2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

SCOPE: A Stochastic Computing Engine for DRAM-based In-situ Accelerator

Shuangchen Li* Alvin Oliver Glova* Xing Hu* Peng Gu* Dimin Niu† Krishna T. Malladi†
Hongzhong Zheng† Bob Brennan† Yuan Xie*
University of California, Santa Barbara* Samsung Semiconductor Inc†
{shuangchenli, aomglova, huxing, yuanxie}@ece.ucsb.edu*, {dimin.niu, k.tej, hz.zheng}@ssi.samsung.com†

***Abstract*—Memory-centric architecture, which bridges the gap between compute and memory, is considered as a promising solution to tackle the memory wall and the power wall. Such architecture integrates the computing logic and the memory resources close to each other, in order to embrace large internal memory bandwidth and reduce the data movement overhead. The closer the compute and memory resources are located, the greater these benefits become. DRAM-based in-situ accelerators [1] tightly couple processing units to every memory bitline, achieving the maximum benefits among various memory-centric architectures. However, the processing units in such architectures are typically limited to simple functions like `AND/OR` due to strict area and power overhead constraints in DRAMs, making it difficult to accomplish complex tasks while providing high performance.**

In this paper, we address the challenge by *applying stochastic computing arithmetic to the DRAM-based in-situ accelerator*, targeting at the acceleration of error-tolerant applications such as deep learning. In stochastic computing, binary numbers are converted into stochastic bitstreams, which turns integer multiplications into simple bitwise `AND` operations, but at the expense of larger memory capacity/bandwidth demands. Stochastic computing is a perfect match for the DRAM-based in-situ accelerators because it addresses the in-situ accelerator's low performance problem by simplifying the operations, while leveraging the in-situ accelerator's advantage of large memory capacity/bandwidth. To further boost the performance and compensate for the numerical precision loss, we propose a novel Hierarchical and Hybrid Deterministic (H^2D) stochastic computing arithmetic. Finally, we consider quantized deep neural network inference and training applications as a case study. The proposed architecture provides $2.3\times$ improvement in performance per unit area compared with the binary arithmetic baseline, and $3.8\times$ improvement over GPU. The proposed H^2D arithmetic contributes $11\times$ performance boost and 60% numerical precision improvement.

I. Introduction

Bridging the gap between compute and memory is the key to addressing both the "memory wall" and "power wall" challenges. On one hand, the bandwidth and latency of the system memory scales much slower than the computing unit performance [2]. On the other hand, data movement is much more expensive than computing itself - the data transfer between hosts and off-chip memory consumes two orders of magnitude more energy than a floating point operation [3], and technology scaling cannot solve this problem [4].

This work was supported in part by Samsung Semiconductor Inc, NSF 1730309, 1719160, 1500848, and SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory (CRISP).

Memory-centric architecture appears to be a promising solution to these challenges. It redesigns both the compute and memory and puts them as close together as possible to embrace local memory's latency/bandwidth benefits and to reduce data movement overhead. There are two representative types of memory-centric architectures. One is the *memory-rich processor*, which packs as much memory as possible with the conventional processors or accelerators. Increasing on-chip memory capacity is a straightforward yet effective approach. For example, Google's TPU [5], a neural network accelerator, utilizes 24MB SRAM as on-chip scratchpad memory. However, due to the large SRAM/eDRAM memory cell [6], [7], the memory capacity is limited, thus the data movement overhead still remains a problem. Interposer-based 3D stacked High-bandwidth Memory (HBM) [8]–[10] is another memory-rich processor approach. For example, the latest GPUs [11] have 16GB HBMs as device memory. Google's TPU2 [12] also packs 64GB HBM2 memory to address the memory bottleneck for TPU1. However, the performance of such interposer-connected dense memory is not competitive with on-chip local memory. For example, the data access energy for the HBM is 4pJ/bit [13], whereas that for the eDRAM in DaDianNao [14] is about 0.07pJ/bit [15]. Overall, designing an efficient yet large memory for memory-rich processors is challenging. The other type of memory-centric architecture is referred to as *compute-capable memory*, which is also known as processing-in-memory (PIM) architecture. It puts the processing unit either inside the commodity DRAM core or adjacent to the memory core, e.g., in a logic die, stacked with the DRAM die through through-silicon vias. However, system memory is highly optimized for low cost, and therefore the processing unit added to the memory has tight area and power constraints. As a result, the performance of the compute-capable memory is limited. For example, NeuralCube [16] offers only 132 GOPs whereas GPUs can reach 40 TOPs [11].

The DRAM-based in-situ accelerator [1] is proposed to address the challenges faced by both the memory-rich processors and the compute-capable memory architecture. *This in-situ accelerator is different from conventional PIM architectures.* It is a standalone accelerator instead of part of system memory, so it is highly optimized for speed instead of low cost. DRAM, with its smaller memory cells ($6F^2$), allows for large on-chip memory capacity. Furthermore, it tightly combines the compute and memory by attaching a simple processing unit

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00062

to each memory bitline (BL), and leverages the large number of BLs inside memory for massive parallelism. However, due to the inefficiency of building processing logic circuits with the DRAM technology, it only supports simple bitwise operations (NOR) and bitwise shifts [1], [17]–[19]. In order to support complex computations such as additions, it breaks those instructions into multiple bitwise Boolean logic operations and processes them one by one. This scheme takes hundreds of cycles for multiplication (MUL) instructions, which raises the challenge for high performance.

To address this challenge, we adopt stochastic computing arithmetic for the DRAM-based in-situ accelerator. Stochastic computing is an approximate computing method that has been studied for decades [20]. Stochastic computing converts a MUL to a simple bitwise AND operation, dramatically reducing the complexity. It reduces the MUL's latency on the DRAM-based in-situ accelerator by 48× since the latter can do very long vector bitwise operations in parallel. However, the benefit comes at the cost of *data explosion*, since typically stochastic computing represents an n-bit fixed point integer by the probability of the appearance of "1" in a 2^n-bit bitstream, demanding larger memory bandwidth. Note that this challenge would offset the performance improvement when stochastic computing is adopted in a conventional Von Neumann architecture, but not for a DRAM-based in-situ accelerator, which tightly couples large memory with compute units.

In this paper, we propose *SCOPE*, the StoChastic cOmPuting Engine for DRAM-based in-situ accelerators. *SCOPE* tailors and adapts the stochastic computing arithmetic to such in-situ accelerator architecture. It converts the complex MULs into simple AND operations, thus dramatically reducing latency. In return, the in-situ architecture offers large on-chip memory with wide internal bandwidth to address the problem of long bitstreams in stochastic computing, which makes a perfect match between the arithmetic and the architecture. To further overcome stochastic computing challenges (i.e., exponentially long bitstreams, low throughput, and unpredictable numerical precision loss), we propose a novel Hierarchical and Hybrid Deterministic (H^2D) improved arithmetic, introducing three techniques. First, the hierarchical method separately converts the binary number's MSBs part and LSBs part into two shorter stochastic bitstreams, in order to reduce the total bitstream length. Second, we propose a hybrid representation with both a dense binary representation and a compute-friendly stochastic representation, which further reduces the bitstream length. Third, we develop a deterministic approach for the stochastic number generator, significantly improving the numerical precision and reducing the area overhead. Even more importantly, it makes the error reproducible for debugging purposes. The contributions of this paper are summarized as follows:

- We propose the idea of applying stochastic computing to the DRAM-based in-situ accelerator architecture to synergistically reinforce the strengths and address the weaknesses of both paradigms.
- We demonstrate *SCOPE*, the holistic architecture design that supports stochastic computing in the in-situ accelerator.
- We propose a novel Hierarchical and Hybrid Deterministic (H^2D) stochastic computing arithmetic, providing better performance, lower area overhead, and better numerical precision.
- We evaluate CNN/RNN inference and CNN training tasks on *SCOPE* as a case study and compare *SCOPE* with state-of-the-art solutions.

II. Background

This section briefly describes the background of stochastic computing, the DRAM-based in-situ accelerator architecture, and deep learning applications.

Stochastic Computing (SC): trades precision and representation density for simpler logic design and lower power. It has been proposed as an alternative low power computing method [21], and is widely applied to image/signal processing, control systems, or general purpose computing [20], [22]. Stochastic computing uses stochastic bitstreams to represent and compute. The probability of the appearance of "1" in the bitstream ($\{x_i\}$) represents the value of the binary number (X), as shown in Equation (1) and example of in Equation (3).

$$X\ (Binary) \rightarrow \{x_i\}\ (Stoch.),\ X = P(x_i = 1), \tag{1}$$

$$X \cdot Y = P(x_i = 1) \cdot P(y_i = 1) = P(x_i = 1\ \&\ y_i = 1). \tag{2}$$

Stochastic computing simply AND two bitstreams to calculate a multiplication (MUL). The resulting bitstream's probability of the appearance of "1" gives the result of multiplication, as shown in Equation (2) and the example in Equation (4). To calculate addition (ADD), stochastic computing uses a multiplexer to evenly select bits from two operand bitstreams.

$$X = \frac{3}{6}\ (Binary) \rightarrow \{x_i\} = \{0,1,0,1,1,0\}\ (Stoch.), \tag{3}$$

$$Y = \frac{2}{6}\ (Binary) \rightarrow \{y_i\} = \{0,0,1,1,0,0\}\ (Stoch.),$$

$$X \cdot Y = \frac{1}{6}\ (Binary) \rightarrow \{x_i \& y_i\} = \{0,0,0,1,0,0\}\ (Stoch.). \tag{4}$$

There are three major components of stochastic computing. They are (1) a stochastic number generator (SNG), which converts binary data to stochastic bitstreams, (2) logic gates, which calculate MULs and ADDs, and (3) a popcount (PC) unit, which converts the result bitstreams back to binary numbers. For the first part, conventionally, a random number generator (RNG) is adopted, and the binary number is used as the threshold to generate bitstream. It takes 2^n cycles[1] to convert an n-bit binary into a 2^n-bit stochastic bitstream. To reduce energy consumption and improve numerical precision, linear-feedback shift register (LFSR) based SNG method was proposed [23]. Parallel SNGs [24] were proposed based on LFSR to further reduce the conversion latency. For the second part, logic gates, existing work [25], [26] has shown that binary addition is preferred over stochastic additions to obtain better accuracy. This work follows this conclusion and only runs MUL operations in the stochastic domain. For the third part, PC counts the number of "1"s in the bitstream. An

[1] Unless otherwise stated, we use 2^n-bit bitstream to represent n-bit binary.

approximate PC (APC) [27] is proposed to sacrifice result accuracy for reduced energy consumption and latency.

