Cambricon-U: A Systolic Random Increment Memory Architecture for Unary Computing

Hongrui Guo
SKLP, ICT, CAS UCAS
guohongrui21b@ict.ac.cn

Yongwei Zhao
SKLP, ICT, CAS
zhaoyongwei@ict.ac.cn

Zhangmai Li
SKLP, ICT, CAS HUST
lizhangmai@hust.edu.cn

Yifan Hao*
SKLP, ICT, CAS
haoyifan@ict.ac.cn

Chang Liu
SKLP, ICT, CAS UCAS
Cambricon Technologies
liuchang18s@ict.ac.cn

Xinkai Song
SKLP, ICT, CAS
songxinkai@ict.ac.cn

Xiaqing Li
SKLP, ICT, CAS
lixiaqing@ict.ac.cn

Zidong Du
SKLP, ICT, CAS SHIC
duzidong@ict.ac.cn

Rui Zhang
SKLP, ICT, CAS
zhangrui@ict.ac.cn

Qi Guo
SKLP, ICT, CAS
guoqi@ict.ac.cn

Tianshi Chen
Cambricon Technologies
tchen@cambricon.com

Zhiwei Xu
SKLP, ICT, CAS UCAS
zxu@ict.ac.cn

ABSTRACT

Unary computing, whose arithmetics require only one logic gate, has enabled efficient DNN processing, especially on strictly power-constrained devices. However, unary computing still confronts the power efficiency bottleneck for buffering unary bitstreams. The buffering of unary bitstreams requires accumulating bits into large bitwidth binary numbers. The large bitwidth binary number needs to activate all bits per cycle in case of *carry propagation*. As a result, the accumulation process accounts for 32%-70% of the power budget.

To push the boundary of power efficiency, we propose Cambricon-U, a systolic random increment memory architecture featuring efficient accumulation. By leveraging *skew number* data format, Cambricon-U only activates no more than three bits (instead of all bits) from each number per accumulating cycle. Experimental results show that Cambricon-U reduces 51% power on unary accumulation, and improves 1.18-1.45× energy efficiency over uSystolic, the SOTA unary computing scheme baseline, with -1.9%∼+0.77% area overhead.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Hardware** → **Emerging architectures**.

*Corresponding Author.

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10…$15.00
https://doi.org/10.1145/3613424.3614286

KEYWORDS

skew number; systolic array; unary computing;

ACM Reference Format:
Hongrui Guo, Yongwei Zhao, Zhangmai Li, Yifan Hao, Chang Liu, Xinkai Song, Xiaqing Li, Zidong Du, Rui Zhang, Qi Guo, Tianshi Chen, and Zhiwei Xu. 2023. Cambricon-U: A Systolic Random Increment Memory Architecture for Unary Computing. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3613424.3614286

1 INTRODUCTION

Recently, unary computing has re-emerged as a high power and area efficient alternate to conventional binary computing for processing deep neural networks (DNNs) inference on strictly power-constrained IoT devices [9, 30, 31, 37, 43, 53]. The high efficiency of unary computing comes from its simple data format and therefore simple arithmetics. Unary computing encodes binary data into 1-bit data streams (i.e., bitstreams) [15], significantly reducing the power consumption of data read/write. Meanwhile, computing with the unary bitstreams requires simple arithmetic processing unit (e.g., one AND or XNOR gate), avoiding the costly binary arithmetic units (e.g., 16-bit multipliers). Recent works leverage such simplicity for high compact hardware implementations with high computational parallelism and data reuse ratio [46, 55].

However, even if such computational simplicity reaches its ceiling, it cannot break through the power bottleneck caused by the complex buffering of unary bitstreams. As evidence, for state-of-the-art unary computing accelerator uSystolic [55], the buffering of unary bitstreams occupies 58%-78% of a unary systolic array (Figure 1) and 32%-70% of the total power consumption (Figure 4). The reason behind is that the buffering of a unary bitstream requires it to accumulate bit by bit into a large bitwidth binary number. During such unary accumulation, the binary number needs to activate all bits per cycle in case of *carry propagation* (i.e., each bit of the

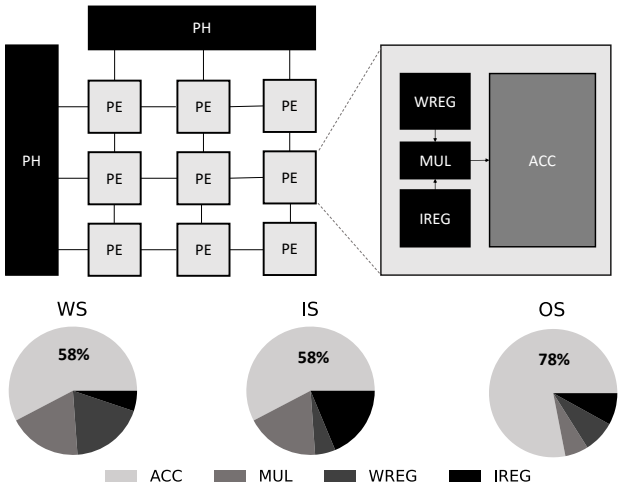


Figure 1: Breakdown of power in a unary systolic array (WS: Weight-stationary; IS: Input-stationary; OS: Output-stationary). Each PE contains a *weight register* (WREG), an *input register* (IREG), a *unary multiplier* (MUL) and an *accumulator* (ACC) for buffering unary bitstreams. ACC accounts for 58%, 58%, and 78% power of the array, respectively.

number may flip due to the ripple of carries from LSB to MSB). For example, the carries during accumulating '1' (from a unary bitstream) onto (the binary number) '01111b' will flip all 5-bits. To handle this case, the full-bitwidth adder (which requires activating all bits of the operator) is needed for unary accumulation, thus incurring large power overhead.

To break through the power bottleneck, we propose a systolic random increment memory architecture (Cambricon-U), which features efficient accumulation when buffering unary bitstreams. On the one hand, the storage module of Cambricon-U is the RIM array, which buffers unary bitstreams from PEs into *skew numbers* (i.e., a non-standard ternary data format where at most one '2' can appear in a digit). Specifically, when accumulating one bit of a unary bitstream into the RIM, unlike the binary number that needs to activate all bits, the skew number only needs to activate the least significant bit or the adjacent bits of '2' (i.e., no more than 3-bits), so that heading off the carry propagation. Therefore, by leveraging the skew number data format, the full-bitwidth adder can be replaced by a half-adder without sacrificing execution time, thus significantly reducing power consumption. On the other hand, the calculation module of Cambricon-U is a systolic PE array that can fully utilize the accumulation efficiency of RIM. Specifically, the PE array is not only responsible for unary arithmetic operations, but also for offsetting the accumulated digit values from '-1,0,+1' to '0,+1,+2', thereby avoiding unary subtraction in RIM. Additionally, multiple PEs share a Converter that converts the skew number (read out from RIM) to the binary number, serving as an interface between the Cambricon-U core and external storage. Moreover, in order to further clarify the efficient unary accumulation mechanism in the Cambricon-U, as a supplement to the architecture design, we also conducts a detailed circuit design and simulation of the RIM.

We conduct experiments to evaluate Cambricon-U against uSystolic [55] under versatile configurations, including rate and temporal-coding schemes, ranging from 32 to 2048 MAC cycles, as well as typical dataflows such as WS, IS, and OS, on MLPerf-Tiny [4] benchmarks. The RIM is evaluated through detailed circuit design and simulation. Experimental results show that RIM reduces the power of accumulation by 51%. The energy efficiency of Cambricon-U is improved 1.18-1.45× over uSystolic, with only -1.9%∼+0.77% area overhead.

This paper makes the following contributions:

- We demonstrate buffering unary bitstreams by accumulating bits to large bitwidth binary numbers is the power bottleneck in contemporary unary computing architectures due to the full bitwidth activation in case of *carry propagation*.
- We propose Random Increment Memory (RIM) which accumulates unary bitstreams to *skew numbers* for power efficient accumulation where no more than three bits are activated per cycle.
- We propose Cambricon-U, a unary computing architecture leveraging RIM for buffering unary bitstreams, which improves energy efficiency with minimal area overhead.
- We evaluate Cambricon-U in the context of power-constrained devices on DNN benchmarks from MLPerf-Tiny [4], and demonstrates the effectiveness of the skew format buffering to reduce energy consumption.

The rest of the paper is organized as follows: Section 2 introduces the basics of unary computing and uSystolic, the state-of-the-art unary computing architecture. Section 3 analyzes the power bottleneck of contemporary unary computing architectures. Section 4 presents the structure and functions of RIM. Section 5 presents the architecture of Cambricon-U. Section 6 describes the detailed circuit design of RIM. Section 7 describes the evaluation methodology and presents the experimental results of Cambricon-U. Section 9 discusses the related works, and Section 10 gives the conclusions.

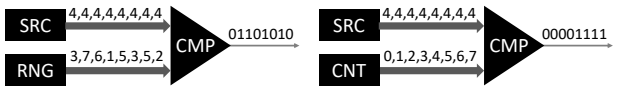
2 BACKGROUND

In this section, we review the basic concepts of unary computing and state-of-the-art unary computing architectures.

2.1 Unary Computing

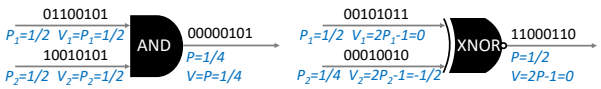
By representing numbers as unary bitstreams, unary computing enables low power multiplication with simple logic gates. After multiplication, unary-binary conversion occurs between consecutive DNN layers.

