DRISA: A DRAM-based Reconfigurable In-Situ Accelerator

Shuangchen Li[1] Dimin Niu[2] Krishna T. Malladi[2] Hongzhong Zheng[2]
Bob Brennan[2] Yuan Xie[1]
[1]University of California, Santa Barbara [2]Samsung Semiconductor Inc.
{shuangchenli,yuanxie}@ece.ucsb.edu
{dimin.niu,k.tej,hz.zheng,bob.brennan}@ssi.samsung.com

ABSTRACT

Data movement between the processing units and the memory in traditional *von Neumann* architecture is creating the "memory wall" problem. To bridge the gap, two approaches, the memory-rich processor (more on-chip memory) and the compute-capable memory (processing-in-memory) have been studied. However, the first one has strong computing capability but limited memory capacity/bandwidth, whereas the second one is the exact the opposite.

To address the challenge, we propose *DRISA*, a DRAM-based Reconfigurable In-Situ Accelerator architecture, to provide *both* powerful computing capability and large memory capacity/bandwidth. *DRISA* is primarily composed of DRAM memory arrays, in which every memory bitline can perform bitwise Boolean logic operations (such as NOR). *DRISA* can be reconfigured to compute various functions with the combination of the functionally complete Boolean logic operations and the proposed hierarchical internal data movement designs. We further optimize *DRISA* to achieve high performance by simultaneously activating multiple rows and subarrays to provide massive parallelism, unblocking the internal data movement bottlenecks, and optimizing activation latency and energy. We explore four design options and present a comprehensive case study to demonstrate significant acceleration of convolutional neural networks. The experimental results show that *DRISA* can achieve 8.8× speedup and 1.2× better energy efficiency compared with ASICs, and 7.7× speedup and 15× better energy efficiency over GPUs with integer operations.

CCS CONCEPTS

• **Hardware → Dynamic memory**; • **Computer systems organization → Reconfigurable computing**; **Neural networks**;

KEYWORDS

DRAM, Accelerator, Neural Network

ACM Reference format:
Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, Yuan Xie. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 14 pages.
https://doi.org/10.1145/3123939.3123977

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3123977

1 INTRODUCTION

The increasing gap between the computational performance of the processors and the memory has created the "memory wall" problem [90], in which the data movement between the processing units and the memory is becoming the bottleneck of the entire computing system, ranging from cloud servers to end-user devices. For example, the data transfer between CPUs and off-chip memory consumes two orders of magnitude more energy than a floating point operation [26], and while technology scaling helps reduce the total energy, data movement still dominates the total energy consumption [47].

To bridge this gap between the computing and the memory, extensive work has been done to explore possible solutions, which can be classified into two categories: The first approach, referred to as the memory-rich processor, sticks with the computing-centric architecture while bringing more memory on-chip. For example, modern processors integrate up to 128MB embedded DRAM (eDRAM) based caches [37]. This on-chip memory not only reduces energy-consuming off-chip memory accesses, but also provides higher memory bandwidth, improving system performance. The second approach, referred to as the compute-capable memory, switches to the memory-centric processing-in-memory (PIM) architecture. Lightweight processing units are designed in the logic die of 3D stacking memories [67] or in the same DRAM die in 2D cases [46, 71] for near/in-memory computing. This approach significantly reduces the traffic between the host and memories, and embraces the large internal memory bandwidth.

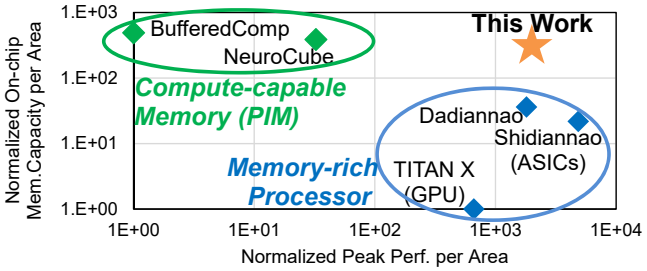

Figure 1: The on-chip memory capacity and computing capability of various approaches [3, 22, 46, 53].

However, both approaches have limitations. As shown in Figure 1, bringing large on-chip memory to the powerful memory-rich processor architectures (the lower right corner) boosts the performance, but the memory capacity is still not enough for data intensive applications. On the other hand, the PIM approaches (the upper left corner) effectively bond more memory to the computing resources, but the performance is not as competitive as GPU/ASICs. For example, Neurocube achieves 132GOPs [53], while the latest GPU can reach 44TOPs [3]. Emerging applications, such as deep learning

and bioinformatics (like meta-genome data analysis [21]), are both compute and memory intensive, with a challenging demand for *both* powerful computing and large memory capacity/bandwidth (the upper right corner in Figure 1), which may not be satisfied by either of these approaches.

Designing a novel architecture to achieve the goal in the target region in Figure 1 is challenging. It is difficult to keep adding more memories to processors, since even the high-density eDRAM suffers from a much larger cell size ($60F^2 - 80F^2$ [31, 38]) than DRAMs ($6F^2$). On the other hand, it is also difficult to improve PIM's performance. For the 3D-based PIM, the area of the processing unit is limited by the logic die's area budget [17]. For the 2D-based PIM, building complex logics with DRAM process technologies results in large area and cost, making the approach unviable for the DRAM industry [18].

The ***goal*** of this paper is to build a processing unit that provides both high computing performance and large memory capacity/bandwidth (the upper right region in Figure 1). To that end, we present a DRAM-based Reconfigurable In-Situ Accelerator architecture, *DRISA*. The accelerator is built using DRAM technology with the majority of the area consisting of DRAM memory arrays, and computes with logic on every memory bitline (BL). By applying the DRAM technology, we achieve the goal of *large memory capacity* for the accelerator. Furthermore, *DRISA*'s *in-situ computing* architecture eliminates unnecessary off-chip accesses and provides ultra-high internal bandwidth. To *avoid the large overhead* caused by building logic with DRAM process, *DRISA* uses simple and serially-computing BL logic. The BL logic has bitwise Boolean logic operations (like `NOR`), which are either performed by the memory cell itself, or by a few add-on gates. *DRISA* can be reconfigured to compute various functions (like additions) by serially running the functionally complete Boolean logical operations with the help of hierarchical internal data movement circuits. Finally, to achieve *high performance* with these simple and serially-computing logic elements, multiple rows, subarrays, and banks are activated simultaneously to provide massive parallelism. We compare four different design options, and present a case study of accelerating the state-of-the-art convolutional neural networks (CNNs). The contributions of this paper are summarized as follows:

