlation and long stream length—and achieves superior area and energy efficiency over existing unary systems. Furthermore, uGEMM's output bit streams exhibit higher accuracy and faster convergence, enabling dynamic energy-accuracy scaling on resource-constrained systems.

I. Introduction

General matrix multiplication (GEMM) is a ubiquitous operation essential in many emerging applications, particularly deep neural networks (DNNs). Though traditionally implemented as software libraries for CPUs and GPUs [29, 46–48], recent works strive for improved energy efficiency via hardware acceleration exploiting parallel multiplications [11, 22, 28, 36, 37, 52, 64, 71]. However, as we maximize parallelism, conventional binary computing where numbers are represented and computed in parallel multiple bits suffers from poor hardware area and energy efficiency due to exponentially growing wire congestion [66].

Unary computing is a promising solution to improve the area and energy efficiency of massively parallel computing by representing numbers as serial bit streams computed by extremely area- and energy-efficient processing units [6]. Benefits of unary computing over non-unary computing have already been proven in prior works [7, 33, 41, 57]. Given analog input from sensors [3, 63], unary computing can directly process sensor data converted by simple analog-to-stochastic converters in a fully-streaming manner, mitigating the energy overheads of conversion and storage in binary computing. Provided binary input and sufficiently high computational intensity, unary computing can embrace orders of magnitude more energy efficiency than binary [31]. Our goal in this work is not to argue superiority over binary systems but rather enable unparalleled efficiency for unary computing, since it has shown promise in recent years.

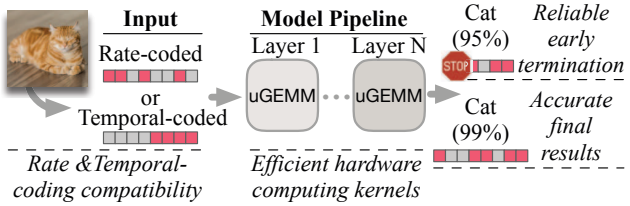


Fig. 1. An illustrative example of uGEMM.

Unfortunately, existing unary computing systems based on *rate coding*, namely *stochastic computing* [14], fail to fully realize the benefit of high area and energy efficiency because processing units require uncorrelated bit streams or long, repeated bit streams as input. As a result, they suffer from either costly random number generators (RNGs) or long latency, both of which result in low energy efficiency. Unary computing systems based on *temporal coding*, like *race logic* [38], have been proposed more recently to mitigate these limitations, but no efficient adder and multiplier units are available, making it inapplicable to GEMM applications.

In this work, we take a novel unary computing approach to enable efficient GEMM processing on extremely area- and energy-constrained devices, overcoming the limitations of existing unary systems. We argue that an ideal unary computing system should not be confined to only either rate or temporal encoding schemes but rather strive for the best of all worlds. We propose a *unified unary GEMM architecture*, dubbed uGEMM, that works with both rate and temporal

978-1-7281-4661-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ISCA45697.2020.00040

encoding schemes and overcomes the above weaknesses of current unary computing systems as illustrated in Figure 1. It is the first to support fully streaming execution and reliable early termination, where bit streams are processed in a continuous pipeline (without conversion between binary and unary), and the output accuracy stabilizes early. This offers unprecedented flexibility to dynamically trade off accuracy for latency to meet hard constraints on energy efficiency.

In this paper, we describe how to overcome the fundamental challenges of building a unified unary computing system and present novel encoding-insensitive arithmetic units for unary GEMM. The contributions are summarized as follows:

- We present novel *unified* unary computing units for multiplication, scaled addition, and non-scaled addition that support both rate and temporal representations.
- We characterize state-of-the-art unary computing units based on robustness regarding *stability*—a new metric introduced in this paper—and insensitivity to correlation.
- We build the first *unified unary GEMM architecture* (uGEMM), demonstrating high accuracy, energy efficiency and low latency. uGEMM saves up to 98% energy (72% on average) compared to the most accurate state-of-the-art unary approaches, with accuracy loss less than 0.5% from the ideal output.
- We enable support for *early termination*, showing that uGEMM can reach >95% accuracy in up to 84% fewer cycles compared to the best rate-coded baseline and up to 99.6% fewer cycles to reach satisfactory stability compared to the best temporal-coded baseline.
- We develop an open-source PyTorch-based cycle-accurate simulator for unary computing [68]. We evaluate uGEMM at an application level via a multilayer perceptron case study, which shows 7% higher classification accuracy compared to conventional unary approaches and demonstrates support for early termination to save latency by 72% while maintaining accuracy.

II. Background and Related Work

In this section, we review key unary computing and GEMM concepts necessary for understanding our uGEMM design. First, we present a taxonomy of unary computing according to their data encoding methods: rate coding [2, 61] and temporal coding [60], shown in Figure 2. Then we review state-of-the-art implementations of GEMM architectures.

A. Rate Coding-Based Unary Computing

Rate coding is a coding scheme where information is contained in the frequency of an event. Rate coding is adopted in stochastic computing, which finds application in low-density parity-check (LDPC) decoding [62, 66], image processing [3, 45] and DNNs [34, 53, 58].

a) Data representation: Rate-coded data relies on the *frequency* of 1s and 0s in the bit stream to represent its value. There are various encoding schemes, the most widely used being unipolar and bipolar formats [14]. In the unipolar format, a value in the range of $[0, 1]$ is encoded as the probability of a bit being 1 in the corresponding bit stream. More specifically, to represent a value $V \in [0, 1]$, the bit stream must satisfy $P(S = 1) = V$, where S refers to a sampled bit in the bit stream, as shown in Figure 2. The bipolar format extends the value range to $[-1, 1]$, and the bit stream follows $P(S = 1) = (V + 1)/2$. Given these formats, any real number can be scaled and mapped into a rate-coded bit stream.

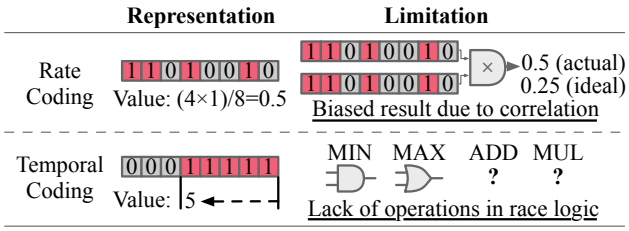


Fig. 2. Unary computing data representations and limitations.

b) Stochastic computing: Stochastic computing performs computation by manipulating input bit streams *statistically*. The majority of functions in binary computing are also supported in stochastic computing [14, 50, 67], and some examples of widely-used linear functional units are listed in Table I. In stochastic computing, an adder with two inputs A and B can be implemented by a two-input multiplexer (MUX) whose select signal is a stochastic bit stream with $P(S = 1) = 0.5$. This computes $V_{out} = (V_A + V_B)/2$, where a scaling factor of 2 is introduced to prevent overflow. Besides the scaled addition using MUX, nonscaled addition can be done with an OR gate. Multiplication can be achieved using an AND gate for unipolar bit streams and an XNOR gate for bipolar bit streams. Recent work focuses on designing efficient nonlinear functional units as well [9, 67].

c) Limitation — correlation problem: Since stochastic computing operates on the basis of statistics, the distribution of 1s in the input bit streams can strongly impact the output accuracy, known as the correlation problem. For example, an AND gate can only accurately compute the product of two input streams when they are not correlated; that is, the ratio of paired 1s/0s and unpaired 1s/0s across the two streams is set according to input values. Otherwise, if there are more paired 1s than expectation, it ends up computing the minimum function [67]. To alleviate the impact of correlation, researchers have proposed to 1) use costly RNGs to suppress correlation [35], 2) control correlation [32] or 3) leverage correlation when computing [4, 9, 67].

One proposal to circumvent this problem is isolating stochastic computing from correlation with a *deterministic* approach [19], where two input bit streams are expanded via different styles of repetition in order to mimic the convolution of two bit streams. Because every bit of one bit stream interacts with every bit of the other, the computation no longer depends on the distribution of 1s. While this method enables the use of typical stochastic computing units with high accuracy, it suffers from long computation latency: 2^{2n} cycles for inputs of two n-bit binary numbers, compared to 2^n cycles in typical stochastic computing. Follow-on work of this determinstic approach [42]

TABLE I
FUNCTIONAL UNITS AVAILABLE IN UNARY COMPUTING.

Function	Rate coding	Temporal coding
Add	[14, 16, 19, 25], uGEMM	uGEMM
Multiply	[14, 16, 19, 57], uGEMM	uGEMM
Add constant		[39, 63]
Minimum	[4, 32]	[38, 63]
Maximum	[4, 32]	[38, 63]
Inhibit		[59, 63]

reduces the long latency back to 2^n cycles by scrambling the bit stream, at the expense of degraded accuracy.