2.1.1 Unary Representations. The representation of a number using a unary bitstream can be achieved through two coding schemes, namely rate-coding and temporal-coding, as shown in Figure 2. The first scheme, known as rate-coding or stochastic computing [2, 15, 35], utilizes the frequency of 1s to represent a number. In this scheme, the frequency of 1s, denoted as P, can be translated into a number in two ways corresponding to two unary bitstream formats: unipolar or bipolar [6]. The unipolar format represents an unsigned number within the range of $[0, 1]$ as P, while the bipolar format represents a signed number within the range of $[-1, 1]$ as $2P - 1$ [29]. Both formats require a stochastic number generator



(a) Rate-coding (b) Temporal-coding

Figure 2: The two coding schemes of unary representations. A unary bit is generated by comparing SRC with either a random number from a *random number generator* (RNG) in (a) rate-coding or a number from a *counter* (CNT) in (b) temporal-coding.



(a) Unipolar (b) Bipolar

Figure 3: Unary multiplication of (a) unipolar and (b) bipolar formats of unary bitstreams, where P is the probability of bit 1s. An AND gate is used to multiply two unipolar bitstreams, and an XNOR gate is used to multiply two bipolar bitstreams.

(SNG) [33] to generate bitstreams. The stochastic number generator is comprised of three distinct components: a random number generator (RNG), a source register, and a comparator. The comparator compares the value of the source register and a random number to generate one bit per cycle. This process is repeated until the desired number of bits is reached.

The second scheme is temporal-coding, also known as race-logic [36, 39, 40, 52], which represents a number based on the transition timing of a signal. In this scheme, a temporal-coded bitstream always contains consecutive 1s followed by consecutive 0s. Unlike rate-coding which uses RNGs to generate bitstreams, temporal-coding replaces RNGs with counters [9, 17, 48, 49], which reduces the power overhead of unary bitstream generation. Additionally, temporal-coding allows for efficient bitstream generation from analog signals, making it more suitable for near-sensor computing applications [53]. In this paper, both rate-coding and temporal-coding are evaluated in order to identify the power bottleneck of current unary computing architectures and to compare with ours.

2.1.2 Unary Arithmetic Operations. A unary GEMM operation is composed of multiplications and additions. Multiplication is performed in unary computing employing elementary logic gates, specifically an AND gate for unipolar format and an XNOR gate for bipolar format, as shown in Figure 3. Within this framework, the AND gate is more preferable to researchers for two reasons. First, the AND gate is notably less expensive than the XNOR gate, with lower power dissipation during multiplication. Second, the AND gate exhibits superior accuracy over the XNOR gate, which frequently produces large errors when operands approach zero [56, 57]. Although the AND gate is notably simpler and more precise than the XNOR gate, it is restricted on unsigned numbers. Thus, researchers tend to execute one inner product via two rounds, one for positive results and another for negative results [31, 44, 46], either in parallel or serial. In an effort to mitigate the double execution overhead, [54] recommends an innovative approach that executes precise multiplication operations within a solitary round, effectively reducing the double time or power overhead. Although this represents a significant advancement, achieving a one-round AND gate multiplier results in greater complexity of the accumulator, thereby increasing the power overhead required for accumulation [55].

Addition is another vital operation in the context of unary GEMM, where it facilitates the combination of multiple bitstreams. When summing two bitstreams, either a multiplexer (MUX) [42] or an OR gate [12] can be employed. Specifically, the MUX operates by performing a scaled addition, denoted as $MUX(P_1, P_2) = (P_1 + P_2)/2$, using two input operands P_1 and P_2. For n-input operands, an n-input MUX is utilized to perform an addition that is scaled by a factor of n, resulting in $MUX(P_1, P_2, ..., P_n) = (P_1 + P_2 + ... + P_n)/n$. However, the large scaling factor n restricts the applicability of MUX addition to narrow widths [12]. Furthermore, to achieve adequate accuracy in practical DNNs, it is necessary to integrate neurons with inputs numbering in the hundreds or thousands, making MUX additions unsuitable. Alternative OR gate-based additions are free of scaling factors but experience systematic errors, with $OR(P_1, P_2) = P_1 + P_2 - P_1P_2$ giving rise to errors that cannot be disregarded when P_1P_2 is relatively large [46]. Moreover, the OR gate-based addition method is further affected by saturation, which decreases accuracy. What is worse, the above addition methods are designed specifically for rate-coding, whose accuracy highly relies on the correlation [1] between the input bitstreams, thus failing to cater for accurate addition of temporal-coded bitstreams [54]. Because of the limitations of unary additions, it is necessary to perform addition in binary domain by accumulating 1s accurately [9, 17, 48, 49, 55].

2.1.3 Unary-Binary Conversion. The unary-binary conversion occurs at the start and end of the execution of one DNN layer. The conversion from binary to unary format is achieved by applying unary stream generators, which utilize a RNG in rate-coding and a counter in temporal-coding. Conversely, the conversion from unary to binary format is typically executed using binary accumulators in light of the inaccuracy of unary adders. Based on the unary multiplication result's sign and absolute value, the binary accumulator adjusts the accumulated value by incrementing, decrementing, or maintaining it [55]. Specifically, given the multiplication result's sign as A and the produced multiplication bit as B, the accumulator modifies the sum S according to Equation 1 where $A = 0$ and $A = 1$ represents positive and negative, respectively.

$$S_{i+1} = \begin{cases} S_i + 1, & A_i = 0, B_i = 1 \\ S_i - 1, & A_i = 1, B_i = 1 \\ S_i, & o.w. \end{cases} \quad (1)$$

2.2 uSystolic

uSystolic [55], the state-of-the-art unary computing architecture, leverages a unary systolic array which comprises a unary PE array and peripheral components (PH), as shown in Figure 1. In the unary PE array, each PE is composed of four components: weight registers (WREG), input registers (IREG), a unary multiplier (MUL) and an accumulator (ACC) for the buffering of unary bitstreams. While all dataflows employ a costly large-bitwidth adder for unary

accumulation (ACC), the other three parts (WREG, IREG, and MUL) are subtly different in various dataflows. In the case of weight stationary (WS) dataflow, the WREG comprises a random number register, a weight absolute value register, and a weight sign register, while the IREG is made up of an input sign register and a D-flipflop (DFF) buffering one unary bit of an input. The MUL consists of a comparator (CMP) to generate one unary bit of weight per cycle, an AND gate utilized for unary multiplication and an XOR gate that computes the sign. The bitwidth of WREG and CMP can be 8-bit or 16-bit to support different resolutions. In the case of input-stationary (IS) dataflow, the structure of a PE is akin to that of WS dataflow, with the only difference being the interchange of weight and input. As for output stationary (OS) dataflow, both the WREG and IREG are constituted by a sign register and a DFF buffering one unary bit. The MUL in OS dataflow only comprises an AND gate and an XOR gate.

The peripheral components (PH) include unary stream generators (UBGs), random number generators (RNGs) and FIFOs. In WS and IS dataflow, the PH on the left side of the unary systolic array consists of a single column of FIFOs and UBG-RNG pairs. Each row of PEs share one pair of UBG and RNG. On the other hand, the PH on the top of the systolic array consists of only FIFOs. In OS dataflow, both the PHs consist of one column/row of FIFOs and UGBs, with each UGB shared by one row/column of PEs. Overall, the PHs play a crucial role in ensuring efficient data reuse and synchronization across the systolic array.

The unary systolic array operates similarly to a binary systolic array [23, 47]. First, if necessary, relevant data are preloaded into the PEs and remain static until the output feature maps are calculated. For example, in WS dataflow, the weights are flowed into weight registers of each PE before computing. The UBGs or RNGs in peripheral components then generate unary bitstreams or random numbers, which are fed to the systolic array and reused by multiple PEs. The reuse of unary bitstreams or random numbers allows locating unary stream generators only at the peripheral components, amortizing the power and area overhead among one line of PEs. As the unary bitstreams or random numbers are streamed in the systolic array, the ACC in each PE accumulates the unary bitstream generated by the multiplier.

3 MOTIVATION

Recent architectures only focus on leveraging simple data format and simple arithmetics for accelerating unary computing. For instance, [49] eliminates the AND gate for unary multiplication and gains higher compact hardware implementations. SkippyNN [17] and DPS [48] focus on reducing MAC cycles with higher computational parallelism. uSystolic [55] pays attention to more efficient data reuse to amortize the power of random number generators across PEs.

Although previous work has fully leveraged the simplicity of unary arithmetic, the hardware power efficiency is still hindered by the extensive unary accumulations when buffering unary bitstreams. As evidence, unary accumulations in the buffering of unary bitstreams (shown in the orange color legend 'ACC' in Figure 4) occupy the largest portion (32%-70%) of the total power consumption in all evaluation scenarios, including rate and temporal-coding

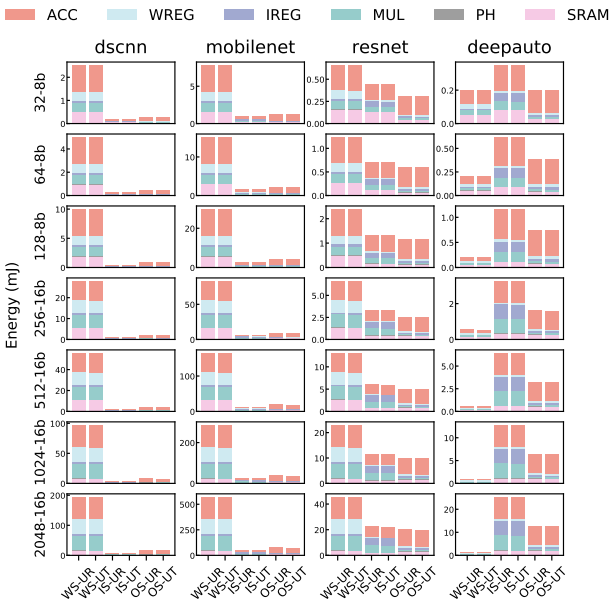


Figure 4: The breakdown of energy utilization of unary systolic array architectures under rate (UR) and temporal-coding (UT) schemes, ranging from 32 to 2048 MAC cycles (the -8b and -16b indicate the 8 and 16 bits for two multiplication inputs), as well as typical dataflows such as WS, IS, and OS.

schemes, ranging from 32 to 2048 MAC cycles, as well as typical dataflows such as WS, IS, and OS.