- We propose an accelerator architecture, *DRISA*, built with DRAM technology. It provides large on-chip memory and in-situ computing benefits. To reduce the overhead of building logic with DRAM process, we use simple Boolean logic operations for computing but achieve high performance after optimizations.
- A set of circuits and microarchitectures are implemented in *DRISA*, including the BL logic design, the reconfigurable scheme, hierarchical internal data movement circuits, and controllers. Optimizations for unblocking the internal data movement bottlenecks and reducing activation latency and energy are presented to achieve higher performance.
- We use CNN acceleration as a case study to demonstrate the effectiveness of our approach, with resource allocation optimizations. We compare four different *DRISA* designs and present conclusions that guide efficient *DRISA* design. We also compare *DRISA* with the state-of-the-art ASIC and GPU solutions for the CNN case study.

2 BACKGROUND

A DRAM chip contains multiple banks, which are connected with a global bus. Each bank has many subarrays that share global BLs. Global decoders decode parts of the addresses to the global wordlines (WLs) that are connected to different subarrays, which consist of cell matrices (mats) as the basic units. Every mat has its private, local WL decoders, drivers, and sense amplifiers (SAs). A DRAM cell is constructed with an access transistor and a capacitor (1T1C). Within one chip, only a single row in a subarray is activated at a time. The mats in a subarray work in a lock-step manner.

The DRAM fabrication process and the logic fabrication process are very different and often incompatible [55]. It is difficult to build logic devices with a DRAM process, or DRAM with a logic process. The transistors in a DRAM process are highly optimized for density and low leakage, at the expense of performance. Moreover, the DRAM process usually has only three metal layers, but the logic process often has more than twelve metal layers, meaning that logic circuits in a DRAM process could suffer from higher interconnect overhead. In summary, building complex logic circuits in a DRAM process is challenging with 22% performance degradation and 80% area overhead [55], which is the key reason why earlier PIM research that put the processor and DRAM on the same die had limited success. On the other hand, building DRAM cells within a logic process results in embedded DRAM (eDRAM), which is also inefficient. eDRAM results in 10x area overhead [31, 64], around 4× more power, and 100× shorter retention time [64], compared with DRAM.

Deep neural networks are a family of machine learning algorithms inspired by human brain structures [57]. The case study in this paper focuses on the CNN inference task. There are typically three types of layers in CNNs: convolutional layers, pooling layers, and fully connected layers. The convolutional layer is described as follows,

$$f_i^{\text{out}} = \sigma\left(\sum_{j=1}^{n_{\text{in}}} f_j^{\text{in}} \otimes g_{i,j} + b_i\right),\ 1 \le i \le n_{\text{out}}, \tag{1}$$

where f_j^{in} is the j-th input feature map, and f_i^{out} is the i-th output feature map, $g_{i,j}$ is the convolution kernel, b_i is the bias term, and n_{in} and n_{out} are the numbers of the input and output feature maps, respectively. σ is the activation function that could be ReLU, clip, hard tanh, etc. The pooling layer sub-samples the feature maps. The fully connected layer calculates the dot product between the input neuron vector and the synapse weight matrix. The output vectors then go through the activation function. Besides these layers, batch normalization [45] is applied, which normalizes the data with mean and variance values collected during training.

3 OVERVIEW

The Key Idea. To implement in-situ computing with large on-chip memory, we build *DRISA* with DRAM process technology. The main challenge is to efficiently build complex logic functions within the DRAM process. We solve this problem by only building simple Boolean logic operations. Figure 3 shows a logical overview of *DRISA*. To avoid building complex circuitry in DRAM process technology, we leverage vast, parallel DRAM internal resources to increase computational ability by serially cascading on simple

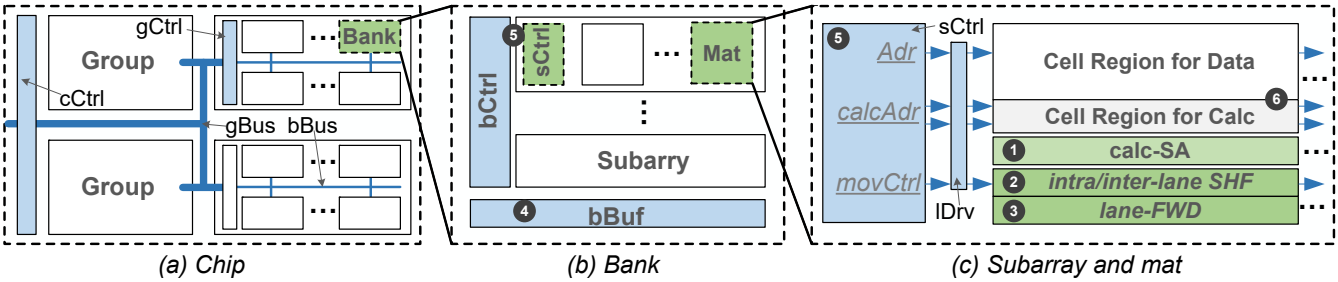

Figure 2: *DRISA* architecture design. (Glossary - cCtrl/gCtrl/bCtrl/sCtrl: chip/group/bank/subarray controller. gBus/bBus: group/bank bus. bBuf: bank buffer. lDrv: local driver. SHF: shifter. FWD: forwarding.)

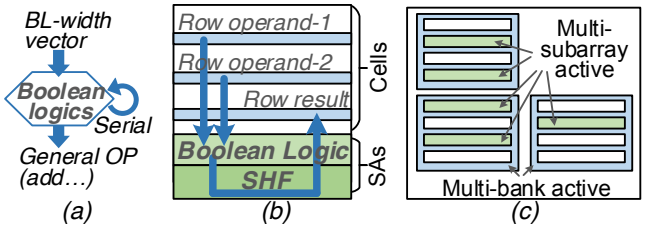

Figure 3: A logical overview of *DRISA*. (a) Performing general operations through serially running Boolean logic operations. (b) Implementing Boolean logic operations with SA's help for each BL. (c) Multi-bank/subarray activation for more parallelism.