DRAM-based In-situ Accelerator [1]: is a memory-centric architecture, which tightly combines compute and memory resources. It is built with DRAM technology in order to achieve GB-level large on-chip memory capacity. In this architecture, processing units are attached to each memory BL in order to fully exploit the internal memory bandwidth. Due to the large area overhead incurred when making logic gates using DRAM technology [28], this architecture only uses simple Boolean logic gates (AND, OR, etc.) as processing units. It can compute these BL logic operations similar to a vector machine (like AVX [29] instructions). For example, it can compute thousands of AND operations in parallel, thus exploiting massive parallelism available in the memory. In order to support general purpose computing (e.g., ADD instructions), it breaks these instructions down into multiple Boolean logic operations and then runs them in order. Because the compute resources are placed as close as possible to the data storage, this architecture achieves largest gains over other memory-centric architectures. It was demonstrated that this in-situ accelerator can achieve 8.8× performance per unit area and 1.2× energy efficiency improvements compared with other ASICs [1].

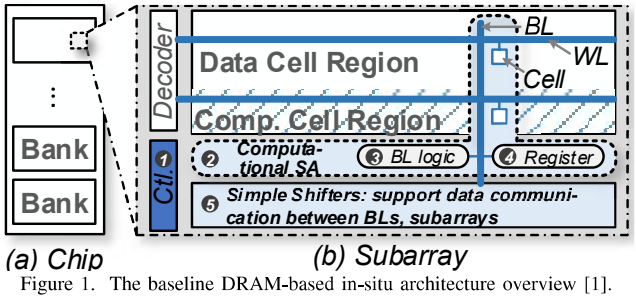

Figure 1. The baseline DRAM-based in-situ architecture overview [1].

An architecture overview is shown in Figure 1. The in-situ accelerator inherits the basic bank-subarray structure from commodity DRAM, so that it can still provide dense on-chip memory. Meanwhile, lightweight micro-architecture and circuit modifications are made to support computing. Apart from decoders and memory cells, which are the same as in commodity DRAM, five extra components are added as follows. (1) Controllers (❶) function as instruction decoders. They break down the instructions (like ADDs) into Boolean logic operations such as AND and then generate addresses and μ-operations accordingly. (2) Computational Sense Amplifiers (SAs, ❷) are designed for each BL. Each of them contains logic gates (❸) and a data register (❹) to support the BL-level bitwise operations[2]; (3) A network of shifters (❺) is added to support data movement across BLs. (4) Data movement hardware has multiple levels of hierarchy. First, inside each subarray, data movement is carried out by shifters. Next, the row-clone [32], [33] method is adopted to move data between banks. In order to process a Boolean logic operation, two (or multiple) source operand rows are activated and the result row is directly produced by the computational SA. The architecture runs the Boolean logic operations many times to accomplish general instructions such as ADD, while using the shifters for the carry-in propagation. Multiple independent subarrays can work in parallel [34] to improve performance. Low latency DRAM techniques are also adopted [33] for higher performance.

Note that the in-situ architecture is different from conventional PIM architecture. This hardware is a standalone accelerator like GPGPUs or Automata [35] integrated by PCIe, instead of a system memory. Thus, it is optimized for high performance rather than low cost. The DRAM on the in-situ accelerator has its private addressable scratchpad memory space, instead of being part of the global system memory space. This means that the accelerator is not limited by main memory constraints, e.g., power, thermal, data organizations, and cache coherency.

III. Motivation

In this section, we introduce the motivation and the key ideas of this work. We describe the challenges of DRAM-based in-situ architecture, and show how adopting stochastic computing can help solve those challenges. Furthermore, the problems of stochastic computing arithmetic and features of the in-situ accelerator which can address these problems are highlighted.

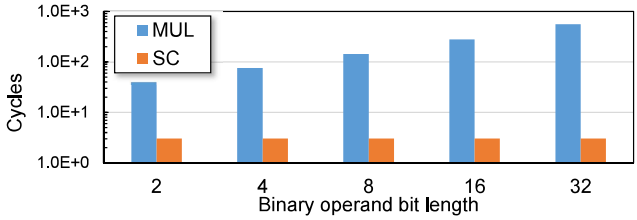

Figure 2. Latency (cycles per MUL) comparison of binary and stochastic computing in DRAM-based in-situ accelerator architecture.

Limitation of DRAM-based In-situ Accelerators: Building computing units in DRAM processes presents several limitations. Different from a logic process, the DRAM process is optimized for high density and low leakage power, so building logic gates in the DRAM process is not efficient (80% area and 22% performance overhead [28]). Therefore, the in-situ architecture can only build simple bitwise logic gates such as AND gates. In order to calculate MULs and ADDs, simple bitwise logic operations are performed repeatedly. The key problem is the slow MUL operation. As shown in Figure 2 (blue bars), an 8-bit multiplication takes 143 cycles, and the cycle count increases exponentially with the operand's bit-length (x-axis). This severely degrades the performance, especially the latency. Consequently, applications that can be used with DRAM-based in-situ accelerators are limited to those mainly having ADD operations.

Opportunities and Challenges of Stochastic Computing: The multiplication operation becomes a single bitwise AND operation in stochastic computing. The in-situ accelerator computes thousands of AND operations in parallel, taking

[2] Note that there are multiple methods to implement the logic functions in the computational SA [18], [30], [31]. In this work, we adopt the *1T1C-mixed* method proposed by Li *et al.* [1].

only 3 cycles. As shown in Figure 2 (orange bars), the 8-bit MUL latency is reduced 47.33× by using stochastic computing. The stochastic computing not only boosts the performance, but also enables adoption of the architecture by more applications. For example, previously, DRAM-based in-situ processors could only accelerate binary weight neural networks to avoid MUL operations, whereas when stochastic computing is adopted, they can support more applications like neural network training [36].

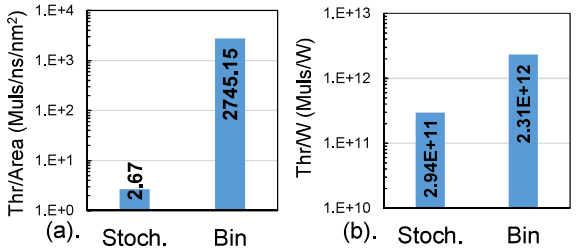

Figure 3. Comparison of (a) performance per area and (b) performance per Watt for 10-bit binary multiplier and 1024-bit stochastic computing multiplier.

However, adopting stochastic computing arithmetic is not straightforward. There are many *challenges*. First, the throughput is degraded, although the MUL latency is reduced. Second, the area and power overheads to support stochastic computing are also downsides. Figure 3 shows the comparison between the stochastic computing and binary arithmetic[3] in terms of 10-bit multiplier throughput normalized by (a) area and (b) power. Both of them have dedicated registers for data buffering. Stochastic computing computes in parallel (1024 ANDs in one cycle), representing the architecture of the in-situ accelerator. Unfortunately, the stochastic computing case only provides 0.1% throughput/area and 12.7% throughput/power of the conventional binary datapath case.

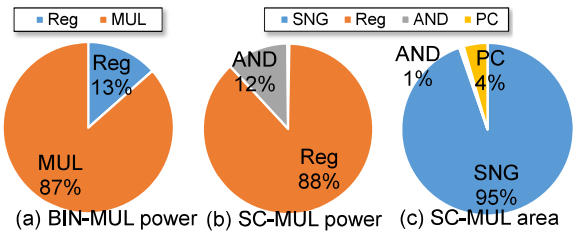

Figure 4. (a) Binary MUL Power breakdown, (b) stochastic computing MUL power breakdown, and (c) stochastic computing MUL area breakdown.

There are two reasons behind the low throughput problem. First, data representations in stochastic computing are very inefficient. An n-bit binary is represented with 2^n-length bitstream in the stochastic domain. The required data storage capacity and bandwidth exponentially increases, demanding more on-chip buffer. As shown in Figure 4(a) and (b), 88% of the power in the stochastic computing circuit is spent on data buffering, whereas it is only 13% in the conventional binary case. Second, stochastic computing incurs large area overhead for the supporting circuits, e.g., the SNG and the PC. As shown in Figure 4(c), 95% area is dedicated to the SNG [24]. Previous work has also shown that stochastic computing increases energy consumption by 3× compared with binary data path due to the SNG overhead [37].

[3]For the SC-multiplier, we design one-cycle fully parallel circuit synthesized with 45nm FreePDK, integrating parameters from state-of-the-art SNG [24] and PC [27].