B. Temporal Coding-Based Unary Computing

Temporal coding is a coding scheme where the timing relation of events contains information. Unary computing using temporal-coded data has been used in various applications such as DNA sequence alignment [38], decision trees [63], and median filter/sorting networks [43, 44].

a) Data representation: Temporal coding encodes data into the timing of a signal's transition (or edge), with the bit stream as a chain of 1s followed by a chain of 0s. The value can be embedded into the generation of the bit stream [43, 44], or into the path a signal takes, i.e., a longer path will delay the edge observed at the downstream computing unit for longer [38], alternatively referred to as race logic.

b) Temporal computing: Unlike conventional stochastic computing [14], temporal computing performs computations deterministically and bit streams can be generated without RNGs. Table I lists all existing functional units available in temporal computing. For example, the minimum function can be performed using a simple AND gate by detecting the first arriving edge (i.e., less delay). Similarly, the maximum function is implemented by an OR gate to detect the most delayed edge. Addition with a constant operand can be performed by adding delay units [39, 63]. Furthermore, race logic can imitate the behavior of winner-take-all inhibition of a multi-input neuron, which only passes the first incoming edge [59].

c) Limitation — lack of essential functional units: Prior to our work, some essential functional units, such as adders and multipliers, have never been proposed for temporal computing. In order to fully exploit the benefit of temporal computing, an area- and energy-efficient implementation of addition and multiplication is necessary.

C. Challenge of a Unified Unary Architecture

Though both rate coding and temporal coding are fundamental forms in unary computing system, the wide range of their existing implementations are highly disparate in a couple aspects. First, there is an inconsistent level of accuracy among rate-coded and temporal-coded implementations, depending on the operations performed. Second, current implementations are specialized and would require expensive regeneration of bit streams in order to support operations that are best with rate coding along with those that are best with temporal coding in the same architecture. This motivates an important research question for unary computing – can we build *unified* unary designs that achieve high accuracy, low hardware complexity and short latency for both representations simultaneously?

D. General Matrix Multiplication in DNNs

GEMM is part of the Basic Linear Algebra Subprograms (BLAS) library that performs matrix multiplication:

$$O = \alpha AB + \beta C, \tag{1}$$

where A, B and C are input matrices and α and β are scaling factors. In DNNs, the most computation-intensive layers are fully connected layers and convolution layers [27]. Fully connected layers are naturally suitable for GEMM, while convolution layers require transformation before GEMM computation [12, 13, 18], supported in popular software deep learning frameworks [1, 21, 51]. However, as evaluated in [20], 95% of GPU runtime and 89% of CPU runtime are spent on software-configured GEMM operations for fully connected layers and convolution layers, and therefore it is critical to reduce its energy consumption by using dedicated energy-efficient GEMM hardware [26, 53, 56, 57].

III. A Systematic View of Unary Bit Streams

In this section, we present two metrics for bit streams that characterize how their distributions of 1s and 0s affect the quality of unary computing results. The first is the *correlation* between two bit streams, which measures how aligned (or dependent) two bit streams are. The second is the *stability* of a bit stream, which measures how fast a bit stream stabilizes. These two metrics are applicable for any style of unary bit streams and enables systematic analysis of arbitrary unary computing architectures.

a) Correlation: The first metric, *correlation*, is the cross correlation [4] between two unary bit streams X and Y of length L, defined as:

$$\mathrm{corr}(X,Y) = \begin{cases} \dfrac{ad-bc}{L\times\min(a+b,a+c)-(a+b)(a+c)}, & \text{if } ad > bc \\ \dfrac{ad-bc}{(a+b)(a+c)-L\times\max(a-d,0)}, & \text{otherwise} \end{cases} \tag{2}$$

Here, a is the number of bit pairs where $\{X_i, Y_i\} = \{1, 1\}$ (i denotes the bit position in a stream), b is number of pairs where $\{X_i, Y_i\} = \{1, 0\}$, c is number of pairs where $\{X_i, Y_i\} = \{0, 1\}$, and d is number of pairs where $\{X_i, Y_i\} = \{0, 0\}$. In a nutshell, the correlation between two bit streams reflects the extent of overlapping 0s and 1s in the bit streams and hints at the achievable accuracy of an operation based on these input bit streams [4]. Correlation is bound to the range $[-1, +1]$.

b) Stability: Rate-coded bit streams inherently have the progressive precision property [5] where an increase in the stream length leads to an increase in accuracy. A partial output bit stream still represents the same value as the full bit stream but with low accuracy, i.e., an early-terminated bit stream is a low-accuracy version of the full bit stream. Early termination, a common concept in approximate computing [8, 15, 54, 65],

TABLE II
DISTRIBUTION METRICS IN UNARY COMPUTING.

Metric		Rate coding	Temporal coding
Correlation	Optimal	0	+1
	Actual	[-1,+1]	+1
Stability	Optimal	1	Min
	Actual	(0,1]	Min

is essential for time- and energy-constrained devices. *Stability* is a new metric we propose in this work to measure how early a bit stream's progressive precision [5] stabilizes/converges.

For a bit stream of length L, V_L represents its final value and V_l represents the progressive precision value of the partial bit stream based on the first l bits ($l \leq L$). If starting from the l-th bit, the bias $\Delta V_l = |V_l - V_L|$ is consistently smaller than a user-defined threshold V_{THD}, then stability is calculated as:

$$\text{Stability} = 1 - \frac{\max\{l|\Delta V_{l-1} > V_{THD}\}}{L} \tag{3}$$

Stability ranges between 0 and 1, with a higher value indicating an earlier progressive precision convergence, implying better adaptiveness to early termination. This metric can be applied to temporal-coded bit streams if we consider the bit streams within a certain time range as Bernoulli sequences in stochastic computing, but will generally yield low stability as temporal-coded data lacks the progressive precision property.

c) Challenge — distribution-sensitiveness: Table II lists the optimal and realistic value ranges for correlation and stability in different coding schemes. Having the optimal input correlation leads to higher final accuracy, while having the optimal stability indicates a higher potential for lowering computation latency with minimal accuracy loss. Current stochastic computing schemes provide no guarantees for either metric, with the exception of the deterministic approach of stochastic computing that guarantees zero correlation at the cost of long latency. In temporal computing, because bit streams always start with a chain of 1s followed by a chain of 0s, the bit stream cannot stabilize until all 1s have been observed. Both correlation and stability are important metrics not only for the input bit streams of unary computing units, but also for the output bit streams of these computing units. This is especially the case for a hierarchical system such as GEMM where an output bit stream can be directly used as the input stream of other components.

IV. uGEMM ARCHITECTURE

In this section, we present our uGEMM architecture and novel linear functional unit designs: uMUL, uSADD, and uNSADD. uGEMM overcomes the challenges of existing unary approaches in Section II. uGEMM features high parallelism, input insensitivity, and early termination, owing to our novel functional units, which outstrip prior designs because:

- They are free of the correlation problem and achieve high accuracy with rate-coded input.
- They are distribution-insensitive, seamlessly accepting inputs represented in either rate coding or temporal coding without costly data conversion.
- They produce highly stable and accurate output bit streams, without trading off latency, for early termination.

We will first introduce uMUL, uSADD, and uNSADD, followed by the integrated uGEMM architecture. For each functional unit design, we theorize its behavior then map the mathematical formulation onto the microarchitecture in Figure 3, accompanied by a walkthrough example in Table III.

A. uMUL: Unary Multiplication

Analyzing the multiplication of two values, we reorganize the probability equation according to the effective operations each logic gate performs, from which we obtain the most concise and practical unipolar/bipolar unary multiplier that 1) produces accurate results without excessive RNGs and 2) is insensitive to input distribution.

1) Mathematical Expression

a) Unipolar: For multiplying two unipolar streams, the theoretical output value, V_{out}, is expressed as the product of two input values, $V_{in,0}$ and $V_{in,1}$:

$$\begin{aligned} V_{out} &= V_{in,0} \cdot V_{in,1}, \\ P(S_{out} = 1) &= P(S_{in,0} = 1) \cdot P(S_{in,1} = 1), \end{aligned} \tag{4}$$

where S_* refers to the input or output bit stream. Gaines uses an AND gate to approximate the desired probability in Eq. 4 [14]. However, this AND-gate multiplier actually implements the joint probability in Eq. 5 instead. This joint probability only matches Eq. 4 perfectly when two input bit streams are independent (i.e., have zero correlation), which is not ensured using conventional bit stream generation.

$$\begin{aligned} P(S_{out} = 1) &= P(S_{in,0} = 1, S_{in,1} = 1) \\ &= P(S_{in,0} = 1) \cdot P(S_{in,1} = 1 | S_{in,0} = 1) \end{aligned} \tag{5}$$

To accurately implement Eq. 4 yet still benefit from a simple AND-gate design, we embed the requirement of zero correlation into uMUL's bit stream generation by enforcing Eq. 6,