Boolean logic. The bitwise Boolean logic operations are implemented in an efficient manner for each BL. To compute, *DRISA* first opens two rows, performs logical operations using SAs modified with logic and shifters, and then writes back to a result row. It achieves reconfigurability by implementing different sets of functions serially for a desired overall function. However, in order to increase throughput with multiple DRAM resources, multiple rows, subarrays, and banks need to be activated simultaneously, leading to challenges that we describe next.

Challenges and Solutions. Our *DRISA* architecture exploits massive DRAM parallelism and achieves large computational throughput for in-situ reconfigurable computing. However, new design methodologies are required to achieve high performance. We outline the challenges and our contributions to address them:

• ***Challenge-1:****Achieving high performance with the simple and serial logic elements.* We target *DRISA* as an accelerator instead of as host memory to avoid tight area constraints, and hence we can optimize it for high performance. *DRISA* requires simultaneous activation of multiple subarrays and banks to provide large parallelism and thereby large computational throughput. To solve this, we propose bank reorganization to enable activating multiple rows (Section 4.4).

• ***Challenge-2:****Unblocking the internal data movement bottleneck.* We propose group/bank buffers to isolate local movements in DRAM and enable moving multiple data buffers in parallel (Section 4.2). We also reorganize the bank to reduce data collisions on the shared data bus by designing a hierarchical bus (Section 4.4).

• ***Challenge-3:****Optimizing ACT to reduce its latency and energy.* Activation (ACT) is a basic step for *DRISA* computing. Directly adopting a DRAM ACT mechanism results in large latency and energy overheads. Our bank reorganization makes WLs/BLs shorter (Section 4.4). We also present split computing and storage array regions, μ-operations, and local instruction decoding to save latency and energy on ACT (Section 4.3).

4 DRISA ARCHITECTURE

We show *DRISA*'s architecture design in Figure 2. It inherits most aspects of standard DRAM design. However, one more hierarchy, called a group, is added between the hierarchy of the chip and bank. Groups are connected with a bus (*gBus*), and controlled by the chip-level controllers (*cCtrl*). Within a bank, bank-level decoders are modified as controllers (*bCtrl*). Bank buffers (*bBuf*) are added to help data movements. In a subarray, a subarray-level controller (*sCtrl*) is added. In a mat, the cell array is split into two regions, for storage and computing, respectively. The SA is modified to support the computing. Extra hardware to support data movements is also added. A mat is logically partitioned vertically into lanes, and each lane is equivalent to an n-bit processing unit. We elaborate on the design details and justify the design choices in the rest of this section.

4.1 Microarchitecture for Computing

The basic operating units for the reconfigurable computing are the Boolean logic operations and the shifters (❶ and ❷ in Figure 2, respectively). For the logic part, there are two approaches that can make DRAMs computing-capable: the *3T1C* solution and the *1T1C* solution. Both of these approaches share the same shifter design.

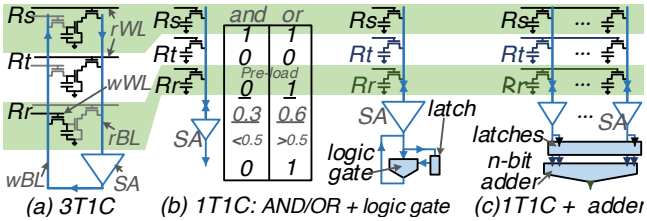

Figure 4: Cell structures. (Glossary - Rs/Rt/Rr: Operand source row s and source row t, result row r.)

• ***3T1C-based computing:*** This design changes standard DRAM cells to *3T1C* and uses cells themselves for computing. Therefore no other circuits are required. *3T1C* cell was used in early DRAM designs [84]. As shown in Figure 4(a), both WLs and BLs are separated as two lines for read and write, respectively. The cell includes two separated read/write access transistors, and an extra transistor

that decouples the capacitor from the read BL. The third transistor also connects the cell in a NOR style on the read BL. Therefore, the *3T1C* cell naturally performs NOR logic (NOR itself is functionally complete) on BL without any extra design changes.

• ***1T1C-based computing:*** This design keeps the standard DRAM cells unchanged, but uses extra circuits attached to the SAs for computing. There are two types of computing. First, it calculates AND and OR using the method proposed by Seshadri *et al.* [80] (on the left part of Figure 4(b)). The result row is pre-stored with 0 (for AND) or 1 (for OR), and three rows are activated simultaneously. Then, after charge sharing (shown as 0.3 and 0.6 in the figure), the SA readout is the logic result, and the result is restored to the result row during the row closing. Note that this operation will also destroy the operand rows, so a row copy before the operations is required. However, the problem is that AND/OR alone are not logically complete. Therefore, the second types of computing is demanded. *DRISA* calculates other logic function like NOT to achieve logical completeness, as shown in the right part of Figure 4(b). Extra circuits for a latch and logic gates (to perform one or some of Boolean logic operations) are added. The operand *Row s* (R_s) is activated first, and the data is stored in the latch. Then, the operand *Row t* (R_t) is activated, and its data, along with the data in the latch, is fed into the logic gate. The result is then read out or restored to the result row. Taking the *1T1C*-based solution to an extreme scenario, we can also design a *n*-bit adder circuit for *n*-bit BLs, as shown in Figure 4(c).

Both of the *3T1C*- and *1T1C*-based solutions make each BL computing-capable. This architecture makes in-situ computing possible, since the memory cell and the BL logic are tightly coupled.

• ***Circuits for intra-lane SHF:*** Shifters are required in our architecture because the bitwise Boolean logic operations only performs logical operations but not arbitrary data movement. Shifters are designed for data shuffling, thereby enabling general-purpose computing. Figure 5(a) shows the shifter circuits that take a 4-bit lane as an example. The circuits are located at ❷ in Figure 2. Figure 5(b) and (c) show the examples for left shift-2 and right shift-3. The circuit design is similar to a barrel shifter. For an *n*-bit lane, we implement arithmetic 1/(*n*-1)-bit right shifts. We also implement 1/*exp*2-bit left shifts. The left shift is either a logical or arithmetic shift. We design filling lines that can fill in 0/1 accordingly. This design can then perform arbitrary shifts by running serially.