Running stochastic computing on *the DRAM-based in-situ accelerator relieves these problems*. Unlike conventional accelerators, which use low density SRAM or eDRAM, the in-situ accelerator has a large amount of DRAM-based on-die memory space, thus it provides high energy efficiency. For example, the DRAM-based in-situ processor [1] spends 33% less energy for buffering 1Mb data than conventional processors using SRAM [15]. Therefore, stochastic computing's requirement for more data buffering is met. In addition, all computing units operate at the BL level, offering ultra high bandwidth so that the stochastic computing can run fully parallel. As a result, *the in-situ accelerator architecture and stochastic computing perfectly match each other. Their advantages compensate each other's disadvantages.*

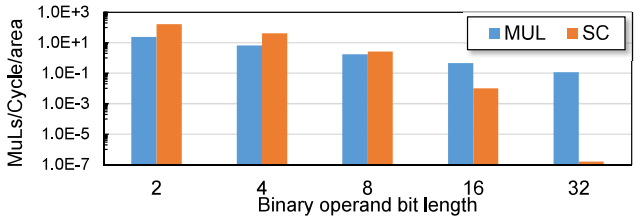

Figure 5. Throughput/Area (MULs per cycle per area unit) comparison of binary and stochastic computing MUL in DRAM-based in-situ accelerator architecture.

Towards A Better Stochastic Computing Arithmetic:

Even when stochastic computing is adopted, certain challenges still remain, although the nature of the in-situ architecture partially solves some issues. Figure 5 shows a comparison between binary MUL operations and stochastic computing MUL operations using the in-situ accelerator. Different from the latency in Figure 2, this data shows the throughput results normalized by the resource area used. Stochastic computing provides limited throughput benefit only if the operand bit length is less than 8. There are three reasons. **Challenge-1:** The bitstream length needs to be reduced. The long bitstream requires large storage and takes more computing resources. This problem is highlighted in Figure 3. These resources could have otherwise been used for more computing parallelism but are wasted to support long bitstreams. **Challenge-2:** The area overhead of the supporting hardware, i.e., the SNG and the PC, as shown in Figure 4(c). **Challenge-3:** stochastic computing's numerical precision needs improvement, which is pointed out as a major challenge in the field [20]. In addition, the error incurred by the stochastic computing would be better to be deterministic, so that the system error could be predictable and controllable. The error should also be reproducible to allow for system debugging. In summary, *a better stochastic computing arithmetic with reduced bitstream length, simplified support circuitry, and improved precision is needed to address these problems.*

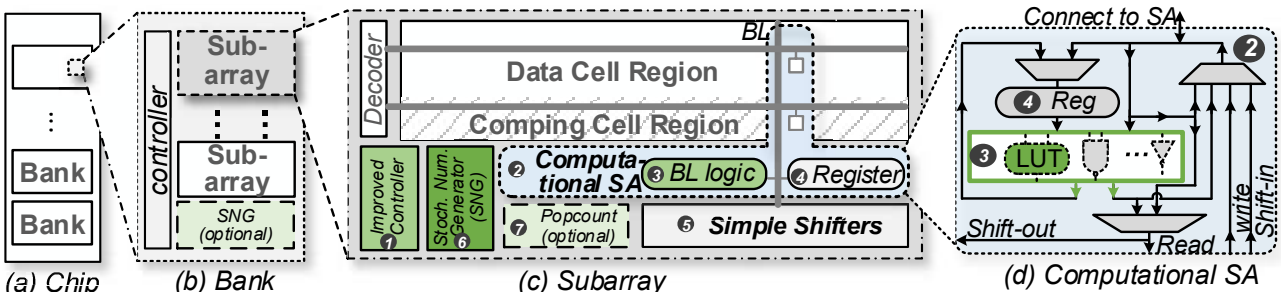

Figure 6. *SCOPE*'s architecture overview (Additional hardware to support stochastic computing is colored green).

IV. SCOPE Architecture

In this section, we describe the *SCOPE* architecture. We first show how stochastic computing is tailored and adapted to the DRAM-based in-situ accelerator architecture, and then we introduce a series of hardware designs to support stochastic computing.

A. Customizing Stochastic Computing

The original stochastic computing arithmetic is tailored to better fit with the DRAM-based in-situ accelerator. First, only MULs is performed with stochastic computing, while ADDs still uses the binary data path. This is done to avoid error accumulation, even though this incurs latency and power overheads for converting data back and forth between stochastic bitstreams and binary. Previous work [25], [26] have also shown that stochastic computing-based ADD seriously degrades the application-level computing accuracy. Second, the unipolar stochastic computing [21], which is similar with the unsigned integers, is adopted with an extra bit as the sign-bit. The bipolar stochastic computing arithmetic (like signed integers) is not used to avoid incompatibility with the proposed H^2D method which will be introduced in Section V. Third, fully parallel stochastic computing is used, which is different from most of the conventional serial stochastic computing data path (e.g., taking 1024 cycles to compute with a 1024-bit bitstream), because the DRAM-based in-situ accelerator already provides massively parallel bitwise operation hardware.

B. Supporting Stochastic Computing

Figure 6 shows the overview of the *SCOPE* architecture. Compared with the baseline architecture shown in Figure 1, *SCOPE* includes additional supporting hardware, which are colored green. These include the controller, Stochastic Number Generator (SNG), popcount (PC) unit, and the logic gates on each BL. The whole stochastic computing workflow using *SCOPE* is described as follows. Step-1, after the instructions are read and consecutively decoded at the bank and the subarray level, SNG is used to convert the binary numbers into stochastic bitstreams, which are then stored along the memory row. Step-2, arithmetic operations are applied on these bitstreams using the bitwise operations provided by each BL. Step-3, the result bitstreams are converted back to binary integers using the PC. In the rest of this subsection, we describe *SCOPE*'s major components following the order of this working flow, and explain how they coordinate and support stochastic computing.

Binary-to-Stochastic Conversion: As shown in Figure 6-❻, SNG converts binary numbers into stochastic bitstreams. Conventionally, an energy-efficient parallel LFSR-based SNG [24] can be adopted, generating 32-bit bitstream per cycle. However, after our stochastic computing arithmetic improvement is introduced later in Section V, an even simpler and faster SNG solution can be used in *SCOPE*. The improved arithmetic allows us to use a lookup table (LUT)-based SNG, which generates the whole bitstream in just one cycle.

We introduce two architecture-level optimizations for the conversion, in addition to the SNG hardware itself. First, we choose to convert binary data to stochastic bitstreams before storing them in the subarray. An alternative approach is storing binary data and not converting them until being issued for logic operations. The former approach stores stochastic bitstreams and occupies 2.5× more subarray capacity, but it saves 50176× conversion tasks since the data is heavily reused[4]. Since *SCOPE* offers large on-chip memory, the proposed approach is preferred to obtain better performance and energy efficiency. Second, we design one SNG for each subarray instead of each bank, so that binary data is not converted to stochastic bitstreams until the inter-subarray data transfer is done. This design choice reduces data movement overhead by transferring dense binary data instead of the large stochastic bitstreams. Otherwise, moving 1KB data takes 2.5× more latency and energy.

Arithmetic Operation: Arithmetic operations are carried out with BL logic operations in computational SAs (Figure 6(c)-❷). Figure 6(d) shows the components of the computational SA, which mainly include a set of logic gates (e.g., `AND` in ❸) and a register (❹). Three multiplexers (MUXs) are used to select the inputs of the register, the inputs of the logic gates, the restore data source, and the shift data source.

MUL is simplified as an `AND` operation in the stochastic domain. *SCOPE* needs three cycles to compute `AND`, as shown in Figure 7(a). In Cycle-1, row X is read and latched in the register, and this row is then restored (closed). In Cycle-2, row Y is read, and $X\&Y$ is calculated and latched, after which row Y is restored (closed) . In Cycle-3, the result row is activated, and is restored by the result value from the register. Note that we

[4] The result is for the weight of a convolution layer in CNN (VGG16's first layer) after H^2D optimizations.

intentionally protect the input operands since they are usually reused in the future. Since DRAM is read destructive, extra cycles is needed to restore original values (X and Y) to the operand rows. ADD in *SCOPE* is still calculated with binary data path. Figure 7(b) shows the working flow to compute a carry-save addition (CSA). This step takes four cycles. In Cycle-1, operand X is loaded. In Cycle-2 and 3, Y and Cin are read, intermediate results are calculated accordingly, and Sum is restored to the memory. In Cycle-4, carry-out Co is calculated and restored back to memory. Note that since operands in sum-reduction are rarely reused in the future, the steps that save operands are skipped, in order to improve performance.

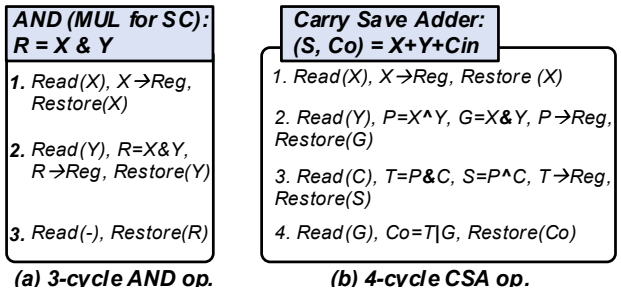

Figure 7. (a) Steps to perform MUL. (b) Steps for carry save addition.

Compared with the baseline architecture [1], we have two optimizations from the hardware design perspective. First, an optimized set of logic gates (Figure 6-❸) was selected so that *two* results of any operation can be obtained. For example, $X\&Y$ and $X\wedge Y$ can be computed simultaneously. This optimized two-output design improves addition operations by 1.75× with 19% area overhead, compared with single-output logic sets. Second, an LUT data path to support the improved H^2D stochastic computing arithmetic is added (described in Section V).