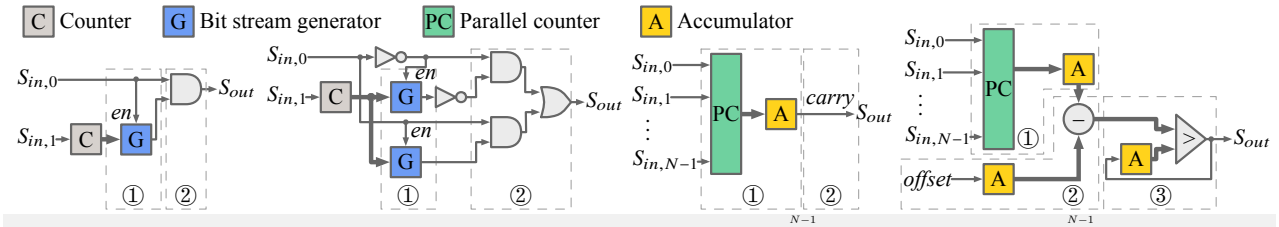
$$P(S_{in,1} = 1 | S_{in,0} = 1) == P(S_{in,1} = 1), \tag{6}$$

Since the AND-gate multiplication only cares about the input of bit stream $S_{in,1}$ when $S_{in,0} = 1$, we define

$$P\left(S_{in,1}^{eff} = 1\right) = P(S_{in,1} = 1 | S_{in,0} = 1), \tag{7}$$

where $S_{in,1}^{eff}$ is the effective bits for AND gate (i.e., when $S_{in,0} = 1$), with which we regulate the following discussion for implementation.

Eq. 6 gives us insight into how to build an accurate multiplier. Recalling the notation in Eq. 2, let a, b, c and d represent the counts of $(1,1)$, $(1,0)$, $(0,1)$ and $(0,0)$ pairs for two input bit streams $(S_{in,0}, S_{in,1})$. The marginal probability is $P(S_{in,1} = 1) = (a + c)/(a + b + c + d)$ and the conditional probability is $P(S_{in,1} = 1 | S_{in,0} = 1) = a/(a+b)$. Eq. 2 states that if correlation is zero, then $ad = bc$. Plugging this in, we get $(a+c)/(a+b+c+d) = a/(a+b)$; thus, zero correlation implies that the conditional probability in uMUL equals the marginal probability. Therefore, the output of uMUL is accurate.



① Conditional bit stream generation (Eq. 6)
② ANDMUL (Eq. 5)

① Conditional bit stream generation (Eq. 10)
② Decomposed XNORMUL (Eq. 9)

① Accumulation $\sum_{n=0}^{N-1} \mathbb{C}_{in,n}$ (in Eq. 12)
② Average by N (in Eq. 12)

① Accumulation $\sum_{n=0}^{N-1} \mathbb{C}_{in,n}$ (in Eq. 17)
② Offset $\mathbb{C}_{offset}$ subtraction (in Eq. 17)
③ MIN operation (in Eq. 17)

(a) Unipolar uMUL (b) Bipolar uMUL (c) N-input uSADD (d) N-input uNSADD

Fig. 3. uGEMM functional units. Thick line: binary signal; and thin line: unary signal.

b) Bipolar: Similar to Eq. 4 and 5 for unipolar, we derive the expression for bipolar data representation in Eq. 8 and 9:

$$\begin{aligned} V_{out} &= V_{in,0} \cdot V_{in,1} \\ 2 \cdot P(S_{out}=1) - 1 &= \big(2 \cdot P(S_{in,0}=1) - 1\big) \cdot \big(2 \cdot P(S_{in,1}=1) - 1\big) \\ P(S_{out}=1) &= P(S_{in,0}=1) \cdot P(S_{in,1}=1) + \\ &\quad P(S_{in,0}=0) \cdot P(S_{in,1}=0) \end{aligned} \tag{8}$$

$$\begin{aligned} P(S_{out}=1) &= P(S_{in,0}=1, S_{in,1}=1) + P(S_{in,0}=0, S_{in,1}=0) \\ &= P(S_{in,0}=1) \cdot P(S_{in,1}=1 | S_{in,0}=1) + \\ &\quad P(S_{in,0}=0) \cdot P(S_{in,1}=0 | S_{in,0}=0) \end{aligned} \tag{9}$$

Following the same reasoning as for Eq. 6, accurate multiplication can be achieved by forcing both effective bits as

$$\begin{aligned} P(S_{in,1}=1 | S_{in,0}=1) &== P(S_{in,1}=1) \\ P(S_{in,1}=0 | S_{in,0}=0) &== P(S_{in,1}=0) \end{aligned} \tag{10}$$

2) Implementation

a) Unipolar: As shown in Figure 3(a), the circuit construction based on Eq. 6 to attain the distribution-insensitive multiplier is straightforward. First, the counter is responsible for holding the probability value for input $S_{in,1}$. This data can be either streamed as serial bits or prestored as a binary value into the counter, both supporting fully streaming dataflow. The former works in an in-stream manner, thus denoted as uMUL-IS. The latter, which prestores data statically, named uMUL-ST, is well-suited for neural networks with static weights. A bit stream generator based on Sobol sequence [35] is then adopted to enforce that the desired probability is generated during $S_{in,0}=1$, which enables the conditional bit stream generation of $S^{eff}_{in,1}$. The generated bit is logic one if the Sobol sequence number is smaller than the probability; otherwise it is logic zero. At the end, an AND gate is adopted to multiply $S_{in,0}$ and $S^{eff}_{in,1}$.

b) Bipolar: For the bipolar architecture in Figure 3(b), we extend the unipolar circuit so that it behaves as an XNOR (decomposed in the figure). The bottom half of Figure 3(b) generates the effective bits when $S_{in,0}=1$, as in the unipolar circuit, producing $P(S_{in,0}=1) \cdot P(S_{in,1}=1|S_{in,0}=1)$. The top half takes the inverse and produces $P(S_{in,0}=0) \cdot P(S_{in,1}=0|S_{in,0}=0)$. The partial results are sent through an OR gate to attain the final bit stream. Note that as uMUL is now free from the correlation problem, it is feasible to share bit stream generators in uGEMM with no additional effort [70].

TABLE III
FUNCTIONAL EXAMPLE OF UGEMM UNITS (UNIPOLAR). VALUES ARE IN DECIMAL.

Cyc	Current Value																		
	$S_{in,0}$	$S_{in,1}$	C	G	$\&_{in,0}$	$\&_{in,1}$	S_{out}	$S_{in,0}$	$S_{in,1}$	$S_{in,2}$	$S_{in,3}$	PC_{out}	A	S_{out}	PC_{out}	$>_{in,0}$	$>_{in,1}$	S_{out}
0	1	-	2	1	1	1	1	1	1	1	1	4	0	1	4	4	0	1
1	0	-	2	3	0	0	0	1	0	0	0	1	1	0	1	5	1	1
2	0	-	2	3	0	0	0	1	1	0	0	2	3	0	2	7	2	1
3	1	-	2	3	1	0	0	0	0	0	1	1	0	1	1	8	3	1
	V_{out}		1/4					V_{out}				2/4			4/4			
	V_{exp}		1/4=(2/4)×(2/4)					V_{exp}				2/4=(3+2+1+2)/(4×4)			4/4=MIN{4, (3+2+1+2)}/4			
	MUL		uMUL					**ADD**				uSADD			uNSADD			

S_{in} and S_{out} denote the bitstream input and output; C, G, A represent counter, bit stream generator, and accumulator; PC, &, and > are parallel counter, accumulator, and comparator. Note that here in uMUL-ST, the 1-count of $S_{in,1}$ is prestored in C.

As uMUL exclusively applies conditional probability to generate those effective bits in $S_{in,1}$, the distribution in the output bit stream actually depends on the distribution in the input bit stream $S_{in,0}$. In other words, if the input to uMUL is generated by conventional stochastic computing units, the output of uMUL can be consumed by subsequent conventional stochastic computing units seamlessly with no additional error. In Table III's example, we show that unipolar uMUL's output value V_{out} exactly matches the expected value V_{exp} thanks to the conditional bit stream generation at cycles 0, which then stops updating the generation if observing $S_{in,0}=0$, for the AND gate.

B. uSADD: Unary Scaled Addition

For an N-input unary scaled addition, we derive a unified expression for both unipolar and bipolar representations based on probabilities and describe the cycle-level operation.