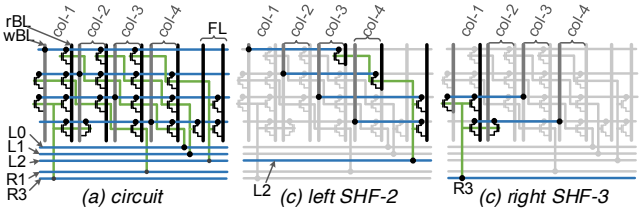

Figure 5: The shift (SHF) circuits and examples. (Wires - Green: Poly. Black/Gray: Metal-1. Blue: Metal-2. Glossary - Lx/Rx: Left/Right shift x. rBL/wBL: read/write bitline. FL: Filling line.)

Our shifter design is optimized for common cases. Even though 1-bit left/right shift alone is functionally complete, we design extra (*n*-1) bit right shift circuits to cover the common case that makes the whole lane all-zero/one according to its sign bit. We also design special *exp*2-bit left shift circuits to cover the most common shifts in full adders, making it 11% faster. We do not design special circuits for every possible shift cases, thus saving 60% shifter circuit area, 84% shifter latency, and 52% shifter energy, without any performance degradation for all μ-operations defined in Table 1.

• ***Reconfigurable computing:*** With the functionally complete logic and shifters, *DRISA* can theoretically accomplish any operations by running serially. In Figure 6, the *3T1C* one-cycle NOR logic is used as an example to show how *DRISA* supports some frequently used operations, i.e., selection (*SEL*), addition (*ADD*), and multiplication (*MUL*). We use the notation shown in the upper-left corner of Figure 6, where the blocks denote rows in DRAMs, white rows are inputs, blue rows are intermediate terms, and green rows are outputs. Arrows connect the sources and result of a NOR logic.

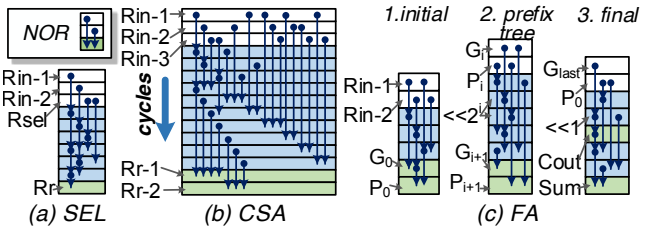

Figure 6: Building operations with basic Boolean logic operations.

To calculate *SEL*, we duplicate the selector 0/1 to a whole row. Then the *SEL* is broken down into NOR logic step by step, as shown in Figure 6(a), which takes 7 cycles and 6 intermediate terms in total. For adders, we show both a carry-save adder (*CSA*), which has three inputs and two outputs without a carry-out, and a full adder (*FA*), which has two inputs and one output and a carry-out. Breaking down the *CSA* into NOR logics, we get Figure 6(b). The *FA* calculation is shown in Figure 6(c). There are three steps: Step one initializes and generates partial terms P_0 and G_0. Step two follows the prefix tree [68] logics, generates P_i and G_i, and iterates $log(n)$ times for an *n*-bit adder. During this step, left shifting is required. Step three finally generates the sum and the carry-out. For the *MUL*, the calculation depends on the multiplier's bit width. If the multiplier is binary, the *MUL* collapses as a XNOR logic. Otherwise, *MUL* is calculated by generating partial terms with shifter and *SEL* and then sums all these partial terms with both *CSA* and *FA*.

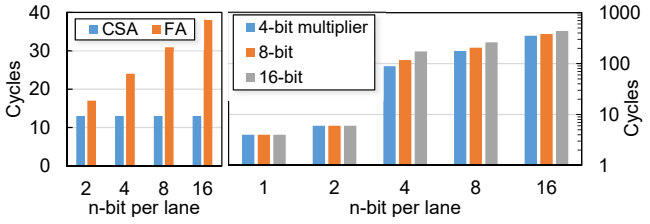

Figure 7: *3T1C* computing cycles for *ADD* and *MUL*.

Figure 7 (left) shows the cycles to compute *CSA* and *FA*. *DRISA* favors CSA because as each lane's bit width increases (x-axis), *CSA* retains a constant latency. Figure 7 (right) shows *MUL* with different lane counts and multiplier widths, both in bits. For 1-bit and 2-bit *MUL*, it takes only 5-6 cycles. However, if the multiplier has more than 3 bits, it takes hundreds of cycles for computing.

4.2 Microarchitecture for Data Movement

A general purpose processor requires flexible data movement. The shift circuits mentioned above only cover inter-lane movement. We design circuits for data movement between lanes in this subsection, in order to have a functionally complete hierarchical shift solution. Specifically, within the subarrays, we design *inter-lane SHF* circuits. Between subarrays, we have improved *RowClone* [81]. We also have the *lane-FWD* circuits, to move data from and to arbitrary lanes.

• ***Circuits for inter-lane SHF:*** *Inter-lane SHF* (❷ in Figure 2) shifts a row from one lane to its adjacent lane. The circuits are shown in Figure 8(a). The shifter contains ping/pong phases. We choose this two-step shift because it saves area, since all lanes share the same shift data wire (the two upper blue wires in Figure 2). Figure 8(b) and (c) show the example of left shift. If shifting right, *ping* and *odd* are first selected for the ping-phase, and then *pong* and *even* for the pong-phase.

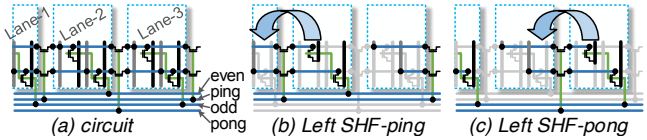

Figure 8: The *inter-lane SHF* circuits and examples.

• ***Circuits for lane-FWD:*** *Lane-FWD* (❸ in Figure 2) supports random read/write from/to an arbitrary lane. Figure 9(a) shows the circuit design. Figure 9(b) and (c) show examples of reading Lane-1 and writing Lane-2, respectively.