To explain this power bottleneck caused by unary bitstream buffering, we would clarify two questions: 1) why buffering unary bitstreams requires extensive unary accumulations, and 2) why unary accumulations bring large power consumption overhead.

The answer to question 1) is that, to save the memory capacity of buffering a unary bitstream, it is necessary to accumulate it bit by bit into a large bitwidth binary number. Specifically, as the unary bitstream typically includes hundreds or even thousands of bits, buffering these bits requires large memory capacity, thus causing high power and area overhead. On the contrary, if accumulating an N-bit unary bitstream into a binary number and only the binary number is buffered, the memory overhead only needs no more than Log_2N bits. For example, a 256-bit unary bitstream can be accumulated into an 8-bit binary number, saving 32× memory overhead.

The answer to question 2) is that, the unary accumulation suffers from the *carry propagation*, thus requiring a costly full-bitwidth adder. Specifically, when accumulating '1' from a unary bitstream onto a binary number, each bit of the binary number may be added with a carry, flipping and generating a new carry for the next bit. As a result, these carries may propagate from LSB to MSB (i.e., namely *carry propagation*), and cause a full-bit-flipping of the binary number. For example, when accumulating '1' onto '01111b', carries will flip all five bits. In case of *carry propagation*, unary accumulation

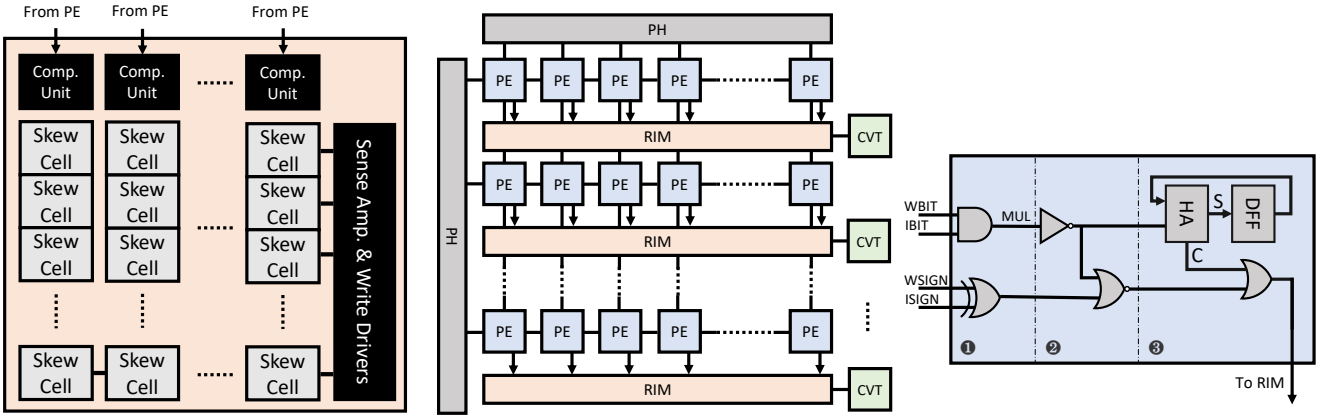


Figure 5: The Cambricon-U architecture (middle) which integrates RIM (left) with a novel PE (right) that transforms accumulation to increment.

needs to activate all bits of the binary number, so it needs to be implemented on a full-bitwidth adder, which causes the large power dissipation of the accumulator. As Figure 1 shows, for all typical dataflows (i.e., WS, IS, and OS), the accumulator in a PE accounts for the largest power consumption (shown in the light gray color legend 'ACC'), which is 3.1-13.3× of the other components. Such a significant power consumption gap between different parts is due to the complexity difference of their corresponding circuit (i.e., a costly full-bitwidth adder v.s. other components, only consisting of simple logic gates, narrow bitwidth registers, and comparators).

In short, given that buffering unary bitstreams in a memory saving manner inevitably requires extensive unary accumulations, to push the power efficiency boundary of unary computing architectures, a dedicated hardware design for accelerating unary accumulations is urgently called for.

4 RANDOM INCREMENT MEMORY

To break through the power bottleneck of unary computing, we propose *Random Increment Memory* (RIM) for efficient unary accumulations.

4.1 Structure

RIM is mainly comprised of *skew cells* and *computation units* for power-efficient increment of *skew numbers*, as shown on the left of Figure 5.

4.1.1 Skew Cell. RIM leverages skew cells to memorize skew numbers. The skew number system [13, 38] is a non-standard positional system where the weight of the n-th digit is $2^{n+1} - 1$. The key difference between the skew number system and the binary number system is that at most one '2' can appear in a skew number. Therefore, each skew cell is designed to memorize one of three values : '0', '1' or '2'. Besides memorization, each skew cell can also activate appropriate skew cells in the increment process. As a result, the acceleration core performs calculation in unary format and accumulation in skew format, while the uncore part, including the memories for input/output feature maps and weights, can keep using traditional binary format.

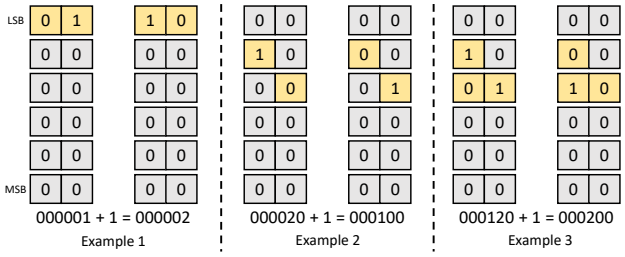


Figure 6: Examples of the increment of a skew number. The flipped bits are annotated. At most three bits are flipped in one increment step.

4.1.2 Computation Unit. The increment of skew numbers only needs to activate no more than 3 bits, thus heading off the carry propagation. The increment of skew number operates depending on whether a '2' exists in the skew number. If the skew number does not contain a '2', the least significant digit of the skew number is incremented. As the resultant least significant digit is still no more than '2', no carry bit will propagate to the next digit. As shown in the Example 1 in Figure 6, increment a skew number '000001' leads to '000002', where only two bits are flipped. If the skew number does contain a '2', the only '2' will be cleared to '0' and the next digit of the '2' is incremented. In this case, because the '2' is unique in the skew number, the next digit of the '2' still cannot exceed '2' after increment, thus also cutting off the carry propagation. As the Example 2 and 3 in Figure 6 show, the increment of a skew number containing a '2' only needs to flip two or three bits. Overall, both circumstances only need to activate at most three bits in an increment step, no matter the bitwidth of the skew number.

As a result, the computation units only need a half-adder instead of a full-bitwidth adder for increment, thereby considerably reducing power consumption. In one increment step of a skew number, only one digit is activated to be incremented. The incremented digit is '0' or '1', as the unique '2' will not activate itself, according to the rules of skew increment. Therefore, the increment only involves two kinds of computation: '0+1' and '1+1'. Both '0+1' and

'1+1' can be accomplished by a half-adder instead of a full-bitwidth adder. The half-adder is comprised of only two logic gates, while the costly full-bitwidth adder consists of tens or even hundreds of gates, which consumes much more power than a half-adder.

4.2 Functions

4.2.1 Increment. RIM is mainly designed to perform efficient increment on the skew numbers selected by bits from PEs. As shown in the left of Figure 5, ❶ RIM accepts a line of bits, which indicate which skew numbers are to be incremented. ❷ Each skew number then activates one of its skew cells according to the position of '2'. Different skew numbers can activate skew cells at different positions at the same time. ❸ The activated skew cells are then incremented by the computation units at the top line of RIM. ❹ At last, the '2's, if any, adjacent to the activated skew cells are cleared to zero. The skew increment is more efficient than a binary increment, in that only skew cells and the '2's (instead of all bits) are activated in each increment step, overcoming the *carry-propagation* of the binary increment. Experiments show the skew increment can save 51% energy of traditional binary accumulation, by modeling the power of RIM accumulator and binary accumulator using SPICE and ICC2, respectively (See Section 7.1).

4.2.2 Parallel Read-Out. Each RIM also supports to read out all bits of one skew number and clear it to zero in one cycle. When a unary bitstream has completed accumulating to a skew number, all bits of the skew number can be read out to the sense amplifiers at the right side of RIM. The reason of incorporating the read-out function in RIM is that the skew numbers belonging to the same RIM do not finish accumulation at the same cycle. That is, in one cycle, a skew number is read out while the others can still be in the process of accumulation. The coexistence of reading-out and accumulation allows preserving the dataflow of the unary systolic array architectures, thus inheriting the benefits such as generality, spatial-temporal reuse, early termination support, etc [55].

5 CAMBRICON-U ARCHITECTURE

5.1 Overview

Figure 5 depicts Cambricon-U's overall architecture, which is an array of RIMs integrated with a unary systolic array to fully utilize the accumulation efficiency of RIMs. Specifically, the PEs are designed to transform accumulation to increment. The peripheral components (PHs) on the left and top of the systolic array are the same as those in unary systolic array architectures, as shown in Figure 1. On the right side of the systolic array is a column of converters serving as an interface between the Cambricon-U core and external storage.