1) Mathematical Expression

The goal of a scaled ADD is to calculate the mean value V_{out} among N inputs $V_{in,n}$ as in Eq. 11, where $\mathbb{C}$ denotes the number of 1s in the bit stream, m is 1 for unipolar and 2 for bipolar, k is 0 for unipolar and 1 for bipolar. The same notation holds for all equations below. Then we divide the total count of both input stream $S_{in,n}$ and output stream S_{out} by length L for unipolar data, or scale it by $\frac{2}{L}$ then subtract by 1 for bipolar data, according to data representation. To stay within

the value range, the added inputs are scaled by N to obtain the final output value. Despite their differences, both unipolar and bipolar representations reduce to the same simplified operation of counting the number of 1s as in Eq. 12.

$$\begin{aligned} V_{out} &= \frac{1}{N} \cdot \sum_{n=0}^{N-1} V_{in,n} \\ \frac{m \cdot \mathbb{C}_{out}}{L} - k &= \frac{\sum_{n=0}^{N-1} \left(\frac{m \cdot \mathbb{C}_{in,n}}{L} - k \right)}{N} \end{aligned} \tag{11}$$

$$\mathbb{C}_{out} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} \mathbb{C}_{in,n} \tag{12}$$

2) Implementation

The architecture of uSADD for unipolar/bipolar bit streams is shown in Figure 3(c). First, N input streams are fed into a N-bit parallel counter [49], producing a count every cycle. This is then accumulated, and the accumulator asserts its carry signal whenever it overflows, i.e., exceeds N. This signal is returned as the output of uSADD, effectively normalizing the sum of input 1s by N across all input bit streams.

As for the output, uSADD outputs bit streams according to the inputs. Therefore, like uMUL, if uSADD directly takes inputs from conventional stochastic computing units, the output can also be fed to conventional stochastic computing units without extra error. As shown in the example in Table III, unipolar uSADD generates, taking these four unipolar inputs, scaled results correctly via its accumulator, only emitting a one when the accumulator reaches four.

3) Proof of Error Bound

This section proves that regardless of rate or temporal coding, the maximum error for uSADD is $\frac{N-1}{L \cdot N}$ for unipolar and $\frac{2 \cdot (N-1)}{L \cdot N}$ for bipolar, which is negligible for large bit streams.

Eq. 13 first shows the error $\Delta\mathbb{C}$ between uSADD's input count of 1s ($\mathbb{C}_{in}$) and the actual output count $\widetilde{\mathbb{C}}_{out}$. The takeaway is that the error is always between $[0, N-1]$; this error is the leftover value of the accumulator at the end of the bit streams.

$$\begin{aligned} \Delta\mathbb{C} &= \sum_{n=0}^{N-1} \mathbb{C}_{in,n} - N \cdot \widetilde{\mathbb{C}}_{out} \\ &= \left(\sum_{n=0}^{N-1} \mathbb{C}_{in,n} - N \cdot \widetilde{\mathbb{C}}_{out} \right) \bmod(N) \in [0, N-1] \end{aligned} \tag{13}$$

Eq. 14 then shows the error ΔV_{out} between uSADD's ideal output value V_{out} and the actual output value $\widetilde{V}_{out}$.

$$\Delta V_{out} = V_{out} - \widetilde{V}_{out} = \frac{m}{L \cdot N} \cdot \left(\sum_{n=0}^{N-1} \mathbb{C}_{in,n} - N \cdot \widetilde{\mathbb{C}}_{out} \right) \tag{14}$$

Combining Eq. 13 and 14, Eq. 15 shows that the error bound of uSADD for both unipolar and bipolar.

$$\Delta V_{out} = V_{out} - \widetilde{V}_{out} = \frac{m}{L \cdot N} \cdot \Delta\mathbb{C} \in \left[0, \frac{m \cdot (N-1)}{L \cdot N} \right] \tag{15}$$

C. uNSADD: Unary Non-Scaled Addition

Besides the scaled addition [3, 45], many application scenarios also require identical scale between the inputs and outputs [53, 57]. However, conventional OR-gate based non-scaled addition is inaccurate even with special up-scaling logic or bit stream regeneration [53, 67], and is not practical for realistic application. We present uNSADD to perform non-scaled addition with better accuracy. Our uNSADD significantly improves the accuracy of unipolar non-scaled addition, and it is the first design to support bipolar non-scaled addition.

1) Mathematical Expression

We formulate the behavior of uNSADD in Eq. 16, 17 and 18. If overflowing outside $[0, 1]$ in unipolar mode and $[-1, 1]$ in bipolar mode, the output value of uNSADD, V_{out}, is always clipped to $[-k, 1]$ as in Eq. 16, where m is 1 for unipolar and 2 for bipolar, k is 0 for unipolar and 1 for bipolar. In terms of output bit count, which is always non-negative, both modes have the same format as in Eq. 17 with the offset value in Eq. 18. We clip the bit count to the maximum value of output length L when the sum overflows. Furthermore, the result can be simplified as a minimum function due to the fact that 0 is the lower bound of clipping.

$$\begin{aligned} V_{out} &= \text{clip}\left(\sum_{n=0}^{N-1} V_{in,n}, -k, 1 \right) \\ m \cdot \frac{\mathbb{C}_{out}}{L} - k &= \text{clip}\left(\sum_{n=0}^{N-1} \left(m \cdot \frac{\mathbb{C}_{in,n}}{L} - k \right), -k, 1 \right) \end{aligned} \tag{16}$$

Therefore we have the unified form,

$$\begin{aligned} \mathbb{C}_{out} &= \text{clip}\left(\sum_{n=0}^{N-1} \mathbb{C}_{in,n} - L \cdot \frac{k}{m} \cdot (N-1), 0, L \cdot \frac{k+1}{m} \right) \\ &= \min\left(\sum_{n=0}^{N-1} \mathbb{C}_{in,n} - L \cdot \mathbb{C}_{offset}, L \right) \end{aligned} \tag{17}$$

where,

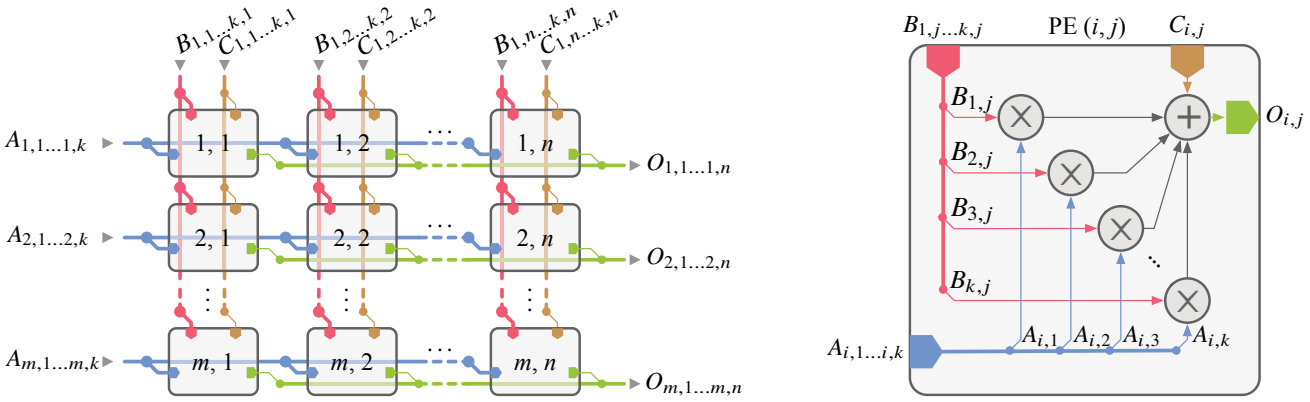
$$\mathbb{C}_{offset} = \frac{k}{m} \cdot (N-1) \tag{18}$$

2) Implementation

Figure 3(d) shows the architecture of uNSADD for unipolar/bipolar bit streams. Likewise, uNSADD adopts a parallel counter to count the bits at each cycle then stores it in the accumulator. Then, the accumulated offset is subtracted from the accumulated input bit count at this cycle, resulting the current anticipated output bit count. When this bit count is greater than the historical bit count, the output is 1; otherwise 0. Table III shows unipolar uNSADD's operation. The output is strictly forced by the difference between the anticipated ($>_{in,0}$) and historical ($>_{in,1}$) output bit counts, with larger anticipated bit count leading to a one as output.

D. Integrated uGEMM Architecture

We now describe how our novel functional unit designs are integrated together to form a holistic architecture for GEMM applications. Without loss of generality, we assume $\alpha = 1$ and



(a) uGEMM architecture (b) uGEMM PE internals

Fig. 4. uGEMM architecture and its PE. Thick line: multi-bit stream; and thin line: single-bit stream.