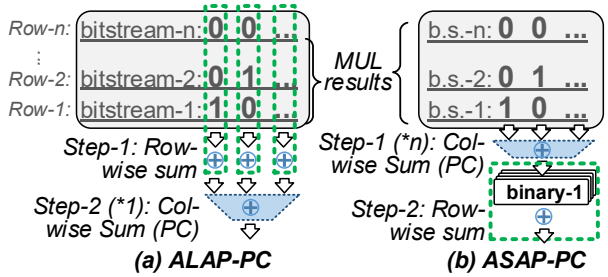

Figure 8. Calculating the sum of partial MUL results in MAC operations (a) PC as Late as possible, and (b) PC as Soon as possible.

Stochastic-to-Binary Conversion: After calculating MULs in the stochastic domain, we convert the data back to binary data before any further computations, in order to avoid error accumulation. We choose not to adopt the prevailing approximate popcount (APC) proposed in previous work [27] (Figure 6-❼). Although the APC reduces the popcount latency, it significantly decreases the numerical precision (see Figure 14). Moreover, APC requires designing extra hardware in *SCOPE*, which takes 11.71% area overhead per subarray. Therefore, we leverage the already existing column-wise addition operations in *SCOPE* for the conversion, instead of paying extra hardware overhead and suffering precision loss if APC is used.

In order to improve the performance of stochastic-to-binary conversion, we propose a method called ALAP-PC, which calculates the PC as late as possible. This is an effective optimization for vectored multiply-and-accumulate (MAC) operations, which are widely used in many applications such as those that require matrix multiplication. In such situations, there are two types of sum tasks. One is the sum of the bits within the stochastic bitstream to convert it back to binary data, i.e., the popcount. It requires addition across columns, since a bitstream is stored along in one memory row. The other type is the sum of all the MUL results. It requires addition across rows, since the results are stored in different rows. The key idea of the ALAP-PC is to perform the row-wise sum-reduction first, and the column-wise sum-reduction (PC) as late as possible. Figure 8 shows (a) the ALAP-PC and (b) the opposite, ASAP-PC, in which column-wise sum-reduction is done first and then row-wise sum-reduction is executed.

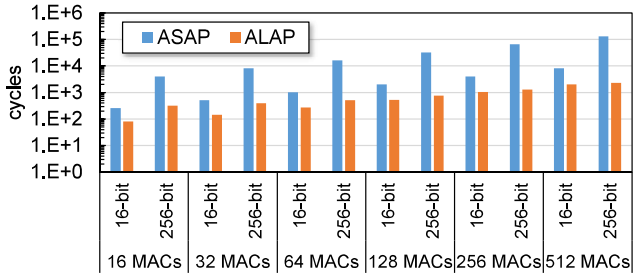

Figure 9. ASAP-PC vs. ALAP-PC.

Figure 9 shows the performance comparison between these two methods. The x-axis represents the number of MAC operations, with either 16-bit or 256-bit stochastic bitstream. The data shows that the ALAP-PC achieves up to 56× improvement over ASAP-PC. This is because row-wise sum-reduction can use CSA, which has no carry prorogation and can be calculated efficiently by lock-step bitwise operations within 4 cycles. On the other hand, the column-wise sum-reduction needs full adders (FAs), which takes 22 cycles for 8-bit operands. The ALAP-PC uses less of the inefficient column-wise sum-reduction operation and hence saves latency.

V. H^2D Arithmetic

The conventional stochastic computing arithmetic suffers from the exponential bitstream length and the numerical precision loss, as discussed in Section III. In this section, we describe our proposed improved stochastic computing arithmetic composed of three key techniques.

A. Hierarchical Stochastic Bitstreams

In order to reduce the length of stochastic bitstreams, the hierarchial method divides the binary data into two parts and separately converts these parts into stochastic bitstreams. As shown in Figure 10, a binary number is divided into the most significant bits (MSBs) part and the least significant bits (LSBs) part. Taking an 8-bit binary number as an example, the

MSBs-part is its higher 4 bits and the LSBs-part is its lower 4 bits. Instead of converting the whole n-bit binary data into (2^n-1)-bit stochastic bitstream, the proposed hierarchial method separately converts the $\frac{n}{2}$-bit MSB/LSB parts into two ($2^{\frac{n}{2}}$-1)-bit stochastic bitstreams (referred to as partial bitstreams). Consequently, the stochastic bitstream length is reduced by $2^{\frac{n}{2}-1}$. The MUL operation in the context of the hierarchical method is not a bitwise AND anymore, as shown in the lower right corner of Figure 10. It contains three partial bitstream MULs, one shift, and two addition operations. Note that it is unnecessary to consider the MUL of two LSB-parts because in stochastic computing, the result is truncated from 2n-bit to n-bit for an n-bit MUL.

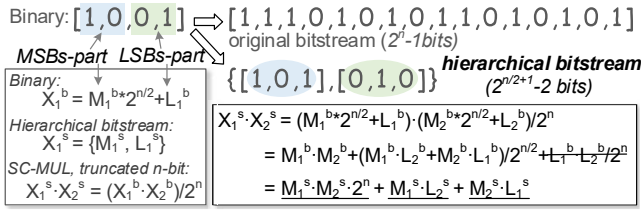

Figure 10. Hierarchical stochastic representation.

The benefit of the hierarchical method is the reduction of the bitstream length (8× for 8-bit integer), which then translates to throughput improvement. One drawback of this method is that it makes MUL more complex. For the 8-bit integer example, 6× more cycles are required. However, after considering the MUL overhead, it still brings 5.40× throughput improvement. Another downside is that it degrades the numerical precision because it introduces more error to the MSB-parts. However, we show that this disadvantage can be compensated by other methods, and we show it obtains 60% accuracy improvement when all these techniques are combined in Section VIII-C.

B. Hybrid Binary-Stochastic Bitstreams

Here, we discuss the proposed hybrid binary-stochastic data representation and arithmetic. As shown in Figure 11, the original stochastic bitstream is divided into groups with three bits. Then, binary representation is used inside each group, while a set of these groups makes up the hybrid binary-stochastic bitstream. To perform MUL operations on the hybrid bitstreams, 2-bit binary MUL is executed within each group, and then a series of the result groups forms as the result hybrid bitstream. The 2-bit MUL for each group can be implemented with simple bitwise operations, as shown in the lower part of Figure 11. Note that we choose 3-bit stochastic subsequence to group as a 2-bit binary instead of a longer subsequence. This is because a longer subsequence requires more-than-2-bit binary MUL operations, which results in larger latency and area overhead that may cancel out the benefit of a short bitstream.

The benefit of the hybrid representation is a 1.5× reduction of bitstream length since it condenses every 3-bit stochastic subsequence into 2-bit binary, which can lead to higher throughput and energy efficiency. In addition, the hybrid method improves the numerical precision by 33%, since part of the MUL calculation is performed in the accurate binary domain. The only drawback is the 2-bit binary MUL calculation. A custom LUT is designed for this operation as shown in Figure 6-❸. It takes one more cycle than the original 3-cycle stochastic MUL. However, hybrid representation still offers 11% MUL throughput improvement even after considering this overhead.

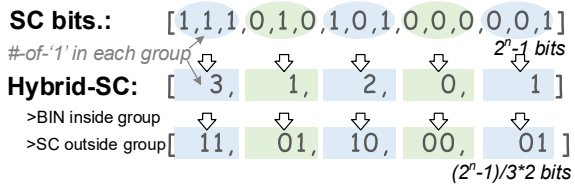

Figure 11. Hybrid binary-stochastic representation.

C. Deterministic SNG

Previous work [38], [39] have shown that the stochastic numbers are not necessarily completely randomized. Consequently, we propose a deterministic SNG. Instead of using conventional random number generator hardware (e.g., RNG or LFSR) for the SNG, we use a pre-programmed lookup table (LUT). As shown in Figure 12, one operand is converted into stochastic bitstream in a "0"s-"1"s-"0"s style. The number of "0"s before "1"s are preset as an offset, which is looked-up from the LUT. The number of "1"s is equal to the binary data. The other operand is converted into stochastic bitstream in a periodic style. In other words, "1"s are evenly distributed in the stochastic bitstream. For both of the operands, the number of "1"s are set to the original binary number. Designing the deterministic approach is easier in *SCOPE*, because the error propagation is limited by only computing one MUL operation in the stochastic domain and converting stochastic bitstreams back to binary immediately.

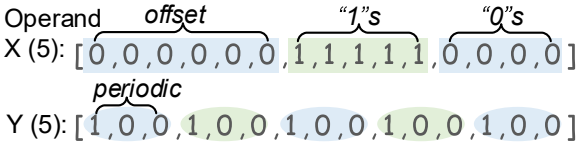

Figure 12. A deterministic approach for stochastic number generation.

The deterministic approach has two advantages. First, it simplifies the SNG as a simple LUT, thus it provides 53.2× area reduction. Second, it improves the stochastic computing numerical precision by 23%, as shown in Figure 14. Moreover, it makes stochastic computing error predictable and reproducible, which makes it possible to control the error and debug the program.

D. H^2D: Putting all together

The three techniques (*Hierarchial, Hybrid, and Deterministic*) are orthogonal to each other, so all of them can be applied together. We collectively name these techniques H^2D. For an 8-bit integer, H^2D can reduce the bitstream length by 12× and improve the throughput by 5.2×. The numerical precision is also improved by 60%, as will be shown in Section VIII-C.

VI. Discussions

In this subsection, we discuss *SCOPE*'s design choice, integration, interface, limitations, and applications.