In the rest of this section, we first introduce how PE is designed to fully give into play the low power increment of RIM. Then we show the converter which converts a skew number to a binary number to avoid additional memory access and capacity brought by the larger bitwidth of a skew number. Finally, we explain how to address the scalability issue of Cambricon-U.

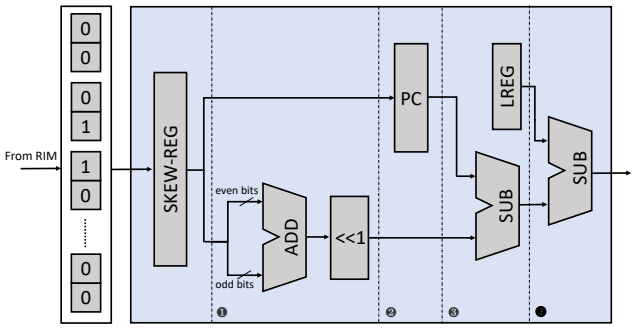


Figure 7: The converter that converts a skew number read out from RIM to a binary number. The converter consists of a skew-number register (SKEW-REG), a bitstream length register (LREG), a parallel counter (PC) to sum up the values of skew-cells, which can be 0,1,or 2, an adder (ADD) and two subtractors (SUB).

5.2 PE

As RIM can only perform increment, the PE should be designed to transform accumulation to increment. The accumulated digit values can be '-1,0,+1' in each accumulation step, while the RIM only accepts '0,+1'. To bridge the gap between accumulation to increment, two steps should be taken by each PE. Firstly, the accumulated digit values are offset by '1' to avoid subtraction. Specifically, the '-1,0,+1' values are replaced by '0,+1,+2'. Secondly, to transform the '0,+1,+2' to '0,+1', each PE maintains 1-bit state to ensure that only 1-bit is transferred to RIM per cycle.

The transformation can by accomplished by simple hardware, as shown in the right of Figure 5. In Figure 5, the transformation module is comprised of three parts. The first part is the unary multiplier which generates the multiplication result in sign-magnitude format [57], which contains one bit for sign (SIGN) and one bit for the absolute value (MUL). The two bits are fed to the second part, which generates one of '0,+1,+2' according to the value encoded by SIGN and MUL. Finally, the third part generates 1-bit to RIM given the 2-bits from the second part. The reduction in the number of bits is achieved by incorporating 1-bit state inside the PE. The 1-bit state consists a half-adder and a D-flipflop (DFF). The DFF is updated by the output of the half adder. The carry bit generated by the half adder is OR-ed with the high bit of the '0,+1,+2' from the second part. The output of the OR gate is then transferred to the RIM. Note that transforming accumulation to increment leads to the value offset by the length of accumulated bits, so the skew numbers inside a RIM should subtract the length of accumulated bits to restore their original values.

5.3 Converter

Figure 7 depicts the design of a converter that converts a skew number to a binary number, in order to avoid additional memory access and capacity. As a skew number has as twice bitwidth as a binary number, the external memory access and capacity will be doubled if storing numbers in skew format. To avoid such overhead, a converter is designed to transform a skew number to a binary

number in four steps: ❶ An adder adds up the even bits (the high bit of a skew cell) and odd bits (the low bit of a skew cell) of a skew number, and append a '0' to the LSB of the sum. ❷ A parallel counter (PC) counts the number of 1s in the skew number, where a '2' accounts for two 1s. ❸ A subtractor subtracts the number of 1s from the addition result, and generates the binary format number corresponding to the read-out skew number. ❹ The generated binary number should subtract the length of accumulated bits in order to eliminate the influence of the transformation from accumulation to increment. The length of accumulated bits is a static value, such as the length of a unary bitstream, which can be known before the execution of a DNN layer. As a result, the length register (LREG) can be set only when beginning to execute a DNN layer, and can be overlapped with the unary bitstreams flowing in the systolic array.

One converter can be shared by multiple PEs without contention, thus amortizing the power overhead. Usually, a bitstream, which is much longer than the number of PEs sharing a converter, is streamed in the pipeline of these PEs. Due to the long length of the bitstream, all of these PEs can be occupied by the same bitstream. Therefore, it is impossible for the PEs sharing a converter to read out two results from RIM at any cycle. As at most one PE per RIM generates output at the same cycle, only one column of converters are located at the right side of the systolic array, as shown in Figure 5. Because the PEs per RIM share the same converter, the converters add only 3% power overhead of each PE of an 8-bit 64×64 systolic array. The data is obtained using ICC2 (See Section 7.1.2). The low overhead of converters demonstrates that it is worthwhile to convert skew numbers to binary numbers, avoiding doubling the external memory access and capacity.

5.4 Scalability

As the read-out operation relies on shared buses, as in normal SRAM, the number of columns in a RIM cannot be scaled infinitely, thus limiting the horizontal scalability. The horizontal scalability issue can be addressed through banking. Each bank can have multiple rows of PEs, but each row of a bank only has one RIM. The multiple banks are splitted by pheripheral components, such as sense amplifiers, write drivers, and additional registers and adders (for WS and IS), but share the unary bitstream generators and RNGs. The overhead of banking is small for two reasons. First, the banking only introduces simple hardware like adders and registers, not including the costly unary bitstream generators. Second, the size of the systolic array can still scale in the vertical direction. Therefore, we choose banking for simplicity and efficiency.

The key to ensuring Cambricon-U can scale-up is to avoid RIM contentions caused by sharing converters between PEs. In general, to ensure accuracy, the length of a bitstream (e.g., 512 bits) is much longer than the number of PEs (e.g., 32 PEs) sharing the same converter, meaning that all the pipeline stages of the group of PEs can be fully occupied by the bitstream. As a result, in one cycle, the group of PEs only read out at most one skew number from RIM, thus avoiding contention for RIM and ensuring scalability friendliness.

Specifically, to further illustrate why Cambricon-U can avoid contentions, we would clarify the sharing schemes of converters under different dataflows, respectively. For WS and IS dataflow, each RIM is equipped with one converter. When a PE has finished processing the last bit of a unary bitstream, the accumulated value (i.e., the multiplication result) is read out from the RIM to the sense amplifiers at the right side of the bank in one cycle. Then the read-out data is transformed to binary format by a converter and accumulated by an adder to obtain the partial result of the inner-product. For OS dataflow, each row of PEs share a converter, as the bitstream of an inner-product is usually long enough to occupy all PEs in a row. As a result, only the RIMs in the rightmost bank are equipped with one column of converters, while other banks only needs a column of registers instead of converters to stream out the read-out data out of the acceleration core in a pipeline manner.

6 CIRCUIT-LEVEL DESIGN OF RIM

In this section, we describe the detailed circuit design of RIM and the steps to perform one increment in RIM.

6.1 Implementation

RIM is composed of skew cells which consists of one SkewBit and one DataBit, as Figure 8 shows. Besides a normal cell, the SkewBit can activate neighbor skew cells. All bitcells are 8T SRAM cells that support reading-out one column of bits in parallel, as in [22]. In a SkewBit, the TG and CMOS transistors constitute a two-level selection of four HWL sources. The first level selects from GND and $A[i-1]$, the activation signal from the last skew cell, by $C1/C1B$. The GND is selected when pre-charging BL/BLBs, and $A[i-1]$ is selected when performing computation. The second level selection is controlled by $C2/C2B$, $C3/C3B$ and $C4$. The signal selected by the first level is also selected by the second level when $C2/C2B = 1/0$ before the step when the '2', if any, is cleared to zero. In this step, $C3/C3B$ or $C4$ selects a signal according to the number of '2' in the column. If there are two '2's, the $C3/C3B$ selects $TQ[i+1]$ to let the next SkewBit activate the current one. If there is one '2', the $C4$ selects VDD to open the pass transistors of the bitcells of all SkewBits. The only SkewBit storing a '1' is then cleared to zero.

The NOR gate is used to avoid recurrent writing. The recurrent writing would happen when a new '2' is written to RIM, and the new '2' would activate the next skew cell recurrently until the newly written value is not '2'. To solve this issue, we utilize a NOR gate to mitigate the recurrent activation signal, as shown in the right bottom of Figure 8. The NOR gate takes the QB signal of the bitcell of the current Skew-Bit, denoted as $QB[i]$, and the Q signal of the bitcell of the last Skew-Bit, denoted as $Q[i-1]$, as inputs, and the generated signal is $NOR(QB[i], Q[i-1]) = AND(Q[i], QB[i-1])$, which means that only when the current SkewBit is '1' and the last SkewBit is '0', the next SkewCell can be activated. The AND gate is replaced by a NOR gate to reduce two CMOS transistors.

The Computation Unit is responsible for increment and control signals generation ($C3/C3B$, $C4$, and END), as shown in the left top of Figure 8. The increment is performed by two SA/WDs. The control signals B/BB and $WDATA/WEN$ control the increment operation inside SA/WDs. When B is '0', the BL/BLB cannot be charged/discharged even if WEN is '1'. Otherwise, the $SBL/SBLB$ and $DBL/DBLB$ are charged/discharged by the two SA/WDs, according to the latch. This avoids an explicit half-adder to reduce

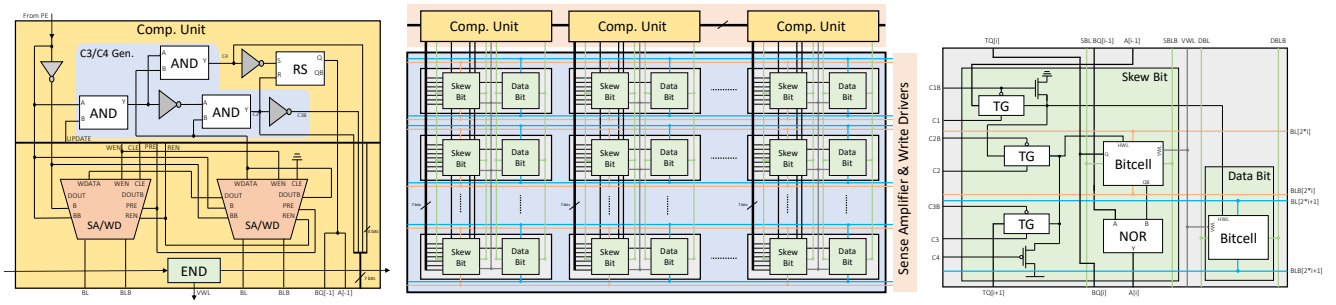


Figure 8: The circuit-level design of RIM. To the middle: RIM Overview. To the left: Computation Unit. To the right: Skew Cell.