$\beta = 1$ in Eq. 1. We assume that matrices O, A, B and C are of the size $(m \times n)$, $(m \times k)$, $(k \times n)$ and $(m \times n)$, respectively. Figure 4(a) shows the architecture of uGEMM. Input A, B, and C go through the m-by-n processing element (PE) array in the center of the figure. Each row in the array has n PEs, all fed by the same row of A (each with k elements), simultaneously without buffering. Each column in array has m PEs, all input by the same column of B (each with k elements), at the same time without buffering, too. Each PE in the array takes only one element from C. As shown in Figure 4(b), the $(i, j)_{th}$ PE performs the multiply-accumulate (MAC) of the i_{th} row from A and the j_{th} column from B, with k elements for each row and column, then adds the $(i, j)_{th}$ element from C as the result. The following highlights key advantageous features of the uGEMM architecture.

a) Highly parallel PE array: Different from traditional bit-parallel binary architectures that suffer from wire (interconnect) congestion and high power overhead [11, 17, 22], uGEMM's unary computing units are designed with extremely simple logic for low wire congestion, thus high parallelism. At each cycle, the same input bit stream is observed simultaneously by all corresponding PEs, and the output of k-by-k MAC operation is produced in parallel from every PE. Note that if multiple PEs take in the same input operand, the bit stream generators (shown in Figure 3(a, b)) for the other non-shared operands can be shared, saving the area and power costs of additional generators. Compared to binary systems, the highly parallel architecture of uGEMM minimizes data scheduling, resulting in less data movement between the MAC array and the memory, thus reducing energy consumed on memory access [10, 11, 69].

b) Accurate input-insensitive uGEMM PE: As shown before, we re-innovate designs of multiplier (MUL) and adder (ADD) in stochastic computing to overcome the input correlation problem and achieve high accuracy without the latency penalty imposed by deterministic approach. This is done by revisiting the mathematical fundamentals of unary computing. In addition to correlation insensitivity, uGEMM is coding-insensitive, compatible with both rate-coded and temporal-coded data. Being capable of tolerating the different underlying distributions imposed by the two coding methods with minimal bit stream manipulation, the entire uGEMM can use only one type of RNG, which is not possible in prior works.

c) Latency reduction by reliable early termination: Recall that the output is sent out from each PE one bit at a time in a streaming manner, and one of the drawbacks in previous unary architectures is the extended latency in generating the bit-serial output. Thanks to uGEMM's functional unit design, the high-quality output bit stream ensures fast convergence to the anticipated value, providing an opportunity for early terminating uGEMM's output bit streams when it is precise enough, which can lead to large reductions in computation latency while still maintaining accurate results. Moreover, uGEMM is capable of performing fully in-stream processing without the delay of binary-unary interconversion adopted in previous unary computing architectures [57].

With these carefully designed functional units, uGEMM has higher accuracy than those in conventional stochastic computing and also supports temporal-coded data. The next section systematically evaluates uGEMM in terms of accuracy, stability and hardware efficiency.

V. Evaluation and Results

In this section, based on the two metrics in Section III, we follow a bottom-up approach to systematically analyze uGEMM performance from unit level to architecture level.

A. Evaluation Objectives

The following questions will be answered by comparing uGEMM and its components against proper baselines regarding the evaluation metrics proposed.

a) Q1: How much do uGEMM's units improve over other stochastic MULs and ADDs in output accuracy and stability?

- Accuracy: how close are the outputs of the proposed units to binary references, measured by root mean square error.
- Output/Input stability ratio (O/I stability): this metric, calculated by dividing the stability of the output stream by that of the input, is used for quantifying the effect

of one PE. Note that we fix the stabilization threshold $V_{THD} = 0.05$ and saturate the ratio at a maximum of one if the unit maintains or improves the bit stream stability.

For each component, we vary these factors:

- Input resolution: this factor refers to the correspondent binary bitwidth before conversion, to which the length of input unary bit streams is quadratic.
- Input coding type: how will the input representation influence the accuracy and stability of the output? Recall the characteristics of two aforementioned representations:
  - Ideal rate-coded input (RC input): low correlation and high stability
  - Ideal temporal-coded input (TC input): high correlation and low stability.

  Above types are representative of streams with extreme characteristics for typical stochastic ADD and MUL units and provide a bound of their accuracy and stability.

In addition, we qualitatively discuss the robustness of uGEMM functional units to bit flipping errors.

b) Q2: How efficient is the hardware implementation of individual uGEMM components? We compare the hardware implementation efficiency in terms of area, power, latency and energy against previous state-of-the-arts.

c) Q3: How well does uGEMM, under different configurations, perform compared to other unary GEMMs? At the architecture level, we consider these two targets:

- Accuracy: what is the final accuracy of uGEMM and how the error propagates through functional units?
- Support for early termination: by looking at progressive precision [5] and stability, we evaluate how well each unary GEMM design supports early termination.

We investigate the hardware cost of area, power, latency and energy, which are affected by utilization and sharing of hardware resources. Therefore, we implement different configurations of GEMM for various combinations of data polarities (unipolar/bipolar) and operation requirements (e.g., scaled/non-scaled addition).

B. Evaluation of uGEMM Functional Units

With an in-house C++-based cycle-accurate unary simulator, we simulate our uGEMM functional units, as well as prior stochastic MUL and ADD implementations. For bit stream generation, we use Sobol sequence generators, which have shown superior efficiency for stochastic computing [35]. To mitigate the randomness in bit stream generation [35], the results are averaged across millions of random trials, as different input values and different Sobol sequence generators vary the output accuracy and stability.

1) uMUL

a) Setup: We compare uMUL, including its static and in-stream implementations (uMUL-ST and uMUL-IS), with previous ANDMUL/XNORMUL [14] and more recent NMUL by Najafi et al. [42] for stochastic multiplication. uMUL-ST perfectly ensures $P\left(S^{eff}_{in,1} = 1\right)$ equal to the expected probability of input 1 by preloading one multiplicand, e.g. static weights

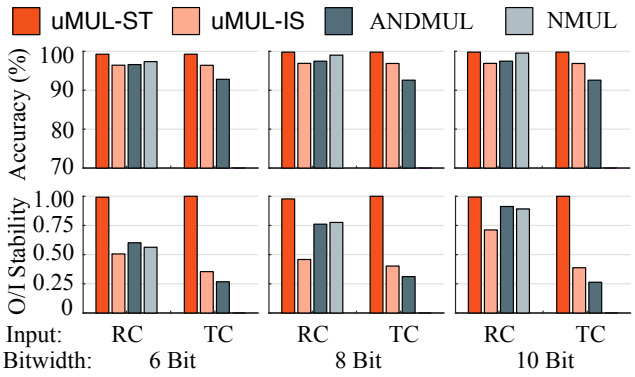


Fig. 5. Unipolar uMUL accuracy and O/I stability.

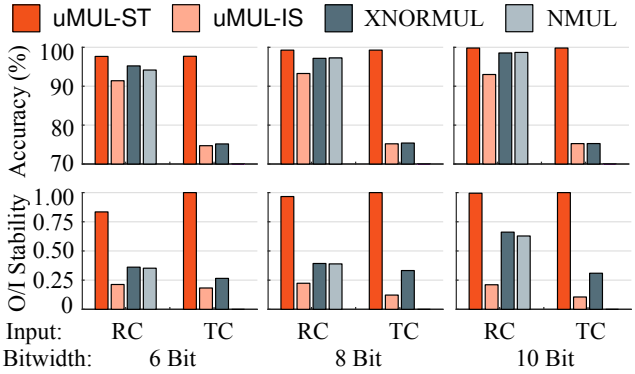


Fig. 6. Bipolar uMUL accuracy and O/I stability.

in DNNs, into the counter. uMUL-IS receives both streaming inputs and computes simultaneously. For each module, we report the accuracy using unary input streams generated from 6-, 8-, and 10-bit binary data respectively, and results from unipolar and bipolar input data are shown in Figure 5 and 6.

b) Results: For unipolar input, uMUL-ST has consistently superior output accuracy and O/I stability ratio among all, even better than ANDMUL with rate-coded input of almost zero correlation for each bit width. Input coding poses negligible impact on uMUL-ST's accuracy and stability as expected from the derivation in Section IV-A. With n-bit data, NMUL in [42] truncates the computation from 2^{2n} in [19] back to 2^n by scrambling bit streams. This induces incompatibility with temporal coding, reintroducing of the correlation problem, too. As such, NMUL asymptotes uMUL in terms of accuracy and stability in most cases but never surpasses uMUL. Though uMUL-IS slightly weakens the guarantee of probability integrity, its overall accuracy is comparable to the ideal of ANDMUL and more robust to temporal-coded input with high correlation and low stability. Unlike unipolar ANDMUL, uMUL-ST does not degrade the stability, while uMUL-IS is on the comparable level as ANDMUL.

Although the accuracy and O/I stability ratio drop in the bipolar implementation due to lower data resolution, similar findings are discovered that uMUL-ST shows robust output accuracy and stability for all input compared to XNORMUL, and bipolar uMUL-IS still achieves acceptable and stable results.

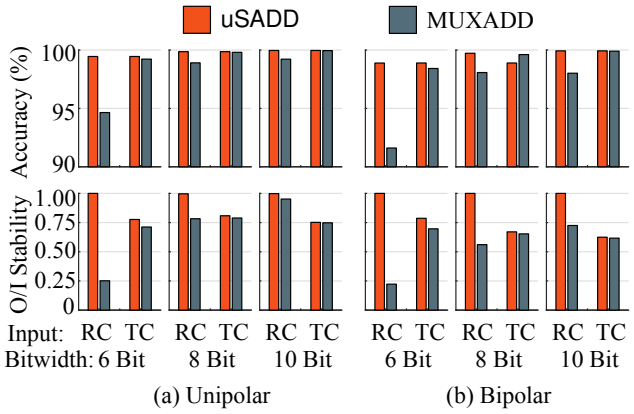


Fig. 7. uSADD accuracy and O/I stability.