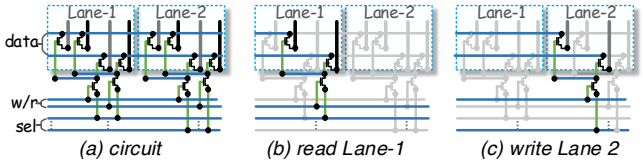

Figure 9: The lane-FWD circuits and examples

• ***Enhanced RowClone with bank buffer:*** We improve upon the existing *RowClone* [81] technique in *DRISA*. Besides original row-to-row copy, *DRISA* can choose either to copy the whole row, or to repeatedly copy the data from a certain lane. Furthermore, we add bank buffers (❹ in Figure 2) in order to tackle Challenge-2 (preventing data movement from becoming the bottleneck). The limitation of *RowClone* lies in the shared memory data bus. Although multiple subarrays/banks work simultaneously in *DRISA*, the initial *RowClone* only works with two subarrays at one time since it utilizes the shared data bus between banks. Bank buffers, which are implemented by registers, isolate intra-bank *RowClone*s from other banks, so that multiple intra-bank *RowClone*s work in parallel in different banks.

4.3 Microarchitecture for Controllers

• ***Instruction design:*** We abstract an instruction set shown in Table 1. The instruction follows the R-format in MIPS [72], which contains the opcode, the address for two (or possibly one or three) input rows and the output row, and the *funct* code that describes detailed controls. *DRISA* has the basic instructions for Boolean logic operations and data movement mentioned earlier, as shown in the left side of Table 1. In addition, there are also μ-operations, including frequently used functions (*SEL, ADD, MUL, MAX,* etc). Next, bulk data copy and also compute reductions with addition/maximal/minimal operators are supported. Finally, a vector-wise inner-product operation is also supported. Note that instructions like control transfer are carried out by the host and therefore not included in Table 1.

Basic Instr.	opcode	funct	μOps.	opcode	funct
	Logic (NOR etc)	type of logic		*Calc. (FA etc)*	N/A
	SHF	L/R, offset, filling		*Bulk-copy*	length, stripe
	Lane-SHF	L/R, offset		*R-SUM*	length
	Dup-copy	N/A		*R-MAX/MIN*	length
	Copy	bank/group/chip		*Inner-product*	length

Table 1: The basic instructions and μ-operations.

• ***Multi-level controllers:*** *DRISA* has four levels of controllers (❺ in Figure 2): chip, group, bank, and subarray-level controllers. They support simultaneous multi-subarray/bank activation for better parallelism. The first two levels (chip/group) of controllers are essentially decoders, but they can also help with data movement. The bank-level controllers decode the instructions. They convert the instructions and μ-operations into addresses, vector lengths, and control codes, and then send them to the controllers in the active subarray. The subarray controller consists of address latches, local decoders, and counters. The address latches are essential for multi-subarray activation [54]. The counters are used for continuously updating addresses to local decoders for the bulk-style μ-operations.

• ***Split array regions:*** The cell array is split into the data region and the compute region (❻ in Figure 2). They share BLs and SAs, but have separate decoders in the subarray controllers. This separation reduces the area and performance overhead while supporting multi-row activations (required by computing in Section 4.1). A strong decoder that activates multiple rows in one cycle is costly. On the other hand, designing a latch for each local WL and serially decoding for the active rows [60] wastes too much latency and energy. Instead, in the split array case, the data region that has most of the cells does not need multi-row activation. The compute region that stores the intermediate data (blue rows in Figure 6) only contains a few (typically 16) rows. Designing a strong one-cycle decoder is much easier.

Without the split regions, a strong one-cycle decoder takes 204.3% area overhead (compared with a normal 256 fan-out decoder). On the other hand, the serial solution only takes 4.3% area overhead, but results in 10.8% peak performance degradation. After adapting the split cell region idea, we have one-cycle decoding with only 19.02% area overhead.

4.4 Optimizing Bank Reorganization

We reorganize bank/array in *DRISA* to optimize for performance and energy efficiency. Conventionally in DRAM memories, bank/array organizations are optimized for memory density. We switch the optimization objective in *DRISA* since we are now designing accelerators instead of memories.

• ***Improving the parallelism:*** We have to improve parallelism to achieve high performance. It takes *DRISA* around 30 cycles for *FA*. If considering each cycle as `tRC`, *DRISA*'s adder runs as slow as ~1MHz. To overcome Challenge-1, we present two techniques:

(1) *DRISA* simultaneously activates multiple subarrays in multiple banks. To achieve this, each subarray and bank has their independent controllers with latches. Previous work [54] shows such modification incurs ignorable area overhead. The detailed controller design is shown in Section 4.3. (2) Furthermore, *DRISA* activates more rows at one time. We reorganize the banks and subarray by making the subarray smaller (few number of rows) but with larger quantity. Therefore, the number of rows/subarrays that can be active simultaneously is increased. Later, Figure 11 shows that it costs 58% more area but gains 4× more parallelism. This is worthwhile when optimizing for performance per area.

However, there are limitations for out parallelism improvements. First, the power budget is a hard constraint. Second, modern DRAMs use open-BL architecture, where the SA works with a differential sensing mechanism. It needs an idle adjacent subarray as a reference. Therefore, adjacent subarrays cannot be activated simultaneously, i.e., we can at most activate 50% of all the subarrays. Third, more compute parallelism is not necessarily better, since internal data movement may become the bottleneck. Section 6.2 shows that the subarray's effective utilization can be as low as 10%, due to data blocking.

• ***Unblocking the data movement bottleneck:*** For Challenge-2, we propose four techniques. (1) We design bank/group buffers in Section 4.2, in order to isolate local movements and parallelizing them. (2) We reorganize the banks by reducing the number of subarrays per bank while increasing the bank quantity. Fewer subarrays per bank reduces the data conjunctions for intra-bank data movements. (3) We add groups and group buffers, so that inter-bank data movements inside different groups work in parallel. (4) We propose the bulk-style μ-operations (Section 4.3) to reduce instruction data movement (and decoding). One μ-operation only requires one-time data transfer and decoding, and then the local controller auto-generates bulk instructions.

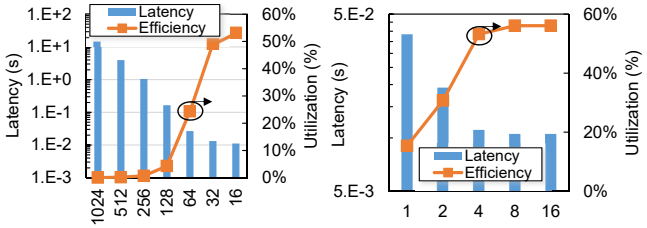

Figure 10: The latency and resource utilization for real application (VGG-16 on *1T1C-mixed*). Left: Impact of number of subarrays per bank. Right: Impact of number of groups.