Stochastic vs. Approximate MUL: Approximate MUL simplifies MUL circuit to trade numerical precision for high efficiency [40]. However, these circuit techniques cannot be applied for DRAM-based in-situ computing architecture because the overhead of the non-SIMD behavior of the approximate MUL offsets its efficiency benefits. To demonstrate this, we provide a rough comparison. A recent work on approximate MUL [40] provides 1.89× latency reduction and 2.64× throughput/area improvement with precision loss of 0.018 error-std. Our stochastic MUL outperforms this by offering 6.8× latency reduction and 4× throughput/area improvement, while with smaller precision loss of 0.011 error-std.

System integration: *SCOPE* is positioned as an accelerator or co-processor. It is integrated using the PCIe bus, similar to the TPU [5], GPUs [11], or Micron's Automata [35]. *SCOPE* has sufficiently large on-chip memory so that no additional memory (DRAM) is required. In addition, *SCOPE* can be scaled out by connecting multiple chips together, as in the case of multiple GPUs.

Programming interface: Although the programming interface is beyond the scope of this paper, here, we briefly discuss the hypothetical framework. We use a high level framework (such as Tensorflow) as the user interface, in which case the underlying *SCOPE* is transparent to the users. The integration of *SCOPE* and such framework is similar to the case of the integration of Tensorflow and TPU [5]. To this end, we will provide highly optimized operator-level libraries which is integrated to the computational graph generated by a framework and connects the operations in the graph to the back-end hardware, i.e., *SCOPE*. The optimized operator-level library consists of micro-kernels, which, for example, maps a convolution layer to the *SCOPE* with its instructions (similar to DRISA [1]).

Limitations: Although the proposed H^2D improves the numerical precision, it is still limited to applications that can tolerate approximate computing. In addition, the data is limited to fixed-point integer format. However, we argue that domain specific hardware design prefer to customize data width to optimize its performance. For example, TPU [5] uses 8-bit data, Micron's Automata [35] uses integers for pattern recognition, and an ReRAM-based neural network accelerator [41], [42] applies approximate computing as well. In the next paragraph, we will show a variety of applications that fall under this requirement. Note that for workloads having a small portion of task with floating point computing, we can always leverage the host CPU for help.

Target applications: Any approximate computing application can work on *SCOPE*. The preferred usage is for applications that are data intensive, have significant data parallelism, and can tolerate approximate computing. In this paper, we consider deep learning applications as a case study. However, broader application fields such as (1) image/signal processing, (2) bioinformatics [43], [44], (3) classic machine learning, and (4) graph analysis [45], [46] can also potentially be used with *SCOPE*. We leave the demonstration of other applications as our future work.

VII. A Case Study: Deep Learning

In this section, we consider CNN/RNN inference tasks and CNN training tasks as a case study.

A. Quantized Deep Learning

We tailor DNN for Stochastic Computing since they are well-known for being error tolerant. Data quantization [47]–[53] and compression [54] methods have been well studied. Even aggressively quantized CNN training using 1-bit weight, 2-bit activation, and 4-bit gradient shows tolerable recognition accuracy degradation [52]. Previous work have also evaluated applying stochastic computing to CNN inference [25], [26], [37], [55] and training [36].

We target edge or mobile application scenario. For inference, we target applications that require high accuracy such as self-driving vehicles. In these cases, ultra-low bit quantization is not preferred, e.g., Ternary-NN [56] has 2% precision drop on ResNet, but using 8-bit fixed-point (like *SCOPE*) achieves the same accuracy as floating point [57].

For training, we target online learning such as transfer/incremental learning [58] demanded by IoT devices due to the diversity of application scenarios and customers privacy protection [59]–[61]. Note that plenty of previous work have demonstrated feasibility of training with lower bit integers, as below. Chen *et al.* [62] uses binary data to present both weight and activation, while 12-bit data for gradient, achieving 11.5% error rate on CIFAR-10. Miyashita *et al.* [63] uses 5-bit, 4-bit, and 5-bit data to represent weight/activation/gradient, respectively. It achieves 6.21% error rate on CIFAR-10. Wu *et al.* [64] uses 2-bit, 8-bit, and 8-bit data to represent weight/activation/gradient, resulting in 5% top-1 accuracy loss on AlexNet. DoReFa-net [52] uses 1-bit, 2-bit, and 4-bit data to represent weight/activation/gradient. It also achieves 53% top-1 accuracy (3% loss from golden model) with AlexNet on ImageNet if applying all INT8 training.

B. Tailoring DNN for Stochastic Computing

In this case study, we run CNN/RNN inference task and CNN training task on *SCOPE* as follows. We apply the quantization DNN methods, but replace the integer MUL with stochastic computing methods. In the back-propagation process, the numerical error induced by stochastic computing is modeled as part of the quantization noise, which is taken into consideration during the gradient calculation.

SCOPE is able to support most of the layers and operations which can be implemented with INT8-MAC. For other operations, there are three options. One is to approximate the unsupported operation with MAC or shift operation. For example, Batch Normalization is calculated in *SCOPE* by approximating division and square-root with shift. The second method is to use the DRAM as a look up table. For example, activations like *tanh* can be calculated by this method. If neither of the above method works, the final method is to send the operation back to the CPU for computing, similar to the approach that TPU takes [5].

VIII. Experiments

In this section, after the description of the experimental setup, we demonstrate the evaluation of *SCOPE*. Then we evaluate the numerical precision improvement of the proposed H^2D method. Finally, we show the performance and energy evaluation when running deep learning applications on *SCOPE*.

A. Experimental Setup

The configuration is described as follows. *SCOPE* is evaluated under 22nm DRAM technology, and has 8Gb DRAM, which is split into 1024 banks, and these banks are further grouped into 64 groups. For each bank, there are 16 subarrays, and each subarray has 256 rows and 2048 columns, an SNG, and a 16-bit adder. We compare *SCOPE* with baselines described in Table I in the following experiments.

Table I
Baseline descriptions.

DRISA [1]	The DRAM-based in-situ acc. w/o stoch. comp.
***SCOPE*-vanilla**	The *SCOPE* w/o H^2D using original stoch. comp.
***SCOPE*-hyb** ***SCOPE*-hier**	*SCOPE* w/o the whole H^2D, both w/ the deterministic SNG but only w/ either *hybrid* or the *hierarchial*
***SCOPE*-H^2D**	*SCOPE* with the whole H^2D design **(our proposal)**
GPU	Pascal TITAN X [11] with 40.4 TOPs peak INT8 performance and 12GB GDDR5 device memory

To evaluate *SCOPE*'s circuit, we used a heavily-modified CACTI-3DD [65] that models the *SCOPE* architecture for the DRAM structures. We use FreePDK45 [66] technology to obtain the computational unit (Figure 6-❸) parameters since this logic process has similar pitch size with the 22nm DRAM's peripheral region [67]. We synthesize the computational unit with Design Compiler to extract the power, performance, and area parameters, and plug them into the CACTI-based model. The *SCOPE* runs at 124MHz and the detailed area and performance are reported in the next subsection. We also develop behavioral level simulators to evaluate *SCOPE*'s effective performance and energy consumption running given applications. The performance simulator first breaks an operation (e.g., a convolution layer) into *SCOPE* instructions to each subarrays. In this process, the parallelism, dependency, and resource allocation are taken into consideration. The overall performance/energy are finally accumulated from the performance counter implemented with each subarray in the performance simulator. Note that the GPU results are measured from dual Pascal TITAN-X with FP32 precision. We conservatively scale these results by 4× as *GPU-INT8* for fair comparison. We also conservatively exclude one third of the GPU board power consumption for cooling purpose. *SCOPE* does not need extra off-chip device memory due to its in-situ computing architecture, which is a major cause for its large area. To have a fair comparison, we include off-chip device memory as part of GPU's area, since *SCOPE* has the equivalent memory on-die.

B. Performance and Area Evaluation

Table II shows the comparison between baselines in terms of the MUL latency, the fused MUL-ADD peak throughput, chip area, and performance per unit area. We highlight three key observations from these results. First, simply adopting stochastic computing to the DRAM-based in-situ accelerator decreases the MUL latency by 47.6×. However, this degrades the throughput by 1.21× because of the long bitstream (see Section III). Second, hierarchical and hybrid methods both increase the throughput by reducing the bitstream length at a reasonable cost of longer latency and/or extra area overhead. These methods increase the performance per unit area of *SCOPE*-vanilla by 4.4× and 1.1×, respectively. Third, putting all the arithmetic optimizations together, *the* H^{2}D *shows* 4.2× *and* 6.16× *better performance and performance per unit area compared with the* SCOPE*-vanilla.*

Table II
Peak Performance Comparison.

	DRISA[a]	***SCOPE***				**GPU**
		vanilla	**hier**	**hybrid**	***H^{2}D***	**INT8**
MUL latency[b]	143	3	17	4	21	0.5
Peak TOPs[c]	1.65	1.36	5.98	1.55	7.08	40.4
Area (mm^2)	258.2	259.42	258.2	273.38		1631[d]
Peak GOPs/Area	6.39	5.24	23.16	5.67	**25.90**	24.76

[a] We re-evaluate DRISA [1] with 8Gb setup and integrating more accurate post-layout logic gates parameters; [b]Cycle; [c]INT8-fused MUL-ADD operation, including SNG/PC; [d]Normalized to 22nm, including 12GB DRAM.

Table III
Comparison with commodity DRAM.