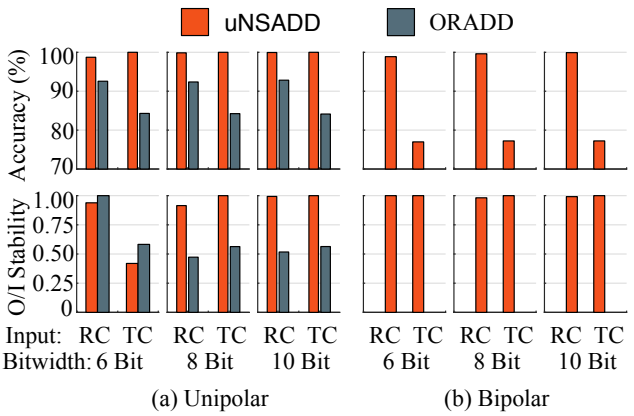


Fig. 8. uNSADD accuracy and O/I stability.

2) uSADD

a) Setup: We evaluate the uSADD against MUXADD for stochastic addition in Section II-A. The same setup for bit width and input representations is used here as before, while we fix the number of ADD channel to two in the evaluation of individual functional units.

b) Results: Figures 7(a) and 7(b) shows the uSADD output accuracy and O/I stability ratio for unipolar and bipolar format respectively. uSADD has almost perfect accuracy regardless of input types unlike the MUXADD whose accuracy is dependent on input types since uSADD has no stochastic selection signal to introduce correlation problem in MUXADD. Also, it helps uSADD better retain bit stream stability. For MUXADD, the final accuracy for temporal-coded input is higher than rate-coded input because when input bit streams are maximally correlated, the influence of selection signal is negligible. Larger input bit widths contribute to performance, too. For bipolar, we observe the same trend as that of unipolar, but the accuracy and the ability of maintaining stability reduce proportionally because of lower input resolution.

3) uNSADD

a) Setup: ORADD in Section II-A is the baseline for uNSADD's unipolar non-scaled stochastic addition. Meanwhile, uNSADD is the first to incorporate bipolar non-scaled stochastic addition by setting the offset. Effects of input type, bit width, and input channel are still investigated.

b) Results: Figure 8(a) shows the unipolar uNSADD's performance, whose accuracy is almost perfect for both input representations compared to ORADD, whose accuracy degrades greatly with temporal-coded input. uNSADD retains better stability than ORADD, too. Results also indicate that uNSADD is stable with different bit widths, while longer input streams contribute to some improvement.

From Figure 8(b), we conclude that the design of uNSADD is compatible with bipolar data using rate-coded input data, too, whose accuracy rates above 90% in all cases and reaches almost 100% with 10-bit inputs. The stability retainment is satisfactory as well with rate-coded input. Though, facing temporal-coded inputs, the accuracy and stability retainment become less reliable due to less stable input bit streams, the accuracy still shoots almost to 80%. Nevertheless, this worst case scenario does not apply to uNSADD in uGEMM since the bipolar uNSADD accepts input of better quality from uMUL's output instead of raw highly unstable temporal-coded input.

4) Robustness Analysis

uGEMM functional units leverage register-based logic, like RNGs in uMUL and counters in uSADD and uNSADD, to ensure their precise behavior, which expose higher vulnerability to hardware errors. Though the robustness might decrease on the microarchitecture level, the system-level robustness still maintains similar to prior unary systems, as prior unary systems require significant amounts of, even more, correlation manipulating logic [32] to ensure high accuracy.

5) Hardware Implementation

a) Setup: The hardware for unipolar/bipolar multiplier, scaled adder, non-scaled adder of 8-bit data precision is synthesized using Synopsys Design Compiler with TSMC 45 nm technology and a clock frequency of 400 MHz.

The results are listed in Table IV. including uGEMM's uMUL, uSADD, and uNSADD. Note that all multipliers have two inputs and adders have eight inputs. We first compare against the well accepted classic stochastic design by Gaines [14], which supports all operations except the bipolar non-scaled addition. Sim et al. propose an improved multiplier that trades off accuracy for energy efficiency [57]. The deterministic approach by Jenson and Riedel [19], which repeats stochastic bit streams with counters for accuracy, is applied to existing stochastic functional units. We also consider a latency-optimized deterministic approach by Najafi et al. [42], where counters are substituted with linear-feedback shift registers (LFSRs). We implement the rotation method for both deterministic approaches as described in [19, 42]. We implement all bit stream generators (excluding the counters in Jenson's and LFSRs in Najafi's) as Sobol sequence generators [35].

b) Results: The unipolar uMUL's synthesis results indicate that it has better area, power, and energy efficiency compared to all its counterparts. Note that Jenson's deterministic approach trades off latency for precision via repeating bit streams. For the scaled addition, uSADD only introduces minor area and power overhead compared to Gaines' stochastic

TABLE IV
HARDWARE COST COMPARISON OF UNARY FUNCTIONAL UNITS.

Unit	Polarity	Area (μm^2)	Power (μW)	Latency (ns)	Energy (pJ)
Multiplier					
uMUL	Unipolar	207.6	60.9	640	39.0
	Bipolar	364.3	99.8	640	63.9
Gaines	Unipolar	378.7	122.1	640	78.1
	Bipolar	377.0	128.2	640	82.0
Sim	Unipolar	278.0	80.7	640	51.6
	Bipolar	278.7	83.0	640	53.1
Jenson	Unipolar	520.2	131.6	163840	21561.3
	Bipolar	526.2	137.9	163840	22593.5
Najafi	Unipolar	521.4	139.1	640	89.0
	Bipolar	514.0	141.4	640	90.5
Scaled adder					
uSADD	Both	60.5	32.3	640	20.7
Gaines	Both	55.7	29.1	640	18.6
Jenson	Both	84.5	30.1	163840	4931.6
Najafi	Both	86.1	29.0	640	18.6
Non-scaled adder					
uNSADD	Unipolar	181.7	65.0	640	41.6
	Bipolar	196.5	69.7	640	44.6
Gaines	Unipolar	11.8	5.1	640	3.3

units with a large improvement in terms of accuracy observed from previous results. The unipolar scaled adder introduced in Jenson's work is also of low area and power overhead but higher latency. uNSADD is the first to support bipolar non-scaled addition. This universal and accurate design trades off some hardware efficiency for high adaptiveness and accuracy according to our output evaluation before.

Overall, the implementations of uGEMM's individual components are proven hardware-efficient. Furthermore, we will investigate its higher level efficiency in GEMM related to the types of operations, resource sharing, etc.

C. Evaluation of uGEMM Architecture

On the GEMM level, we consider final accuracy, fitness to early termination, and hardware cost. uGEMM's target is preserving the area and power efficiency as previous unary architecture but reducing its error, latency, as well as providing compatibility for distinct input codings. Based on different uGEMM functional units, we evaluate all the combinations for building an 8-bit fully parallel uGEMM. We opt to employ uMUL-ST in uGEMM for a fair comparison against the baseline GEMMs where inputs are generated with static values. We fixed the GEMM shape to $m = k = n = 16$ for evaluation purpose. Same as our functional unit evaluation, we again evaluate Gaines', Sim's, Jenson's and Najafi's unary schemes.

1) Accuracy and Compatibility of Early Termination

a) Setup: We provide the same rate-coded and temporal-coded input cases to uGEMM and applicable baselines. The final accuracy after complete computation is reported for each configuration. Further, we show the progressive accuracy curves throughout the processing duration and mark where it becomes stable, assuming that it is stable when keeping above 95% of the achievable accuracy. Under this assumption, we report the cycles it takes to stabilize. Results are averaged over numerous random tests for better statistical profiling.

TABLE V
FINAL ACCURACY COMPARISON OF GEMMS.