Figure 10 (left) shows that reducing the number of subarrays per bank from 1024 to 64 achieves 576× better performance with only 1.5% area overhead. This increase of resource utilization is the evidence that this performance gain comes from faster data movements. Figure 10 (right) shows that as the number of groups increases from 1 to 16, it achieves 3.65× better performance. In addition, by adapting μ-operations, we achieve another 22.94% and 3.43% reduction on latency and energy, respectively.

• ***Optimizing ACT latency and energy:*** Reducing the ACT overhead is essential in *DRISA*. In a typical DRAM, a activation cycle (tRC) takes 46ns [2] and 24.9% of the memory power consumption [96]. Such a long clock period and large energy consumption are challenging for *DRISA*, which computes serially with multiple cycles per operation. To tackle this Challenge-3, we propose three techniques. (1) We reorganize arrays with shorter BLs (fewer columns) and WLs (fewer rows) [86]. Shorter BLs and WLs result in smaller RC and hence smaller latency and energy, as well as easier and faster decoding. (2) We include the extra group hierarchy between the chip and banks. This makes the bus become hierarchical and hence more scalable, so that the bus overhead is reduced. (3) The μ-operations also help; one μ-operation may contain hundreds of ACT instructions, but it is only transferred on the global bus for one instance. Average ACT cost is then reduced.

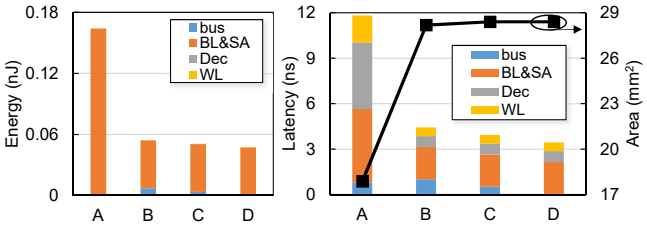

Figure 11: Reducing the activation energy and latency (A: 8 banks, 1K-16K subarray; B: 512 banks, 256-2K subarray; C: B with group (128 banks per group); D: C with local decoding.)

Figure 11 shows the ACT energy and latency for four cases (A to D) with area results. *Case-A* is the bank organization in the original DRAM memory. We observe that the BL dominates both the latency (41%) and energy (99%) in ACTs. This is the motivation for our first technique that switches to shorter WL/BLs, as shown by *Case-B*. 4× shorter BLs and 8× shorter WLs yields 62.5% latency and 66.9% energy/bit reduction. The downside is 58% larger area. However, this is acceptable since *DRISA* is an accelerator optimized for performance, not a memory optimized for density. *Case-C* shows the second technique's benefit. By having the group hierarchy, it reduces the latency and energy spent on the bus by 49.9% and 50%, respectively. *Case-D* shows that the third technique is effective. By adapting μ-operations, it further saves 12.7% latency and 6.9% energy.

4.5 System Integration

We briefly discuss how to integrate *DRISA* into the system (detailed software/system support is out of the scope of this paper and planned as future work). Considering that *DRISA* is a co-processor or accelerator instead of a memory, it is integrated in the same manner as a GPU or FPGA, not a PIM system.

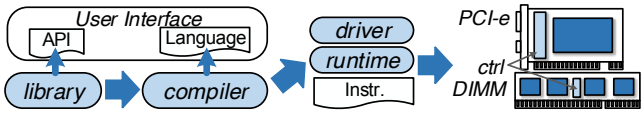

Figure 12: Integrating *DRISA* into the System.

For the software component , *DRISA* follows the same programming model as Automata [27]/GPU/FPGA. *DRISA* requires a special programming language or framework, like CUDA for GPU and AP SDK [6] for Automata. A corresponding compiler is also necessary. In order to map general purpose program onto *DRISA*, programmers can treat *DRISA* as a multiple issued vector machine, similar to programming with AXE/SSE. To make programming easier, application-specific APIs should also be provided to the users. The

DRISA compiler compiles the high-level descriptions into *DRISA* instructions, and it works along with the driver and the runtime engine to offload tasks onto *DRISA*, transfer data, and control *DRISA* to finish the task.

For the hardware, both PCIe and DIMM solutions are applicable. PCIe integration (like GPU, FPGA, Automata [27]) provides sufficient power delivery and well-developed control system. The DIMM solution (like AC-DIMM [35]) requires *DRISA* to support a DDR-like interface but function like an accelerator. This solution is still within active research. The advantages of the DIMM solution is simplicity for scaling-out, considering that the number of DIMM slots is much more than PCIe's. The downside is that the power budget for each slot is low, which limits the performance of every individual *DRISA*. For both of those solutions, there is an SoC controller on board, which supports inter-chip communications.

To scale-out for applications with larger data sets than *DRISA*'s memory capacity, *DRISA* follows the solution of multi-GPUs, which leaves the partitioning job to programmers or frameworks (like Torch [9]).

4.6 Discussion

Limitations. *DRISA* is not suitable for floating point calculation, even though it is capable of this functionality. The limitation lies in the lock-step within a subarray, since all lanes in one subarray share the same controller. This dramatically hurts the performance of floating point operations, because the internal control of floating point calculation is data dependent. For example, the shift for significands alignment is based on the subtraction result of exponential biases. Therefore, every lane potentially requires a different bit shift, which is not supported by the lock-step architecture. Instead, a single floating point operation is required to run on a whole subarray instead of a single lane, massively reducing lane-level parallelism.

Process variation. Multiple experimental studies and patents have already established the viability of 3T1C and 3 row activation designs in DRAM, even in the presence of manufacturing variations. In addition, Micron's Automata has demonstrated the feasibility of heterogeneous circuits design with DRAM process technology. Specifically for DRISA, we examine the impact of these variations and general DRAM challenges, since the computed results depends on the nominal operation of the proposed bitwise logic computational the DRAM cell level.

The first challenge is with variable cell voltage level due to charge-leak, thereby impacting the charge-sharing operation with bitline. *DRISA* is not impacted by this challenge since every bitwise logical operation is preceded by a data-copy to the source and destination rows (Rs, Rt). This naturally constitutes DRAM restore operation, charging the voltage levels to the cell value and offsets any charge leaking that could affect cell-sharing operation with three-rows. In addition, *DRISA* can tolerate 8ms retention time, compared with 64ms in commodity DRAM, which makes it even more robust.