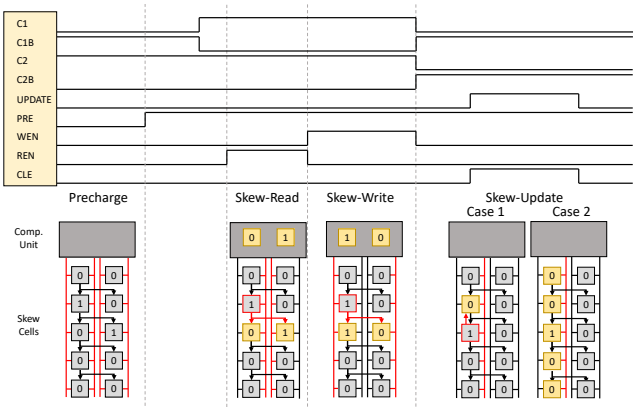


Figure 9: The four steps of one RIM increment.

power consumption. The control signals generation is also achieved by simple logic, as shown in the shadow part of the left top of Figure 8. Only five basic logic gates are leveraged to generate $C3/C3B$ and $C4$. Moreover, the END signal, which indicates whether reading out a column of bits, is generated by a D-flipflop whose value is piped in from the leftmost Computation Unit. In addition, an RS latch is placed beside the control signals generation logic to utilize $C3$ and $C4B$ as inputs to provide the inputs ($A[i-1]$ and $BQ[i-1]$) of the first SkewBit.

6.2 Operation Steps

One increment in RIM has four steps: Pre-charge, Skew-Read, Skew-Write, and Skew-Update, as shown in Figure 9. The first step, Precharge, prepares to read data from RIM. In the Skew-Read step, the '2' activates the next skew cell to the Computation Unit. Afterward, the BL/BLB is discharged for a while. Then, the REN signal is set to '1' and the sense amplifier senses the difference between BL and BLB and stores the read value into a latch. Next, the Skew-Write step is started by setting WEN to '1' to increment the activated skew cell and writes the new value back to RIM. Finally, the Skew-Update step has two cases. First, if a new '2' is written to RIM in the Skew-Write step, the new '2' activates the old '2' and clears it to zero. Second, if no new '2' is generated, all bits are activated to write zero, as the old '2' cannot activate itself. Note that in this case, only one bit is flipped, so the power is still acceptable. In both cases, the $UPDATE$ is set to '1' and $C1/C1B$ and $C2/C2B$ are flipped. Moreover, appropriate spaces are inserted to limit the short-circuit current.

7 EVALUATION

In this section, we evaluate Cambricon-U with respect to energy efficiency, power, area and application accuracy, and compare Cambricon-U with various unary systolic array architectures.

7.1 Experimental Setup

7.1.1 Modeling of RIM. We use SPICE model to simulate a RIM which contains 32 skew numbers (totally 1Kb, 16 digits or 32 bits per skew number) under the FreePDK 45nm technology node [41], using Nominal process corner and VTG devices. Using the Cadence Virtuoso software, we design the circuit and layout of a RIM, with DRC and LVS passed. We extract the parasitic parameters and perform post-layout simulation under 1.1V for accurate power statistics. As shown in Figure 10, the RIM layout consists of the skew cells, the computation units at the top and sense-amplifiers/write-drivers at the right side. The excitation signals for simulation are generated based on additional execution statistics from trace profiling. All powers are obtained by integrating current over time.

The simulation results show that each skew number dissipates 39.0 uW on average when performing increment, and reading one skew number out dissipates 664.3 uW on average. The static and dynamic power/current of one increment are 0.32uW/0.29uA and 38.68uW/35.16uA, respectively. The layout of the RIM shows that it has 1.04×10^{-2} mm^2, equivalent to each skew number contributing 324.8 um^2 area. Although the area is a bit larger than that of an accumulator, the power dissipation is only 41% (totally 49% plus the modification of PE) of an accumulator because of the small number of activated bits. Moreover, the simulation shows that the frequency of increment and read-out can follow the frequency of the systolic array, which operates at 400 MHz, the same frequency as the uSystolic [55] baselines.

7.1.2 Hardware Configurations. The Cambricon-U architecture is compared with uSystolic [55], the state-of-the-art unary computing architecture, with respect to diverse coding schemes such as rate-coding and temporal-coding, MAC cycles ranging from 32 to 2048

Table 1: Hardware Characteristic of Cambricon-U (OS-UR, 8-bit)

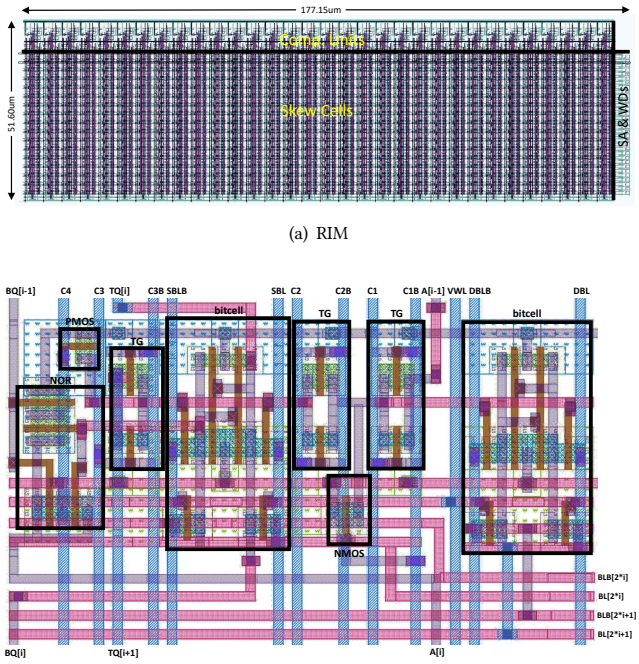
	Area (mm^2)	(%)	Static Power (mW)	Dynamic Power (mW)	Total Power (mW)	(%)
uSystolic	3.78	100	63.30	490.84	554.14	100
ACC	1.43	37.83	13.15	373.14	386.29	69.71
IREG	0.11	2.91	0.90	38.79	39.69	7.16
WREG	0.11	2.91	0.90	38.79	39.69	7.16
MUL	0.02	0.53	0.27	28.84	29.11	5.25
PH	0.02	0.53	0.18	8.01	8.19	1.48
SRAM	2.09	55.29	47.90	3.27	51.17	9.23
Cam-U modified	-0.05	-1.32	-10.42	-162.43	-172.85	-31.19
ACC	1.23	-5.29	1.71	187.43	189.14	-35.58
CVT	0.10	+2.65	0.84	15.36	16.20	+2.92
PH	0.05	+1.32	0.36	15.93	16.29	+1.46
Cam-U	3.73	98.68	52.88	328.41	381.29	68.81

and dataflows including WS, IS and OS. The hardware characteristic of Cambricon-U is summarized in Table 1.

Cambricon-U: Other than RIM, the remaining parts in Cambricon-U are implemented in Verilog RTL. The power and area are synthesized using Synopsys Design Compiler and Placed&Routed using Synopsys IC Compiler II under the FreePDK 45nm technology node [41] with Nominal process corner, VTG cells, and 1.1V voltage. The Cambricon-U has a 64×64 unary PE array operating at 400MHz, the same as the baselines. We leverage SRAM buffers as the external memory of the acceleration core. The SRAM energy and area are modeled using CACTI7 [3]. The sizes of the SRAM buffers are selected according to the MLPerf-Tiny benchmarks [4], with 64 KB for input buffer, 64 KB for output buffer and 512KB for weight buffer.

uSystolic: For the uSystolic baselines, we extend the uSystolic-Sim [55] simulator to support input-stationary and output-stationary dataflows. For the fairness of the comparison, we also set the uSystolic baselines to have a 64×64 PE array, with the same capacity of SRAM buffers as Cambricon-U. The number of banks and the line-width of each SRAM buffer is selected to minimize the average energy consumption for each coding scheme, bitwidth and dataflow. The frequency of uSystolic is 400 MHz, the same as [55]. The hardware power and area are obtained in the same way as Cambricon-U.

7.1.3 Benchmarks. We evaluate Cambricon-U on the MLPerf-Tiny [4] benchmarks. The benchmarks consist of four DNN models towards different fields. The first model, DS-CNN [58], is a depthwise-separable CNN for keyword spotting. The second model, MobileNet [18], is a lightweight CNN model for visual wake words. The third model, ResNet [16], is a tiny residual CNN model which takes the data format of CIFAR-10 dataset [26] as input for image classification. The last model, DeepAuto [25], is a multi-layer fully-connected network for anomaly detection. The accuracy of these benchmarks is evaluated using an open-source unary computing simulator, UnarySim [54].



(a) RIM

(b) Skew cell

Figure 10: The layout of (a) RIM and (b) Skew cell.