Design		Unipolar scaled	Unipolar non-scaled	Bipolar scaled	Bipolar non-scaled
uGEMM	RC	99.82%	100%	99.57%	97.59%
	TC	99.82%	100%	99.54%	61.37%
Gaines	RC	97.21%	44.13%	91.48%	N.A.
	TC	91.60%	70.45%	65.80%	N.A.
Sim	RC	98.30%	99.95%	95.52%	N.A.
	TC	91.47%	70.93%	91.48%	N.A.
Jenson	RC	100%	N.A.	100%	N.A.
	TC	100%	N.A.	100%	N.A.
Najafi	RC	97.5%	N.A.	93.4%	N.A.
	TC	N.A.	N.A.	N.A.	N.A.

b) Results: The final accuracies of all the GEMM implementations are shown in Table V. uGEMM attains accuracies of higher than 99% under all combinations of requirements on data polarity and scale except for the bipolar non-scaled temporal-coded input due to overflowing, which is induced by the low-stability TC input in this case. Nevertheless, uGEMM achieves almost perfect final accuracy comparable to Jenson's. Furthermore, uGEMM supports more input combinations, providing flexibility using same hardware. Regarding its counterparts, the accuracy is reasonable. Gaines' classic design works stably in most cases except for the unipolar RC data with non-scaled addition and bipolar data with scaled addition. By probing the input and the unit's response, we argue that the accuracy drop is caused by input-associated overflow and instability, which is reflected in Figure 9. Sim's design works well without the problem troubling the conventional stochastic units, which partially attributes to its binary addition, avoiding the inaccuracy in addition. Jenson's approach achieves perfect final accuracy as expected. The latency-optimized deterministic approach by Najafi achieves higher accuracy than Gaines' but lower than uGEMM and Sim's.

In Figure 9, we plot the curves of progressive accuracy from the first to the last cycle (256_{th} for uGEMM, Gaines', Sim's and Najafi's but 256^2_{th} for Jenson's). The stable points, from where the progressive accuracy maintains above 95% are highlighted. Whereas it statistically implies a higher likelihood that the accuracy will not degrade drastically when the computation is terminated earlier if stable points are closer to the y-axis in this plot. The gray area bounded in between the curves shows the difference between accuracies with rate-coded input and temporal-coded input, and a smaller area indicates lower input-coding sensitivity. In a nutshell, for each combination and input type, uGEMM outperforms all other approaches in terms of how early its accuracy can stabilize, up to 84% and 99.6% (on average 53% and 51% across all configurations) fewer cycles for RC and TC input. Also, except for the configuration of bipolar non-scaled GEMM, uGEMM has smaller shaded area comparatively, suggesting a higher compatibility of input data representation and distribution. Note that Najafi's design does not support temporal coding due to bit stream scrambling.

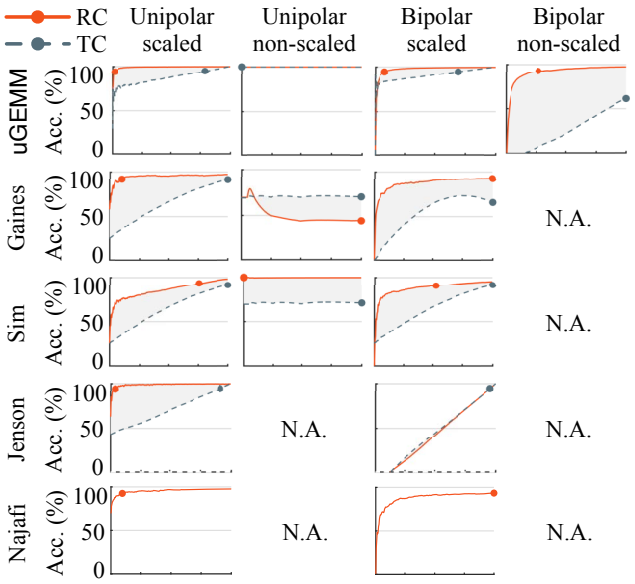


Fig. 9. Progressive accuracy (curves) and stability (dots) comparison of GEMMs. Cycle ranges from 1 to 256 in uGEMM, Gaines', Sim's and Najafi's; and 1 to 256^2 in Jenson's.

2) Hardware Implementation

a) Setup: As shown in Table VI, we evaluated all hardware implementations of corresponding configurations of GEMM for unipolar/bipolar data and scaled/non-scaled accumulation. For each version, we utilize its inherent features to optimize the implementation as better as we can. For uGEMM and Sim's GEMM, we share RNGs for the same input branches of different multipliers (input 1) in Figure 3 if their inputs 0 come from the same source data, as those RNGs will be updated synchronously. For Jenson's GEMM, the repeated bit streams can be constructed in the same manner regardless of correlation entirely, yielding a more eager resource sharing of bit stream generator. Due to the fact that the Gaines' GEMM does not have those inherent features, no sharing mechanism is applied to retain accuracy. Najafi's design has to maintain enough LFSRs to keep the accuracy, because its truncation mechanism emulates the behavior of Gaines' design using LFSRs, where the correlation problem is critical.

b) Results: From Table VI, we find that all versions of uGEMM consume small area and low power, whose values are lower than those of Gaines', Sim's and Najafi's. We also observe that Jenson's GEMM is significantly more lightweighted than others. We conclude this difference is mainly due to resource sharing schemes under the accuracy constraint recalling the comparable unit-level area and power. However, one drawback of Jenson's design is its long latency, accounting for the higher energy consumption. For both unipolar-scaled GEMM and unipolar-non-scaled GEMM, uGEMM consumes the least power, while for bipolar-scaled GEMM, it is only less than 0.02 µJ higher than the lowest one by Sim's. The bipolar-non-scaled uGEMM shows pleasant hardware efficiency, too, related to the ratio between of multiplications and additions.

TABLE VI
HARDWARE COMPARISON OF GEMMS.

GEMM	Area (mm^2)	Power (W)	Latency (µs)	Energy (µJ)
Unipolar scaled				
uGEMM	0.43	0.15	0.64	0.07
Gaines	1.57	0.50	0.64	0.32
Sim	0.52	0.18	0.64	0.11
Jenson	0.08	0.02	163.84	3.91
Najafi	1.26	0.46	0.64	0.29
Unipolar non-scaled				
uGEMM	0.44	0.12	0.64	0.07
Gaines	1.55	0.49	0.64	0.31
Sim	0.50	0.15	0.64	0.09
Bipolar scaled				
uGEMM	0.76	0.21	0.64	0.13
Gaines	1.56	0.50	0.64	0.32
Sim	0.53	0.18	0.64	0.12
Jenson	0.08	0.02	163.84	3.90
Najafi	1.25	0.45	0.64	0.29
Bipolar non-scaled				
uGEMM	0.77	0.20	0.64	0.13

To conclude, uGEMM attains excellent area, power, latency, and energy efficiency with a guarantee of accuracy compared to the state-of-the-art unary paradigms, providing better design trade-offs, i.e., best early termination capability.

3) Comparison with Non-Unary GEMM

a) Bit-Parallel Binary GEMM: Based on prior measurements [31] where unary matrix multiplication and convolution are shown to be 13% and 180% more energy-efficient than their binary counterparts, we estimate that with support for early termination (which reduces the unary latency by 84%), uGEMM can achieve similar energy efficiency as binary matrix multiplication and around 10× that of binary convolution. The unary-binary hybrid DNN accelerator in [33] improves the energy efficiency by 1.23× under the same area budget as the pure binary implementation.

b) Bit-Serial Binary GEMM: Due to less wire congestion, bit-serial computing has slightly higher energy efficiency compared to bit-parallel binary computing. According to [55], the bit-serial implementation of a DNN accelerator [23] sees around 1.7× and 1.3× higher performance and energy efficiency compared with the bit-parallel Eyeriss in [11]. Though promising, this falls short of our estimated 10× improvement in energy efficiency for uGEMM, discussed above.

Moreover, as previous studies demonstrate, in practical sensing systems, uGEMM can outperform such binary schemes by operating on unary data directly from sensors without expensive unary-to-binary data conversion [7, 33, 41, 57]. In [24], the authors use specialized analog-to-stochastic converters to transform the sensed analog signal into unary data without any binary conversion, leading to 56.3× and 2.1× average improvement in area and energy consumption, respectively.

VI. uGEMM ON MULTILAYER PERCEPTRON

As a case study, we evaluate uGEMM's accuracy and capability of early termination at the application level and compare

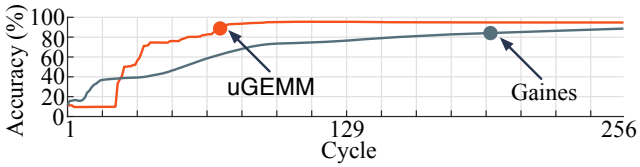


Fig. 10. Progressive MLP accuracy of uGEMM and Gaines.

against a baseline implemented with Gaines' functional units.

a) Setup: We develop a custom cycle-accurate unary computing simulator integrated with PyTorch [68]. We train a MLP model on the full MNIST training set [30] and report the progressive accuracy of classification results on the full test set. The MLP has three fully connected layers with ReLU activation [40], resulting in 791,522 MUL-ADDs. The baseline uses bipolar scaled addition and scales data up with the technique in [26]; other performance-enhancing techniques are not applied. We implement uGEMM with bipolar uMUL and uNSADD, fed with rate-coded data. Note that this implementation is the worst-case scenario for rate-coded uGEMM based on Figure 9.

b) Results: The floating-point MLP model achieves inference accuracy of 96.87%, and the accuracy is 96.08% when being quantized to 8-bit and using binary processing elements as a baseline. We observe from Figure 10 that uGEMM is able to achieve negligible accuracy loss and maintain 98.6% accuracy of the binary 8-bit MLP using the entire 256 cycles, i.e., 94.7% actual accuracy. Note that at each cycle, the accuracy is the mean accuracy of the entire test set at that specific cycle. Moreover, uGEMM demonstrates its excellent capability of early termination – after the 71st cycle (marked as colored dot in Figure 10), uGEMM stably maintains accuracy within 5% of its final accuracy. In contrast, Gaines' design only attains a final MLP accuracy of 88.58%. Moreover, to be stable, approaching its maximum accuracy, Gaines' design needs 195 cycles and a large number of RNGs to minimize correlation, unlike uGEMM whose units can share a single RNG for weights with the same input. These results all suggest that uGEMM can achieve excellent application accuracy similar to its binary counterpart while further saving time and energy in the computation compared to its unary alternative.