The second challenge relates to variation in the cell-level capacitances that could affect the bitwise logical operation due to strong or weak capacitances. Fortunately, *DRISA* has a 4-point approach that ensures strong immunity to these challenges:

• First, process variations' impact in the context of multi-row activation in 1T1C-based designs were already examined in detailed by Buddy-RAM [82]. The impact was shown to be minimal, affecting special patterns of cell values with specific cell capacitance strengths. However, even for these cases, measurements show that the logic function sustains even with ±20% process (40% cell-cell) variation. For the case of *DRISA* design, the theoretical limits allow ±33% capacitance variation for a 3 wordline design, but our evaluation places a tighter bound of ±28% to ensure safe margin for sense amplifier. DRAM systems today have a process variation much less than this tolerable ±28% (56% cell to cell). For example, the capacitance difference between two generation is only 10~15% [70]. Also, industry inventions on new DRAM capacitance structures significantly increase the capacitance of DRAM cell and therefore reduce the impact of variation [70].
• Next, as discussed in Section 4.1, *DRISA* array structure is fundamentally different and is tailored to be an accelerator with 16× smaller array size than commodity DRAM (256-by-2048 v.s. 1024-by-8096). This results in a proportionately smaller number of cells sharing the local bit lines which is therefore significantly shorter. This in turn improves the ratio between local bitline and the cell capacitances and therefore the sensing ability beyond commodity DRAM and other existing approaches.
• The impact of process variation can also be handled at the circuit level. Unlike cost-optimized commodity DRAMs, *DRISA* can tune the SA design and spend more area for extra reliability [51]. Also, in 1T1C, we can also use the logic gates (Figure 4(b) right side) for pure digital computing, where no multi-row ACT or SA is involved, immune to process variation.
• Finally, we can apply architecture-level method to avoid defective modules. For example, *DRISA* can examine the capacitance variation during the manufacturing and testing phase. Cells that are detected to contain more than the acceptable variation will be masked and instead replaced with spare row and column by using already prevalent fusing techniques, consistent with existing DRAM. Since the spare DRAM cells are already implemented in the state-of-the-art DRAM chips, there is no extra hardware overhead. With a high threshold, we expect this to produce similar yield as a normal process. Another way is to apply defect-aware mapping method similar to ArchSheild [66].

The final challenge relates to DRAM yield as a result of coexistence of logical elements and the DRAM cells. By virtue of design, these logical components i.e. shifter, are integrated after the sense amplification stage and do not interfere with the highly-optimized DRAM cell level, IO lines' layout. As such, it does not affect DRAM yield either.

***DRISA* is an accelerator, not host memory.** We position *DRISA* as an accelerator or co-processor instead of as part of the host memory, because memory designs are extremely optimized for low-cost, but our target is to build a high performance accelerator. Conventionally, DRAM is cost sensitive and is unlikely to be changed. However, by avoiding being a part of the host memory, *DRISA*'s area is not the primary optimization priority, and we can trade-off area overhead for better performance. As we change the design goal to high performance, *DRISA* has greater room to re-design the DRAM

arrays and peripheral circuits for high performance parallel computation. In addition, we treat *DRISA*'s memory space similar to the device memory or scratchpad memory on GPU/FPGA/Automata, which avoids issues in data coherence, data reorganization, and address translation if using *DRISA* as host memory.

5 ACCELERATING CNN: A CASE STUDY

In this section, we map CNNs (inference) on *DRISA* as a case study. Note that the purpose of the case study is to show the methodology of application mapping. *DRISA* is not limited to only CNN applications.

5.1 Quantizing CNN for *DRISA*

NN algorithms are originally based on floating point calculations. NN data quantization work [40, 44, 59, 89, 97] helps to tackle the challenge by quantizing the floating point activation data and weight into fewer bits of fixed-point data and then retraining the NN to reduce the accuracy loss. Furthermore, research studies have found it is even possible to quantize weights into binary data [24, 25, 77]. After proper training, BWN [77] (binary weight, floating point activation data) shows 0%, 8.5%, and 5.8% top-1 accuracy degradations, compared with all floating point golden models on AlexNet [56], ResNet-18 [41], and GoogleNet [87], respectively. Even more aggressively, BNN [24] and XNOR-Net [77] also binarize the activation data. It shows Top-1 accuracy degradation of 12.4% and 18.1% on AlexNet and ResNet-18, respectively.

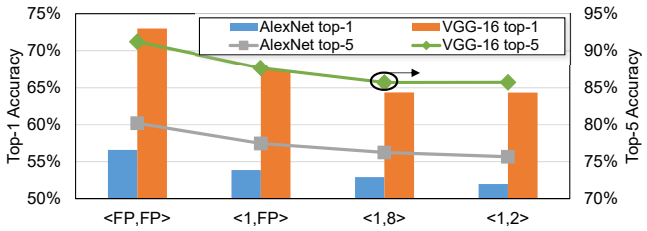

Figure 13: Training *DRISA*-friendly CNN on ImageNet (<x,n> denotes x-bit weights and n-bit fixed point activation data).

We apply 1-bit weights with 8-bit activations (<1,8>) for *DRISA*. This is because *DRISA* runs faster if one of the multipliers have 1 or 2 bits, while it is insensitive to the other operand (Figure 6). Therefore, it is not necessary to adopt the extreme BNN/XNOR-Net quantization cases (both binary weight and activation data). Instead, we need an eclectic way with binary weights and fixed-point activation data (between BWN and BNN/XOR-NeT). Specifically, we use binary weights for all layers (including the first and last layers), and use shifter for approximate weight scaling. Figure 13 shows our training result for AlexNet and VGG-16 [85] on ImageNet [78]. Note that Figure 13 shows a "worst" case scenario, since we have not applied fine tuning for our training. More training epochs, larger batch sizes, image augmentation [92], better initialization [97], and better learning rate tuning will effectively increase the accuracy. It has much larger potential since BWN (<1,FP>) [77] has been reported as 56.8% top-1 accuracy, which sets a upper bound of our accuracy.

5.2 Resource Allocation

DRISA provides row-to-row operations, which are treated as SIMD vector instructions (like AVX [5]). In CNN applications, more than 99.9% of the operations can be aggregated as vector operations [61]. For these scalar operations or vector operations that are shorter than *DRISA*'s row size, we fill zeros in any unused slots.