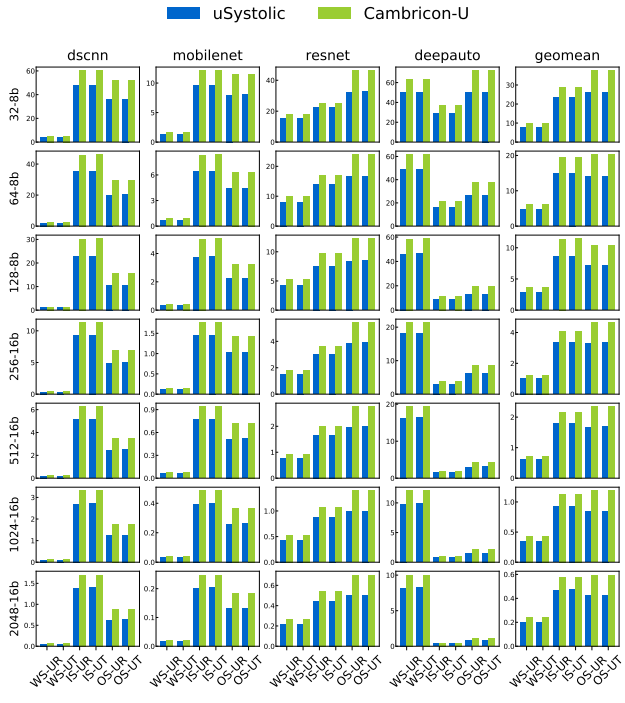


Figure 11: The energy efficiency (10^2 samples/J)

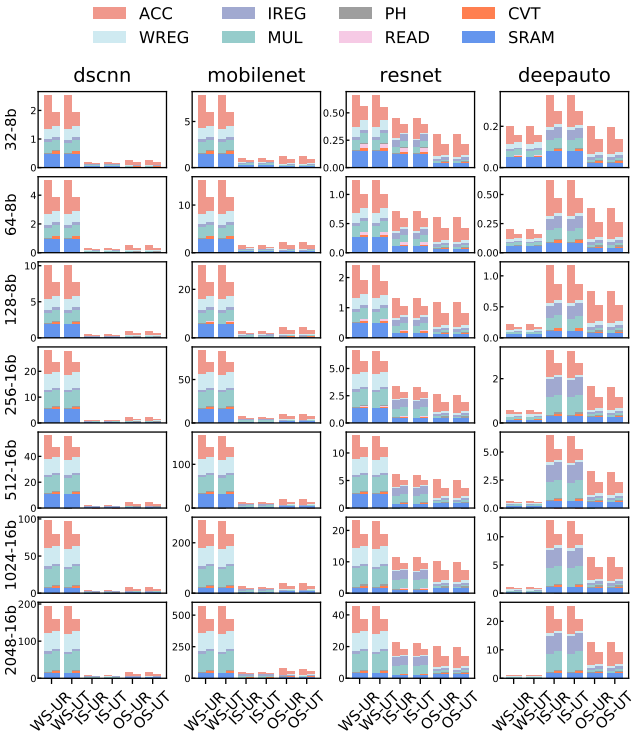


Figure 12: The energy breakdown (mJ). The left and right bar of each pair of bars are uSystolic and Cambricon-U, respectively.

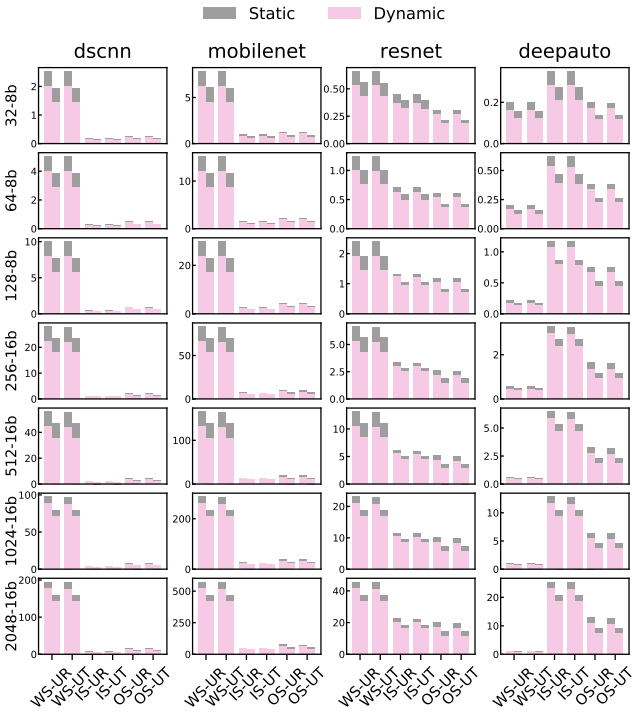


Figure 13: The energy breakdown of static and dynamic energy (mJ). The left and right bar of each pair of bars are uSystolic and Cambricon-U, respectively.

7.2 Energy

Figure 11 shows the energy efficiency improvement of Cambricon-U over uSystolic baselines under various configurations. The horizontal axis is marked in the format *Dataflow-Coding* where WS-UR means WS dataflow and Rate-coding as an example. The vertical axis is marked in the format *(MAC Cycles)-(bitwidth)* where 32-8b means 32 MAC cycles and 8-bit precision. Note that multiple MAC cycles can be supported by a single precision because of early termination. Overall, the average energy efficiency of Cambricon-U is 1.18-1.45× over uSystolic. The average improvement of energy efficiency in WS, IS and OS is 18%-28%, 20%-33% and 38%-45%, respectively. The improvement of OS is the highest because in OS dataflow, the accumulation occupies higher ratio of total energy. The results also demonstrate that RIM is applicable to versatile unary systolic array architectures with various coding-schemes and dataflows. Moreover, Cambricon-U outperforms the original uSystolic with WS dataflow 1.18-1.28×, 2.86-4.06× and 2.96-4.79× on average in WS, IS and OS, respectively. Although WS dataflow is more suitable for fully-connected layers, as the deepauto model exhibits the highest energy efficiency on WS dataflows, it has the lowest energy efficiency for the other three CNN models. The energy efficiency of WS dataflow for dscnn and mobilenet is extremely low because of the severe under-utilization of PEs when executing depthwise-separable convolution layers. As a result, the WS dataflow has the lowest energy efficiency on average.

To further study the influence of RIM, we breakdown the energy cost in Figure 12 and 13. The ratio of accumulation energy is reduced from 32%-70% to 16%-50%. The OS dataflow baselines experience higher energy efficiency improvement than other dataflows because of the higher ratio of accumulation energy, which is 64%-70% compared to 32%-47% and 36%-53% in WS and IS, respectively. The ratios of accumulation energy in Cambricon-U are reduced to 16%-26%, 18%-30% and 44%-50% in WS, IS and OS, respectively. Moreover, we find that the energy consumed by accumulation is reduced (-18%∼-36%) more than the additional energy cost (+2.3%∼+3.8%) of the converters. The converters in Cambricon-U only occupies 2.7%-5.1% of total energy in all the evaluated architectures and benchmarks. The low energy overhead of converters demonstrates that the converters have been efficiently shared by multiple PEs. The ratios of read-out energy of RIM in WS, IS and OS architectures are mostly 7.2%, 10.1% and 0.1%, respectively. The results show that the read-out energy of RIM is also much lower than the RIM increment energy. In addition, the reason of the low fraction of SRAM energy consumption is that the long MAC cycles reduce the number of SRAM banks. For instance, in OS dataflow, a 64×64 PE array only needs two banks for input and weight buffers, thus reducing the energy cost of the crossbars in SRAM buffers. Moreover, as Figure 13 shows, Cambricon-U mainly reduces the dynamic energy of uSystolic. Overall, RIM can reduce the accumulation energy cost with minimal additional energy overhead.

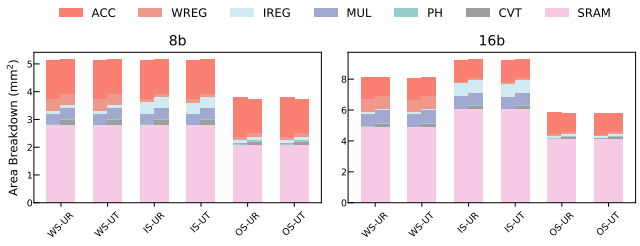


Figure 14: The area breakdown. The left and right bar of each pair of bars are uSystolic and Cambricon-U, respectively.

7.3 Area

Figure 14 shows the area breakdown of the evaluated architectures. The area overhead is only -1.9%∼+0.48% for 8-bit architectures (the two multiplication inputs are 8-bit signed numbers) and -1.2%∼+0.77% for 16-bit architectures. For 8-bit architectures, the RIMs and PEs reduce 3.5%-5.2% of the total area, and converters and PH introduce 2.7%-3.9% and 0.1%-0.8% of the total area, respectively. For 16-bit architectures, the RIMs and PEs reduce 2.0%-3.4%, and the converters and PH introduce 1.7%-2.5% and 0.1%-1.0% of the total area, respectively. RIM incurs roughly lower relative area change to the 16-bit architectures in that the 16-bit architectures require as twice bitwidth as the 8-bit architectures in both the acceleration core and external memory, thus the area is larger than that of 8-bit architectures. The results demonstrate the effectiveness of the converter to avoid doubling external memory capacity by transforming skew numbers to binary numbers before writing the output feature maps to external memory. Additionally, the low area cost is partly due to the large area of the SRAM, which occupies 54%-71% of the uSystolic baselines. The high fraction of SRAM area is due to the simplified unary PE, which has only 30%-56% of the area of a binary PE. However, even for the acceleration core only, the area overhead is only -4.2%∼+1.0% for 8-bit architectures and -4.1%∼+1.9% for 16-bit architectures. The results show that even for the acceleration core, the area overhead is still acceptable.