VII. Conclusion

We present uGEMM, an area- and energy-efficient GEMM architecture enabled by novel arithmetic units that unify rate-coded and temporal-coded unary computing. The proposed design relaxes previously imposed constraints on input unary bit streams, such as low correlation or long stream length, and achieves improved area and energy efficiency over existing unary systems. Most importantly, uGEMM achieves higher accuracy and stability than both prior rate-coded and temporal-coded schemes, facilitating early termination and flexible energy-accuracy scaling on resource-constrained systems. Our PyTorch-based cycle-accurate simulator for unary computing is publicly available [68].

Acknowledgements

We thank the reviewers for their valuable feedback. This work is supported by the Wisconsin Alumni Research Foundation, an Ontario Graduate Scholarship, and NSF under award No. CNS-1845469.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[2] E. D. Adrian and Y. Zotterman, "The Impulses Produced by Sensory Nerve-Endings: Part II. The Response of A Single End-Organ," *The Journal of Physiology*, vol. 61, no. 2, pp. 151–171, 1926.

[3] A. Alaghi, Cheng Li, and J. P. Hayes, "Stochastic Circuits for Real-Time Image-Processing Applications," in *Design Automation Conference (DAC)*, 2013.

[4] A. Alaghi and J. P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," in *International Conference on Computer Design (ICCD)*, 2013.

[5] A. Alaghi and J. P. Hayes, "Fast and Accurate Computation Using Stochastic Circuits," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014.

[6] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, p. 92, 2013.

[7] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[8] S. Behroozi, J. Li, J. Melchert, and Y. Kim, "SAADI: A Scalable Accuracy Approximate Divider for Dynamic Energy-Quality Scaling," in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019.

[9] T. H. Chen and J. P. Hayes, "Design of Division Circuits for Stochastic Computing," in *IEEE Computer Society Annual Symposium on VLSI (IVLSI)*, 2016.

[10] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[11] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[12] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *arXiv:1410.0759*, 2014.

[13] Facebook, "FB (Facebook) + GEMM (General Matrix-Matrix Multiplication)." [Online]. Available: https://code.fb.com/ml-applications/fbgemm/

[14] B. R. Gaines, *Stochastic Computing Systems*. Springer, 1969, pp. 37–172.

[15] K. Ganesan, J. San Miguel, and N. Enright Jerger, "The What's Next Intermittent Computing Architecture," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.

[16] A. J. Groszewski and T. Lenz, "Deterministic Stochastic Computation Using Parallel Datapaths," in *International Symposium on Quality Electronic Design (ISQED)*, 2019.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[18] Intel, "Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)." [Online]. Available: https://github.com/intel/mkl-dnn

[19] D. Jenson and M. Riedel, "A Deterministic Approach to Stochastic Computation," in *International Conference on Computer-Aided Design (ICCAD)*, 2016.

[20] Y. Jia, "Learning Semantic Image Representations at a Large Scale," Ph.D. dissertation, EECS Department, University of California, Berkeley, 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-93.html

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *International Conference on Multimedia (MM)*, 2014.

[22] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of A Tensor Processing Unit," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[23] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *International Symposium on Microarchitecture (MICRO)*, 2016.

[24] S. K. Khatamifard, M. H. Najafi, A. Ghoreyshi, U. R. Karpuzcu, and D. J. Lilja, "On Memory System Design for Stochastic Computing," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 117–121, 2018.

[25] K. Kim, J. Lee, and K. Choi, "Approximate De-Randomizer for Stochastic Circuits," in *International SoC Design Conference (ISOCC)*, 2015.

[26] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic Energy-Accuracy Trade-Off Using Stochastic Computing in Deep Neural Networks," in *Design Automation Conference (DAC)*, 2016.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *International Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[28] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[29] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979.

[30] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[31] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin, "Architecture Considerations for Stochastic Computing Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2277–2289, 2018.

[32] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation Manipulating Circuits for Stochastic Computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[33] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[34] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Y. Wang, "HEIF: Highly Efficient Stochastic Computing-Based Inference Framework for Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1543–1556, 2019.

[35] S. Liu and J. Han, "Energy Efficient Stochastic Computing with Sobol Sequences," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[36] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.

[37] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[38] A. Madhavan, T. Sherwood, and D. Strukov, "Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms," in *International Symposium on Computer Architecture (ISCA)*, 2014.

[39] A. Madhavan, T. Sherwood, and D. Strukov, "A 4-mm^2 180-nm-CMOS 15-Giga-Cell-Updates-per-Second DNA Sequence Alignment Engine based on Asynchronous Race Conditions," in *Custom Integrated Circuits Conference (CICC)*, 2017.

[40] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *International Conference on Machine Learning (ICML)*, 2010.

[41] M. H. Najafi, S. R. Faraji, K. Bazargan, and D. Lilja, "Energy-Efficient Near-Sensor Convolution Using Pulsed Unary Processing," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2019.

[42] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing Stochastic Computation Deterministically," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 12, pp. 2925–2938, 2019.

[43] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, "Power and Area Efficient Sorting Networks Using Unary Processing," in *International Conference on Computer Design (ICCD)*, 2017.

[44] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-Cost Sorting Network Circuits Using Unary Processing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.

[45] M. H. Najafi and M. E. Salehi, "A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 2, pp. 808–812, 2016.

[46] C. Nugteren, "CLBlast." [Online]. Available: https://github.com/CNugteren/CLBlast

[47] Nvidia, "cuBLAS." [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html

[48] Nvidia, "NVBLAS." [Online]. Available: https://docs.nvidia.com/cuda/nvblas/index.html

[49] B. Parhami and Chi-Hsiang Yeh, "Accumulative Parallel Counters," in *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, 1995.

[50] K. Parhi and Y. Liu, "Computing Arithmetic Functions Using

Stochastic Logic by Series Expansion," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 44–59, 2017.
[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," in *Neural Information Processing Systems (NeurIPS)*, 2017.
[52] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," in *International Symposium on Computer Architecture (ISCA)*, 2016.
[53] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
[54] J. San Miguel and N. Enright Jerger, "The Anytime Automaton," in *International Symposium on Computer Architecture (ISCA)*, 2016.
[55] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," in *International Symposium on Computer Architecture (ISCA)*, 2018.
[56] H. Sim, S. Kenzhegulov, and J. Lee, "DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks," in *Design Automation Conference (DAC)*, 2018.
[57] H. Sim and J. Lee, "A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks," in *Design Automation Conference (DAC)*, 2017.
[58] H. Sim and J. Lee, "Log-Quantized Stochastic Computing for Memory and Computation Efficient DNNs," in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019.
[59] J. E. Smith, "Space-Time Algebra: A Model For Neocortical Computation," in *International Symposium on Computer Architecture (ISCA)*, 2018.
[60] J. Smith and J. Shen, "Your Brain is a Unary Computer," in *International Symposium on Computer Architecture (ISCA)*, 2019.
[61] R. B. Stein, E. R. Gossen, and K. E. Jones, "Neuronal Variability: Noise or Part of The Signal?" *Nature Reviews Neuroscience*, vol. 6, no. 5, pp. 389–397, 2005.
[62] S. S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.
[63] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood, "Boosted Race Trees for Low Energy Classification," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.
[64] S. I. Venieris and C. S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016.
[65] D. Wu, T. Chen, C. Chen, O. Ahia, J. San Miguel, M. Lipasti, and Y. Kim, "SECO: A Scalable Accuracy Approximate Exponential Function Via Cross-Layer Optimization," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2019.
[66] D. Wu, Y. Chen, Q. Zhang, Y.-L. Ueng, and X. Zeng, "Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 9, pp. 873–877, 2016.
[67] D. Wu and J. San Miguel, "In-Stream Stochastic Division and Square Root via Correlation," in *Design Automation Conference (DAC)*, 2019.
[68] D. Wu and R. Yin, "UnarySim." [Online]. Available: https://github.com/diwu1990/UnarySim
[69] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
[70] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and Efficient Stochastic Computing Hardware for Convolutional Neural Networks," in *International Conference on Computer Design (ICCD)*, 2017.
[71] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.