Area (mm^2)		**Latency (ns)**		**Energy (nJ)**	
SCOPE	**DRAM**	***SCOPE***	**DRAM**	***SCOPE***	**DRAM**
273.38	61.73	8.02	40.35	0.11	2.13

Table III shows *SCOPE*-H^2D's area, latency, and energy, compared with commodity DRAMs with same capacity. As expected, *SCOPE*'s area is larger than the commodity DRAM. However, note that this work is targeted as a high performance accelerator rather than system main memory. Enabling wider computing capability and higher performance is a priority for this type of accelerator. One reason for the area overhead is the extra hardware, e.g., computational SA in Figure 6-❷. Another reason is the array reorganization. *SCOPE* adopts a large number of smaller yet less dense arrays with short BL and WL [33], which contributes to the much shorter latency and increased parallelism. Consequently, *SCOPE* is 5.03× faster with 19.36× less energy consumption.

Comparison with GPU and Accelerators: As shown in Table IV, *SCOPE* outperforms state-of-the-art GPU and accelerators. In particular, DaDianNao [14] is an accelerator with eDRAM design (an alternative to DRAM). However, it is not comparable to SCOPE since eDRAM cell is more than 10× larger than a DRAM cell. SC-Acc [26] is a stochastic computing-based accelerator design, and due to its memory limitation, it also cannot compete with *SCOPE*.

Note that we include external off-chip DRAM in total area consumption only for fair comparison. Like other PIM/NDP

architecture, *SCOPE* has both memory and computing on one chip, and no extra off-chip DRAMs are required. However, GPUs and other accelerators need extra off-chip DRAM which serves as device memory and is part of their cost. In addition, we only compare peak performance since most baseline's effective data is not available (except for GPU, which we show in the following section). We argue that *SCOPE* has better resource utilization than others due to its in-situ architecture, which avoids hitting the memory bandwidth roofline. Therefore, a better peak performance from *SCOPE* indicates more advantage when comparing effective performance.

Table IV
COMPARISON WITH OTHER WORK.

	GPU	NN-Accelerator			NDP		PIM		
	TITAN-X [11]	DaDian-Nao [14]	Eyeriss [68]	Stoch [26]	Neuro-Cube [16]	Tetris [69]	DRISA [1]	BufComp [70]	SCOPE
TOPs[a]	40.4	5.56	0.08	1.28	0.13	1.02	1.65	0.37	7.08
Core Area[b]	942	96	1.5	9.1	340	340	258	688	273
Ext Mem	12GB	2GB	1GB	1GB	N/A				
Area	741	124	62	62	0				
Total Area	1683	219	63	71	340	340	258	688	273
Perf/Area	24.0	25.4	1.3	18.1	0.38	3.0	6.4	0.53	25.9

[a] Normalized to INT8 MAC; [b] mm^2, Normalized to 22nm (50% less area per tech-node).

Comparison with NDP/PIM: As shown in Table IV, *SCOPE* also outperforms 3D-stacking based architectures such as NeuroCube [16] and Tetris [69]. Such architectures have to put their computing units on the logic die of an HMC/HBM stack. However, the logic die is already occupied with PHY and other support circuits. Thus, the area budget for the computing units is effectively limited.

In terms of 2D-based PIM architectures, most of the PIM work, such as Ambit [17] cannot even support MUL/ADD operations. Representative NVM-based PIM work like PRIME [41] performs analog computations which is beyond the scope of this paper (*SCOPE* only uses digital circuits) and Pinatubo [18] which doesn't support MUL/ADD operations. For PIM architectures with ADD/MUL support like Buffered-Compare [70], the performance is not comparable with *SCOPE*.

Justifying the area overhead: First, *SCOPE* is designed to be used as an accelerator instead of a replacement for the host memory. Therefore, although conventional memory design is optimized for area and cost, we optimize *SCOPE* for high performance. As a result, *SCOPE* uses 4.4× more area than commodity DRAM, but this enables transformation of commodity DRAM into an accelerator that can outperform GPU. In this regard, comparing our accelerator with commodity DRAM would be unfair. Second, *SCOPE* takes a similar approach with Micron's Automata [35] and DRISA [1] in building DRAM-based accelerators. Compared to the estimated Automata implementation [71], *SCOPE* can provide 1.1× more density. Third, other memory technologies with lower density cannot provide large enough on-chip memory capacity for the accelerator. Even the high-density eDRAM suffers from a much larger cell size ($60F^2$ - $80F^2$ [6], [7]) than DRAM ($6F^2$). Using DRAM, *SCOPE* provides 1GB on-chip memory per chip, whereas state-of-the-art GPU has only 34MB [72] and ASIC accelerator provides only 28MB to 34MB [5], [14]. Lastly, the area overhead does not worsen the leakage because it is mainly contributed by low latency design optimization instead of stochastic-computing support circuits. Moreover, *SCOPE* has a low refresh overhead. Our case study shows that all tasks are completed within 8ms, which means most of the rows have been read and restored within the refresh rate of 64ms.

Area/Lantecy breakdown: Figure 13 shows the breakdown of area and latency of *SCOPE*. First, the area breakdown shows that the computing logic gates and the shift circuits should have occupied 5% chip area. However, due to the DRAM design constraints, the narrow BL pitch forces the transistor layout to be folded which results in extra area overhead. Second, the latency breakdown shows that the extra hardware only takes 3%, while the majority of the latency are still spent on memory access operations.

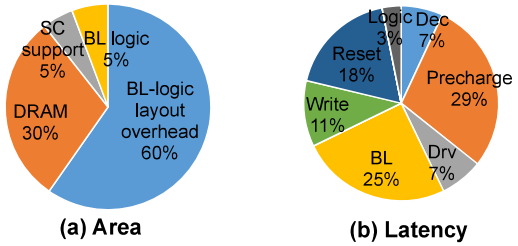

Figure 13. The area, latency breakdown of *SCOPE*.

C. Precision Evaluation

As one of the approaches to approximate computing, the numerical precision in *SCOPE* is very important. In this subsection, we evaluate *SCOPE*'s precision, and show how it can be improved by the proposed H^2D method. Then, we show how *SCOPE* impacts the accuracy of neural networks in our case study described in Section VII.

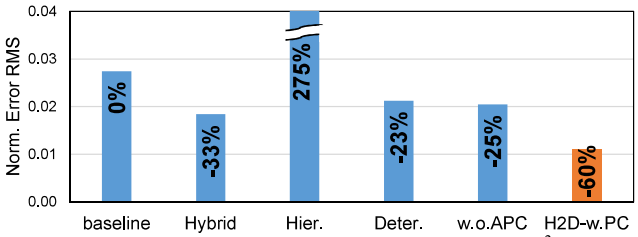

Figure 14. The error RMS of 8-bit MUL for the proposed H^2D method (normalized to range $[0,1]$).

Figure 14 shows the error root mean square (RMS) of all possible INT8 operand combinations. In this figure, the x-axis is different arithmetic configurations, in which the baseline is the vanilla stochastic computing (*SCOPE*-vanilla). Applying hybrid and deterministic methods alone (the second and fourth bar) reduces the error by 33% and 23%, respectively. By replacing the approximate APC used in previous work with the accurate PC (the fifth bar), an extra 25% error reduction is achieved. Even though the hierarchial method increases the error by 275% (the third bar), putting them together with the accurate PC, *H^2D improves the overall precision by 60% over baseline*.

In the experiment, we also evaluate the impact of numerical precision on deep learning applications. The stochastic computing-based CNN evaluation based on previous DNN quantization methods [51], [52]. Stochastic computing is simulated by adding noise to integer MULs according to the error RMS data from Figure 14. The numerical error induced by stochastic computing is then modeled as part of the quantization noise in the back-propagation process. In the CNN inference experiments, we conservatively quantize both weight and activation data to 8-bit integers while using floating-point gradient for the offline training. Then, we apply stochastic computing for all integer MULs in the feedforward process. In the CNN training experiments, in addition to the 8-bit weights and activation data, we also quantize gradients as 8-bit integers. Then, we apply stochastic computing for all integer MULs in both the feedforward and back-prorogation process. For this case study, we have evaluated LeNet [73] on MNIST [74]. It is difficult to examine a larger neural network, because random number generation required to simulate stochastic computing on current GPUs is inefficient, resulting in extremely long training periods. However, we observe that *SCOPE* offers even smaller accuracy loss compared with quantized neural network solutions, so we believe that the performance of *SCOPE* on larger scale neural networks is also similar with other neural network quantization work [51], [52].

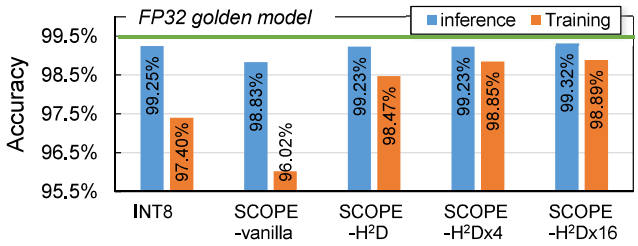

Figure 15. The stochastic computing based CNN inference and training recognition accuracy.