7.4 Accuracy

Figure 15 shows the top-1 inference accuracy of Cambricon-U, FP32 and INT-n (n-bit quantized binary model). Please note that the bits of INT-n is the 2-based logarithm of unary cycles (shown in the x-axis) plus a sign bit. For example, INT-8 is aligned with 128 unary cycles. As Cambricon-U is based on uSystolic, they are identical regarding the accuracy.

8 DISCUSSION

Real-world Evaluation. We evaluate Cambricon-U on four larger ImageNet [10] models, namely MobileNetV1 [18], SqueezeNetV1 [21], DenseNet [19, 20], and EfficientNetB0 [51]. These NN models are widely used in real-world scenarios. To fit the models, SRAM buffers are extended to 12MB, and the PE array is scaled to 256×256. The results show that Cambricon-U has 1.22×-1.47× energy efficiency, -3.2%∼+0.5% total chip area, and -7.6%∼+1.2% acceleration core area over uSystolic. Overall, the improvements are in line with MLPerf-Tiny models.

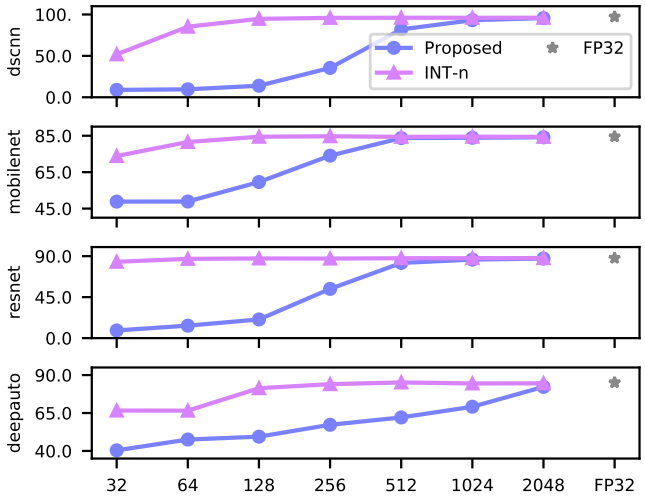


Figure 15: The Top-1 accuracy of the four DNN models.

RIM in Other Unary Architectures. We apply RIM to BISC-MVM [49] and DPS [48] architectures. Both are HUB architectures with binary accumulators. With RIM, the energy efficiency is improved by 34%-41% and 28%-37%, respectively.

Comparison with Other Potential Solutions. A trivial potential solution is buffering a short window (4-bit) of a unary bitstream and using bit-counting logic to reduce the frequency of accumulation. As shown in Table 2, the bit-counting solution is still worse than RIM for two reasons. First, the buffering and counting introduce overheads. Second, the accumulator is required to accumulate 3-bit increments instead of one bit. The results show the supremacy of the proposed RIM accumulator.

Comparison with Binary Design. We compare Cambricon-U with an INT-8 binary accelerator. The binary design consumes 2.1× energy and 2.2× area over Cambricon-U, as the binary PE has a large binary multiplier.

Comparison of Performance. Cambricon-U incurs negligible overhead on performance, as conversion is pipelined with buffering and computing. The latency of the Converter is fixed (1.93ns) regardless of bit length, thus only accounting for ∼1% of the pipeline latency.

ISA Support. Since Cambricon-U is based on uSystolic, they share a similar ISA. Specifically, each binary instruction is augmented

Table 2: Comparison of accumulators in uSystolic, bit-counting, and RIM-based.

	Static Power (uW)	Dynamic Power (uW)	Total Power (uW)	Area (um^2)
uSys.	3.21	91.10	94.31	349.26
Bitc.	4.23	70.83	75.06	499.03
RIM.	0.42	45.75	46.17	340.50

with a new field indicating the MAC cycles. In addition, a configuration instruction is utilized to set the LREG in Converters just before executing each layer.

Limitations. Limitations mainly exist on excessively shrinking the size of the systolic array in Cambricon-U. A too small systolic array (e.g., 2×2 PEs) would undermine the reuse of converters, thus suffering from their extra overhead. However, typically, the size of the PE array exceeds 16×16 (e.g., 128×128 in TPU [23], 32×32 in Cambricon-Q [59], and 96×96 in FSD [5]) to ensure computational parallelism, thereby prominently diluting the overhead of converters.

9 RELATED WORK

In general, dedicated unary computing hardware can be categorized into Fully Streaming Unary (FSU) or Hybrid Unary-Binary (HUB) architectures [55].

9.1 FSU Architecture

The FSU architecture fixes the entire model structure of a specific DNN in the circuit, fully utilizing the parallelism of the target DNN model [24, 32, 43, 54, 56]. Kim *et al*. presented error mitigation techniques to reduce MAC cycles for higher performance [24]. SC-DCNN [43] proposed various stochastic computing function extraction blocks such as inner-product and pooling, which can be carefully selected to minimize energy and area. Yu *et al*. proposed stochastic ReLu and Max functions for high accuracy and energy efficiency [56]. HEIF [32] proposed optimizations of function blocks and weight storage for high energy efficiency. uGEMM [54] proposed novel multipliers that support both rate-coding and temporal coding schemes. SASCHA [45] proposed a sparse SC PE to exploit the sparsity of DNNs. A new encoding method called Extended Stochastic Logic (ESL) [7] has been proposed to address the range limitation of unary bitstreams, and was adopted by Chen *et al*. and Liu *et al*. to improve the accuracy of the targeted DNNs [8, 34]. Although FSU architectures avoid buffering unary bitstreams by accumulating, thus achieving high performance [43, 54], it can neither be versatile for different DNN configurations nor scalable for large-scale DNNs like AlexNet [27]. As a result, FSU architectures are often evaluated on LeNet-5 [28] or small hand-written neural networks with MNIST [11] or CIFAR-10 [26] dataset. Compared to FSU architectures, Cambricon-U is more general and scalable to support versatile real-life DNNs with high energy efficiency.

9.2 HUB Architecture

Unlike FSU, the HUB architecture is designed to support general DNN structures or configurations through tiling [31, 50, 55]. Sim *et at*. proposed a tile-parallel binary-interlaced architecture for scalable stochastic CNNs [50]. HBUBB [14] leveraged the HUB architecture to realize a complete ResNet-18 on FPGA. Lee *et al*. proposed a stochastic-binary hybrid design for efficient near sensor computing. Also, several works exploit the data reuse and accuracy of HUB architectures [9, 31, 44]. Moreover, temporal-coding scheme has also been adopted to reduce RNG overhead and improve accuracy and performance [17, 48, 49, 55]. These works are orthogonal to our research. However, the generality and scalability come at the cost of lower power efficiency. During the calculation process, the unary bitstream (including hundreds even thousands of bits) transferred back and forth between tiles needs to be repeatedly accumulated into a large bitwidth binary number for buffering, thus undermining the power efficiency. Cambricon-U innovatively integrates a novel Random Increment Memory (RIM) which accumulates unary bitstreams to skew numbers, which can accumulate one bit by activating at most three bits, thus addressing the power bottleneck of the HUB architectures.

10 CONCLUSION

We propose a systolic random increment memory architecture, namely Cambricon-U, which features efficient accumulation to buffer unary bitstreams. Cambricon-U integrates a unary systolic array with RIMs, which accumulates unary bitstreams from PEs to *skew numbers*, whose increment only activates at most three bits (instead of all bits), thus enabling power-efficient accumulation. Experiments demonstrate the ability of Cambricon-U to improve energy efficiency 1.18-1.45× over versatile unary systolic array baselines, with the accumulation power reduced by 51%, breaking through the power bottleneck of unary computing.

ACKNOWLEDGMENTS

This work is partially supported by the National Key R&D Program of China (under Grant 2022YFB4501601), the NSF of China (under Grants 62102398, 62222214, 62102399, U22A2028, U19B2019), CAS Project for Young Scientists in Basic Research (YSBR-029) and Youth Innovation Promotion Association CAS.

REFERENCES

[1] Armin Alaghi and John P. Hayes. 2013. Exploiting correlation in stochastic circuit design. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 39–46. https://doi.org/10.1109/ICCD.2013.6657023

[2] Armin Alaghi and John P. Hayes. 2013. Survey of Stochastic Computing. *ACM Trans. Embed. Comput. Syst.* 12, 2s, Article 92 (may 2013), 19 pages. https://doi.org/10.1145/2465787.2465794

[3] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (jun 2017), 25 pages. https://doi.org/10.1145/3085572

[4] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf Tiny Benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).

[5] Pete Bannon, Ganesh Venkataramanan, Debjit Das Sarma, and Emil Talpes. 2019. Computer and Redundancy Solution for the Full Self-Driving Computer. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. 1–22. https://doi.org/10.1109/HOTCHIPS.2019.8875645

[6] B.D. Brown and H.C. Card. 2001. Stochastic neural computation. I. Computational elements. *IEEE Trans. Comput.* 50, 9 (2001), 891–905. https://doi.org/10.1109/12.954505

[7] Vincent Canals, Antoni Morro, Antoni Oliver, Miquel L. Alomar, and Josep L. Rosselló. 2016. A New Stochastic Computing Methodology for Efficient Neural Network Implementation. *IEEE Transactions on Neural Networks and Learning Systems* 27, 3 (2016), 551–564. https://doi.org/10.1109/TNNLS.2015.2413754

[8] Kun-Chih Chen and Chi-Hsun Wu. 2021. High-Accurate Stochastic Computing for Artificial Neural Network by Using Extended Stochastic Logic. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS51556.2021.9401418

[9] Zhiyuan Chen, Yufei Ma, and Zhongfeng Wang. 2022. Hybrid Stochastic-Binary Computing for Low-Latency and High-Precision Inference of CNNs. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 7 (2022), 2707–2720. https://doi.org/10.1109/TCSI.2022.3166524

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[11] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[12] J.A. Dickson, R.D. McLeod, and H.C. Card. 1993. Stochastic arithmetic implementations of neural networks with in situ learning. In *IEEE International Conference on Neural Networks*. 711–716 vol.2. https://doi.org/10.1109/ICNN.1993.298642

[13] Amr Elmasry, Claus Jensen, and Jyrki Katajainen. 2011. Two Skew-Binary Numeral Systems and One Application. *Theory of Computing Systems* 50 (2011), 185–211.

[14] Sayed Abdolrasouol Faraji, Gaurav Singh, and Kia Bazargan. 2019. HBUNN - Hybrid Binary-Unary Neural Network: Realizing a Complete CNN on an FPGA. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 156–163. https://doi.org/10.1109/ICCD46524.2019.00027

[15] B. R. Gaines. 1967. Stochastic Computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (Atlantic City, New Jersey) *(AFIPS '67 (Spring))*. Association for Computing Machinery, New York, NY, USA, 149–156. https://doi.org/10.1145/1465482.1465505

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[17] Reza Hojabr, Kamyar Givaki, SM. Reza Tayaranian, Parsa Esfahanian, Ahmad Khonsari, Dara Rahmati, and M. Hassan Najafi. 2019. SkippyNN: An Embedded Stochastic-Computing Accelerator for Convolutional Neural Networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs.CV]

[19] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. 2019. Convolutional Networks with Dense Connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[21] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360* (2016).

[22] Hongwu Jiang, Xiaochen Peng, Shanshi Huang, and Shimeng Yu. 2019. CIMAT: A Transpose SRAM-Based Compute-in-Memory Architecture for Deep Neural Network on-Chip Training. In *Proceedings of the International Symposium on Memory Systems* (Washington, District of Columbia, USA) *(MEMSYS '19)*. Association for Computing Machinery, New York, NY, USA, 490–496. https://doi.org/10.1145/3357526.3357552

[23] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[24] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. 2016. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2897937.2898011

[25] Yuma Koizumi, Yohei Kawaguchi, Keisuke Imoto, Toshiki Nakamura, Yuki Nikaido, Ryo Tanabe, Harsh Purohit, Kaori Suefusa, Takashi Endo, Masahiro Yasuda, and Noboru Harada. 2020. Description and Discussion on DCASE2020 Challenge Task2: Unsupervised Anomalous Sound Detection for Machine Condition Monitoring. arXiv:2006.05822 [eess.AS]

[26] A. Krizhevsky and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto* (2009).

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (may 2017), 84–90. https://doi.org/10.1145/3065386

[28] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1, 4 (1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[29] Peng Li, David J. Lilja, Weikang Qian, Kia Bazargan, and Marc D. Riedel. 2014. Computation on Stochastic Bit Streams Digital Image Processing Case Studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 3 (2014), 449–462. https://doi.org/10.1109/TVLSI.2013.2247429

[30] Shuangchen Li, Alvin Oliver Glova, Xing Hu, Peng Gu, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2018. SCOPE: A Stochastic Computing Engine for DRAM-Based in-Situ Accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture* (Fukuoka, Japan) *(MICRO-51)*. IEEE Press, 696–709. https://doi.org/10.1109/MICRO.2018.00062

[31] Tianmu Li, Wojciech Romaszkan, Sudhakar Pamarti, and Puneet Gupta. 2021. GEO: Generation and Execution Optimized Stochastic Computing Accelerator for Neural Networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 689–694. https://doi.org/10.23919/DATE51398.2021.9473911

[32] Zhe Li, Ji Li, Ao Ren, Ruizhe Cai, Caiwen Ding, Xuehai Qian, Jeffrey Draper, Bo Yuan, Jian Tang, Qinru Qiu, and Yanzhi Wang. 2019. HEIF: Highly Efficient Stochastic Computing-Based Inference Framework for Deep Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 8 (2019), 1543–1556. https://doi.org/10.1109/TCAD.2018.2852752

[33] Siting Liu and Jie Han. 2017. Energy efficient stochastic computing with Sobol sequences. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 650–653. https://doi.org/10.23919/DATE.2017.7927069

[34] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. 2018. A Stochastic Computational Multi-Layer Perceptron with Backward Propagation. *IEEE Trans. Comput.* 67, 9 (2018), 1273–1286. https://doi.org/10.1109/TC.2018.2817237

[35] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. 2021. A Survey of Stochastic Computing Neural Networks for Machine Learning Applications. *IEEE Transactions on Neural Networks and Learning Systems* 32, 7 (2021), 2809–2824. https://doi.org/10.1109/TNNLS.2020.3009047

[36] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2014. Race Logic: A hardware acceleration for dynamic programming algorithms. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 517–528. https://doi.org/10.1109/ISCA.2014.6853226

[37] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. 2020. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors* 20 (2020), 2533.

[38] Eugene Wimberly Myers. 1983. An Applicative Random-Access Stack. *Inf. Process. Lett.* 17 (1983), 241–248.

[39] M. Hassan Najafi, David J. Lilja, Marc Riedel, and Kia Bazargan. 2017. Power and Area Efficient Sorting Networks Using Unary Processing. In *2017 IEEE International Conference on Computer Design (ICCD)*. 125–128. https://doi.org/10.1109/ICCD.2017.27

[40] M. Hassan Najafi, David. J. Lilja, Marc D. Riedel, and Kia Bazargan. 2018. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 8 (2018), 1471–1480. https://doi.org/10.1109/TVLSI.2018.2822300

[41] NCSU. [n. d.]. FreePDK45. [EB/OL]. https://eda.ncsu.edu/freepdk/freepdk45/ Accessed April 20, 2023.

[42] Weikang Qian and Marc D Riedel. 2010. Synthesizing logical computation on stochastic bit streams. *submitted to Communications of the ACM* (2010).

[43] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. 2017. SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 405–418. https://doi.org/10.1145/3037697.3037746

[44] Wojciech Romaszkan, Tianmu Li, Rahul Garg, Jiyue Yang, Sudhakar Pamarti, and Puneet Gupta. 2022. A 4.4–75-TOPS/W 14-nm Programmable, Performance- and Precision-Tunable All-Digital Stochastic Computing Neural Network Inference Accelerator. *IEEE Solid-State Circuits Letters* 5 (2022), 206–209. https://doi.org/10.1109/LSSC.2022.3200064

[45] Wojciech Romaszkan, Tianmu Li, and Puneet Gupta. 2022. SASCHA—Sparsity-Aware Stochastic Computing Hardware Architecture for Neural Network Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4169–4180. https://doi.org/10.1109/TCAD.2022.3197503

[46] Wojciech Romaszkan, Tianmu Li, Tristan Melton, Sudhakar Pamarti, and Puneet Gupta. 2020. ACOUSTIC: Accelerating Convolutional Neural Networks through Or-Unipolar Skipped Stochastic Computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 768–773. https://doi.org/10.23919/DATE48585.2020.9116289

[47] A. Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv: Distributed, Parallel, and Cluster Computing* (2018).

[48] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. 2018. DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks. In

2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). 1–6. https://doi.org/10.1109/DAC.2018.8465700

[49] Hyeonuk Sim and Jongeun Lee. 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062290

[50] Hyeonuk Sim, Dong Nguyen, Jongeun Lee, and Kiyoung Choi. 2017. Scalable stochastic-computing accelerator for convolutional neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 696–701. https://doi.org/10.1109/ASPDAC.2017.7858405

[51] Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946 [cs.LG]

[52] Georgios Tzimpragos, Advait Madhavan, Dilip Vasudevan, Dmitri Strukov, and Timothy Sherwood. 2019. Boosted Race Trees for Low Energy Classification. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 215–228. https://doi.org/10.1145/3297858.3304036

[53] Di Wu, Jingjie Li, Zhewen Pan, Younghyun Kim, and Joshua San Miguel. 2022. UBrain: A Unary Brain Computer Interface. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 468–481. https://doi.org/10.1145/3470496.3527401

[54] Di Wu, Jingjie Li, Ruokai Yin, Hsuan Hsiao, Younghyun Kim, and Joshua San Miguel. 2020. UGEMM: Unary Computing Architecture for GEMM Applications. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 377–390. https://doi.org/10.1109/ISCA45697.2020.00040

[55] Di Wu and Joshua San Miguel. 2022. uSystolic: Byte-Crawling Unary Systolic Array. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 12–24. https://doi.org/10.1109/HPCA53966.2022.00010

[56] Joonsang Yu, Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. 2017. Accurate and Efficient Stochastic Computing Hardware for Convolutional Neural Networks. In *2017 IEEE International Conference on Computer Design (ICCD)*. 105–112. https://doi.org/10.1109/ICCD.2017.24

[57] Aidyn Zhakatayev, Sugil Lee, Hyeonuk Sim, and Jongeun Lee. 2018. Sign-Magnitude SC: Getting 10X Accuracy for Free in Stochastic Computing for Deep Neural Networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465807

[58] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2018. Hello Edge: Keyword Spotting on Microcontrollers. arXiv:1711.07128 [cs.SD]

[59] Yongwei Zhao, Chang Liu, Zidong Du, Qi Guo, Xing Hu, Yimin Zhuang, Zhenxing Zhang, Xinkai Song, Wei Li, Xishan Zhang, Ling Li, Zhiwei Xu, and Tianshi Chen. 2021. Cambricon-Q: A Hybrid Architecture for Efficient Training. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 706–719. https://doi.org/10.1109/ISCA52012.2021.00061