Figure 15 shows both the inference and training recognition accuracy for integer quantization, *SCOPE*-vanilla, and the proposed *SCOPE*-H^2D method. For inference (blue bars), first, we observe that H^2D-based CNN only has 0.27% accuracy degradation than the golden model (using FP32, 99.5% accuracy), and only 0.02% degradation compared with the integer quantization method. Second, the proposed H^2D method is better than *SCOPE*-vanilla by 0.4%, which is a even larger gap than that between H^2D and the golden model. For training (orange bars), we observe that although H^2D-based training has 1.03% accuracy degradation compared with the golden model, it is 1.07% and 2.45% better than the integer-based solution and the *SCOPE*-vanilla, respectively. As noted above, it is difficult to examine a larger neural network or dataset due to extremely long training time since the random number generation required to simulate stochastic computing on current GPUs is inefficient. However, previous work [52] reports training AlexNet on ImageNet using INT8 results in 53% top-1 accuracy, which is 3% accuracy loss from a golden model. Considering training with *SCOPE* produces smaller accuracy loss than INT8 training in Figure 15, we believe similar trend will appear in a larger dataset. Therefore, we estimate training AlexNet with *SCOPE* will result in no more than 3% accuracy loss. In addition, we observe that having longer stochastic bitstreams, which can help to improve the numerical precision, is not necessary in this situation. The last two bars show that 4× and 16× longer bitstreams only improve the accuracy by 0% and 0.09% in the inference experiment, and 0.38% and 0.42% in the training experiment.

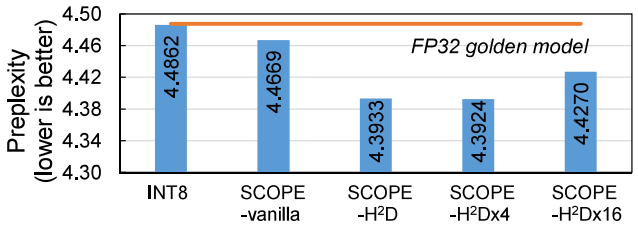

Figure 16. The stochastic computing based vanilla RNN inference for character-level language model.

We also evaluate vanilla RNN inference applications in the experiment, which justify that *SCOPE* can be widely adopted to various deep learning applications. The perplexity is the RNN accuracy metric, in which case lower perplexity is better. We still use 8-bit weights and activation data, and floating-point gradient, and we use stochastic computing for integer MULs in the feedforward processes. We use a vanilla RNN model with 3 layers and 256 neurons per layer running a character-level language model [75]. As shown in Figure 16, we observe that although the integer-based quantization method has a similar result as the golden model, both the *SCOPE*-vanilla and the proposed *SCOPE*-H^2D have a better result than the golden model. The *SCOPE*-vanilla has a 0.02-lower perplexity, and *SCOPE*-H^2D is even better with a 0.09-lower perplexity. In summary, *we have shown that* SCOPE *is effective for some deep learning applications.*

D. Evaluating DNN on SCOPE

In this subsection, we evaluate *SCOPE* with deep learning applications as a case study. We consider four benchmarks, as shown in Table V, in which all the CNN applications run on ImageNet [76] data set and the RNN runs on a character-level language model [75]. The batch size is set to 64 for all the experiments.

Table V
BENCHMARK DESCRIPTIONS.

vgg	VGG-16 [77], CNN inference
resnet	ResNet-152 [78], CNN inference
rnn	Vanilla RNN, 3-layer×256-neuron, inference
Alex-train	AlexNet [79], CNN training

Figure 17 shows the performance per unit area results, which are normalized to the *DRISA* baseline. On average, the proposed *SCOPE*-H^2D is 2.3× better than *DRISA* and 3.8× better than *GPU-INT*. Although *GPU-INT* provides high performance when its resource utilization is high, the data movement may offset its advantages. In addition, compared with the methods without H^2D or with partial H^2D,

SCOPE-H^2D always provides better performance. *SCOPE-H^2D* is 11.6×, 9.0×, and 1.1× better than the *SCOPE*-vanilla, *SCOPE*-hyb, *SCOPE*-hier, respectively. Moreover, the trend stays the same for each particular benchmark. The superior performance of the *SCOPE-H^2D* comes from both the bitstream length reduction and the simplified hardware.

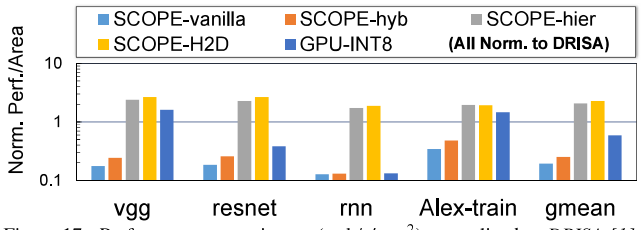

Figure 17. Performance per unit area (task/s/mm^2) normalized to *DRISA [1]*.

Figure 18 shows the energy efficiency (performance per Watt) results, which are normalized to the *DRISA* baseline. On average, the proposed *SCOPE-H^2D*'s energy efficiency is 4.4× better than the baseline *DRISA*, and is 1.7× better compared with *GPU-INT*. *SCOPE* performs better for memory-intensive inference benchmarks, e.g., *ResNet* and *RNN*, since data movement overhead is significantly reduced by storing data within its large on-chip memory. In addition, we also demonstrate the importance of the H^2D method. With the H^2D method, it provides 7.1× better energy efficiency than the *SCOPE*-vanilla. Compared with the partially adopted H^2D (*SCOPE*-hyb and *SCOPE*-hier) baselines, the *SCOPE-H^2D* is 5.4× and 1.2× better on average, and the same trend applies to every benchmark.

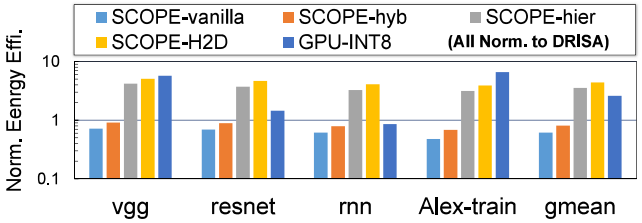

Figure 18. Energy efficiency (task/J) normalized to *DRISA [1]*.

IX. Related Work

In this section, we review related work in stochastic computing, processing-in-memory (PIM).

Stochastic Computing: Previous research have started to apply stochastic computing to neural network (NN) applications since the 90s [25], [26], [37], [55], [80]–[85]. However, the *SCOPE* is **a significant step forward**. From architecture perspective, *SCOPE* adopts an advanced in-memory computing architecture whereas all previous work are SRAM-based ASICs. Furthermore, *SCOPE* uses an optimized stochastic computing arithmetic with the unique H^2D method to complement the in-memory computing architecture. From application perspective, *SCOPE* evaluates both CNN-training and RNN-inference, whereas previous work only focus on CNN-inference tasks. Moreover, this work evaluates state-of-the-art large-scale NNs that previous work have not covered.

Processing-In-Memory Architecture: The key idea of PIM is to offload computing tasks to the processors on or near the main memory [17], [18], [41], [86]–[91]. The drawback of these architectures is that the area budget is limited [92]. As a result, the performance is lower than dedicated accelerators (e.g., Neurocube provides 132GOPs [16] whereas GPUs can reach 40TOPs [11]).

Different from all these PIM work, *SCOPE* is an accelerator design. *SCOPE* has large on-chip DRAM to tightly couple computing and memory resources, and is highly optimized for high performance computing, without the tight area overhead constraint for main memory. In addition, the techniques proposed by *SCOPE*, e.g., the H^2D, are *orthogonal* to the architecture (either PIM or accelerator). PIM architectures can also adopt these techniques to improve their performance and energy efficiency.

X. Conclusion

In this paper, we design *SCOPE*, a holistic architecture which adopts stochastic computing for the DRAM-based in-situ accelerator architecture, two paradigms that synergistically complement each other. Stochastic computing simplifies the complex MUL operations so that MULs can be efficiently calculated with the simple Boolean logic gates available in the DRAM-based in-situ accelerator. In return, the in-situ architecture offers large on-chip memory with wide internal bandwidth to address the problem of long bitstreams in stochastic computing. To further improve the performance and energy efficiency, we propose the H^2D arithmetic optimization methods. We evaluated the proposed architecture with a case study on deep learning applications. The experimental results show that *SCOPE* is 2.3× better than the DRAM-based in-situ accelerator baseline, and 3.8× better than GPU, in terms of area-normalized performance. The proposed H^2D arithmetic optimization improves the performance by 11.6× and boosts the energy efficiency by 5.4×. In addition, H^2D improves stochastic computing's numerical precision by 60% on average, which translates into accuracy improvement when adopted to deep learning applications.

References

[1] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in *International Symposium on Microarchitecture (MICRO)*, pp. 288–301, ACM, 2017.

[2] K. Rupp, "Cpu, gpu and mic hardware characteristics over time," 2013. https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/.

[3] O. Villa *et al.*, "Scaling the Power Wall: A Path to Exascale," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 830–841, IEEE Press, 2014.

[4] B. Shekhar, "Exascale Computing-A Fact or a Fiction?," *Technical talk at IPDPS*, 2013.

[5] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," pp. 1–12, 2017.

[6] G. Fredeman, D. W. Plass, A. Mathews, J. Viraraghavan, K. Reyer, T. J. Knips, T. Miller, E. L. Gerhard, D. Kannambadi, C. Paone, D. Lee, D. J. Rainey, M. Sperling, M. Whalen, S. Burns, R. R. Tummuru, H. Ho, A. Cestero, N. Arnold, B. A. Khan, T. Kirihata, and S. S. Iyer, "A 14 nm 1.1 Mb Embedded DRAM Macro With 1 ns Access," *IEEE Journal of Solid-State Circuits*, vol. 51, pp. 230–239, jan 2016.

[7] F. Hamzaoglu, U. Arslan, N. Bisnik, S. Ghosh, M. B. Lal, N. Lindert, M. Meterelliyoz, R. B. Osborne, J. Park, S. Tomishima, Y. Wang, and K. Zhang, "13.1 A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology," in *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 230–231, IEEE, feb 2014.

[8] D. U. Lee *et al.*, "A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 432–433, feb 2014.

[9] X. Hu, D. Stow, and Y. Xie, "Die stacking is happening," *IEEE Micro*, vol. 38, pp. 22–28, January 2018.

[10] S. Yin *et al.*, "Parana: A parallel neural architecture considering thermal problem of 3d stacked memory," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[11] "NVIDIA GPU," 2016. http://www.nvidia.com.

[12] J. Dean, " Recent Advances in Artificial Intelligence via Machine Learning and the Implications for Computer System Design," 2017.

[13] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained DRAM: Energy-efficient DRAM for Extreme Bandwidth Systems," in *International Symposium on Microarchitecture (MICRO)*, pp. 41–54, ACM, 2017.

[14] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *International Symposium on Microarchitecture (MICRO)*, pp. 609–622, IEEE, dec 2014.

[15] N. P. Muralimanohar, Naveen and Balasubramonian, Rajeev and Jouppi, "CACTI 6.0: A tool to model large caches," *HP Lab.*, pp. 22–31, 2009.

[16] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *International Symposium on Computer Architecture (ISCA)*, pp. 380–392, IEEE, jun 2016.

[17] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *International Symposium on Microarchitecture (MICRO)*, pp. 273–287, ACM, 2017.

[18] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. 53rd Annu. Des. Autom. Conf. - DAC '16*, 2016.

[19] W. Chen *et al.*, "A 16mb dual-mode reram macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 28.2.1–28.2.4, Dec 2017.

[20] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Transactions on Embedded Computing Systems*, vol. 12, pp. 1–19, may 2013.

[21] B. R. Gaines, "Stochastic computing," in *Proceedings of the spring joint computer conference*, 1967.

[22] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *Proceedings of the Annual Design Automation Conference*, 2015.

[23] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *Design, Automation & Test in Europe Conference & Exhibition*, pp. 1550–1555, IEEE, 2016.

[24] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Asia and South Pacific Design Automation Conference*, pp. 256–261, IEEE, jan 2016.

[25] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable stochastic-computing accelerator for convolutional neural networks," in *Asia and South Pacific Design Automation Conference*, pp. 696–701, IEEE, jan 2017.

[26] 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," *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 405–418, 2017.

[27] K. Kim, J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," in *International SoC Design Conference*, pp. 123–124, IEEE, nov 2015.

[28] Y.-B. Kim and T. W. Chen, "Assessing merged DRAM/Logic technology," *Integration, the VLSI Journal*, vol. 27, pp. 179–194, jul 1999.

[29] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, "Intel avx: New frontiers in performance improvements and energy efficiency," *Intel white paper*, vol. 19, p. 20, 2008.

[30] S. Li, D. Niu, K. Malladi, H. Zheng, B. Bernnan, and Y. Xie, "A DRAM-based Reconfigurable In-Situ Accelerator," tech. rep., 2017.

[31] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *Computer Architecture Letters*, vol. PP, no. 99, p. 1, 2015.

[32] V. Seshadri, M. A. Kozuch, T. C. Mowry, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, and P. B. Gibbons, "RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization," in *International Symposium on Microarchitecture (MICRO)*, 2013.

[33] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 568–580, IEEE, mar 2016.

[34] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *International Symposium on Computer Architecture (ISCA)*, pp. 368–379, IEEE Computer Society, 2012.

[35] Micron, "Micron Announces Development of New Parallel Processing Architecture." http://www.micronautomata.com/.

[36] S. Gupta, V. Sindhwani, and K. Gopalakrishnan, "Learning Machines Implemented on Non-Deterministic Hardware," *arXiv Prepr. arXiv1409.2620*, 2014.

[37] 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 *Proceedings of the Annual Design Automation Conference*, 2016.

[38] D. Braendler, T. Hendtlass, and P. O'Donoghue, "Deterministic bit-stream digital neurons," *IEEE Transactions on Neural Networks*, vol. 13, pp. 1514–1525, nov 2002.

[39] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Proceedings of International Conference on Computer-Aided Design*, 2016.

[40] S. Hashemi, R. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 418–425, IEEE, 2015.

[41] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *nternational Symposium on Computer Architecture (ISCA)*, vol. 44, pp. 27–39, jun 2016.

[42] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13, IEEE, mar 2016.

[43] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3d-reram based dna alignment accelerator architecture," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 59:1–59:6, ACM, 2018.

[44] J. Wang *et al.*, "Communication optimization on gpu: A case study of sequence alignment algorithms," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 72–81, May 2017.

[45] A. Basak, X. Hu, S. Li, S. M. Oh, and Y. Xie, "Exploring Core and Cache Hierarchy Bottlenecks in Graph Processing Workloads," in *IEEE Computer Architecture Letter*, IEEE, 2018.

[46] G. Li, G. Dai, S. Li, Y. Wang, and Y. Xie, "GraphIA: An In-situ Accelerator for Large-scale Graph Processing," in *The International Symposium on Memory Systems (MEMSYS)*, ACM, 2018.

[47] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *CoRR*, vol. abs/1603.05279, 2016.

[48] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *CoRR*, vol. abs/1609.07061, 2016.

[49] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating Deep Convolutional Networks using low-precision and sparsity," *CoRR*, vol. abs/1610.00324, 2016.

[50] F. Li and B. Liu, "Ternary Weight Networks," *CoRR*, vol. abs/1605.04711, 2016.

[51] J. Ott, Z. Lin, Y. Zhang, S. Liu, and Y. Bengio, "Recurrent Neural Networks With Limited Numerical Precision," *CoRR*, vol. abs/1608.06902, 2016.

[52] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *CoRR*, vol. abs/1606.06160, 2016.

[53] X. He *et al.*, "Joint design of training and hardware towards efficient and accuracy-scalable neural network inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2018.

[54] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," *CoRR*, vol. abs/1510.00149, 2015.

[55] B. Y. Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, "DSCNN: Hardware-Oriented Optimization for Stochastic Computing Based Deep Convolutional Neural Networks," in *Internetional Conference on Comupter Design*, 2016.

[56] A. Kundu, K. Banerjee, N. Mellempudi, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary residual networks," *arXiv preprint arXiv:1707.04679*, 2017.

[57] S. Migacz, "8-bit inference with tensorrt," in *GPU technology conference*, 2017.

[58] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36, 2012.

[59] M. Song, K. Zhong, J. Zhang, Y. Hu, D. Liu, W. Zhang, J. Wang, and T. Li, "In-Situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems," in *2018 IEEE InternatiOnal SympOsium On High PerfOrmance COmputer Architecture (HPCA)*, pp. 92–103, IEEE, 2018.

[60] O. Temam, "A shift towards edge machine-learning processing," in *IEEE International Solid-State Circuits Conference*, pp. 214–215, 2018.

[61] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1396–1401, IEEE, 2017.

[62] X. Chen, X. Hu, H. Zhou, and N. Xu, "FxpNet: Training a deep convolutional neural network in fixed-point representation," in *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 2494–2501, IEEE, 2017.

[63] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, 2016.

[64] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *CoRR*, 2018.

[65] Ke Chen, Sheng Li, N. Muralimanohar, Jung Ho Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 33–38, EDA Consortium, IEEE, mar 2012.

[66] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *International Conference on Microelectronic Systems Education (MSE)*, pp. 173–174, IEEE, 2007.

[67] M. Sung *et al.*, "Gate-first high-k/metal gate dram technology for low power and high performance products," in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 26.6.1–26.6.4, Dec 2015.

[68] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 367–379, IEEE Press, 2016.

[69] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 751–764, 2017.

[70] J. Lee and J. H. Ahn and K. Choi, "Buffered compares: Excavating the hidden parallelism inside DRAM architectures with lightweight logic," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1243–1248, 2016.

[71] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[72] Nvidia, "Volta GPU Architecture ." http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2017.

[73] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60, pp. 53–60, Perth, Australia, 1995.

[74] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs*, vol. 2, 2010.

[75] "Efficient, reusable RNNs and LSTMs for torch." https://github.com/jcjohnson/torch-rnn.

[76] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[77] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015.

[79] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[80] Young-Chul Kim and M. Shanblatt, "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1109–1118, 1995.

[81] J. Dickson, R. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *IEEE International Conference on Neural Networks*, pp. 711–716, IEEE, 1993.

[82] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *arXiv Prepr. arXiv1509.08972*, 2015.

[83] 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.

[84] H. Sim, S. Kenzhegulov, and J. Lee, "Dps: Dynamic precision scaling for stochastic computing-based deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 13:1–13:6, ACM, 2018.

[85] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-magnitude sc: Getting 10x accuracy for free in stochastic computing for deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 158:1–158:6, ACM, 2018.

[86] R. Nair *et al.*, "Active Memory Cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, pp. 17:1–17:14, mar 2015.

[87] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 126–137, IEEE, mar 2016.

[88] J. Ahn, S. Yoo, and K. Choi, "AIM: Energy-Efficient Aggregation Inside the Memory Hierarchy," *ACM Transactions on Architecture and Code Optimization*, vol. 13, pp. 1–24, oct 2016.

[89] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling techniques for gpu architectures with processing-in-memory capabilities," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, PACT '16, (New York, NY, USA), pp. 31–44, ACM, 2016.

[90] X. Tang, O. Kislal, M. Kandemir, and M. Karakoy, "Data movement aware computation partitioning," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, (New York, NY, USA), pp. 730–744, ACM, 2017.

[91] M. Xie *et al.*, "Aim: Fast and energy-efficient aes in-memory implementation for emerging non-volatile main memory," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 625–628, March 2018.

[92] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *nnual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–13, IEEE, oct 2016.