In order to efficiently utilize *DRISA* hardware, we need to optimize resource allocations for CNNs. CNNs have lots of inherent parallelism (e.g., batch, feature map, and pixel-level parallelism), and so does the *DRISA* architecture (i.e., group, bank, subarray and lane-level parallelism). How to effectively allocate the hardware resource to the application is challenging. We follow two design philosophies: (1) The data movement and computation tasks should be balanced, in order to achieve the highest efficiency. (2) Since *DRISA* favors *CSA* compared to *FA*, we should avoid using *FA* (Figure 7).

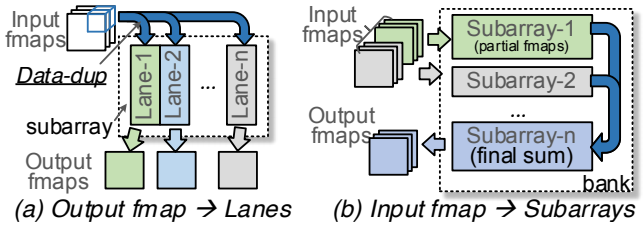

Figure 14: The basic resource allocation scheme for lanes and subarrays.

We design the resource allocation scheme as shown in Figure 14, following these design philosophies. First, all the input data from feature maps are duplicated to all the lanes in a subarray (Figure 14(a)). These lanes work in parallel after weights are preloaded. The output of each lane is a dependent output feature. The purpose for this mapping scheme is to make all of the sum reductions happen in a single lane, so that inefficient *inter-lane SHF* and *lane-FWD* are eliminated. Second, each subarray takes part of the input feature maps. These subarrays work in parallel on the partial results, and an extra subarray sums up the final result (Figure 14(b)). Third, for bank and group level parallelism, we take use of the application's parallelism in pixel and batch, i.e., mapping different regions of the feature map to different banks, and different batches to different groups.

5.3 Mapping Other Applications to *DRISA*

DPU is not limited to only CNN applications. *DRISA* is not limited to CNN inference accelerations. Data quantization in recursive NN (RNN) with 2-bit weights [69] is also an ideal case to run on *DRISA*. Instead of inference, training is also feasible by quantizing gradient data with DoReFa-Net [97].

Furthermore, *DRISA* is not limited to deep learning applications. *DRISA* is designed as a SIMD architecture and can be treated as a vector processor, so a large range of applications can be mapped to *DRISA*. Programs benefit from *DRISA* the most if they have enough data parallelism, if they are both compute and memory intensive, and if they can be mostly computed by integer operations. We are currently working on mapping emerging bioinformatic applications (meta-genome data analysis [21]) to *DRISA*.

6 EXPERIMENTS

In this section, we first describe the experiment setup. Then, overall performance, energy, and area evaluations are presented. The

evaluation for the CNN acceleration case study is also presented with comparisons to the state-of-the-art solutions.

6.1 Experiment Setups for *DRISA*

We evaluate and compare four *DRISA* designs. The configurations are shown as follows.

3T1C: 8-bit lane, 256 rows and 512 columns per mat, 4 mats (256 rows by 2048 columns, or 256 lanes) per subarray, 16 subarrays per bank, 64 banks per group; in total 4 groups and 2Gb capacity. 22nm DRAM process technology and the 3T1C cell size is $30F^2$ [49].

1T1C-nor/mixed/adder: 1T1C-based solutions with NOR logic, or mixed logic gates (including NAND, NOR, XNOR, INV), or adder circuit attached to SA (see Section 4.1). In total, 8 groups and 4Gb capacity. The cell size is $6F^2$. Rest of the configurations are same as *3T1C*'s.

In order to evaluate the brand new hardware, two in-house simulators are developed. First, a circuit-level simulator is built based on CACTI-3DD [50]. CACTI-3DD is DRAM circuit simulator. It provides DRAM latency, energy, and area parameters, which are validated with fabrication DRAMs. Based on it, our simulator modifies the configuration files to reflect array organization. Then we add extra circuits described in Section 4 with APIs provided from CACTI-3DD. The controllers and adders in the *1T1C-adder* solution are synthesized by Design Compiler [4] with an industry library. **The difference between the logic process and DRAM process technologies are capture from parameters in previous research [55].** Second, a behavioral-level simulator is developed from scratch, calculating the latency and energy *DRISA* spends given a certain task like system-C simulation. It also includes a mapping optimization framework for the CNN applications, according to the design space exploration described in Section 5.2.

6.2 Evaluation for *DRISA* Solutions

Table 2 shows the area comparison of the four *DRISA* configurations. First, we observe that even though the memory density of *3T1C* is half as much as others, *3T1C* only takes 17.6% more area than *1T1C-nor*, due to the large cell footprint of the latter. Second, *1T1C-adder* takes the largest area, due to the more complex logic circuit (adders) that are embedded. Third, *DRISA* is almost half as dense as a normal 8Gb DRAM memory. However, *DRISA* is not a cost sensitive memory design. Although it is not as dense as a memory, it very area efficient as an accelerator. Later experiment shows *DRISA* offers the highest performance per area among all kinds of accelerators. Note that though a larger chip with higher performance is feasible, the die size impacts the chip cost. *DRISA* thus has a similar die size to commercial DRAM memories in this paper. For a fair comparison, the following results are normalized by area.

Solution	*3T1C*	*1T1C-nor*	*1T1C-mixed*	*1T1C-adder*	*DRAM-8Gb*
Area (mm^2)	64.58	54.90	65.22	90.91	60.44

Table 2: The area comparison of *DRISA* solutions, including an 8Gb DRAM memory as a reference.

Figure 15 shows the area breakdown. First, we observe that the breakdown of *3T1C* is similar to that of a DRAM memory, where cells and analog IO circuits dominate (79%). The add-on shifters, controllers, buffers, and bus circuits constitute a smaller fraction (less than 5%) in area. Second, the add-on NOR and latch circuits in *1T1C-nor* take 24% of the area, almost as much as the memory cell themselves. This is because of the inefficient implementation of logic circuits with DRAM process technologies. Third, the add-on adder circuits in *1T1C-adder* takes 51% of the total area, resulting in 66% more area than *1T1C-nor*. Again, this result supports the observation that **DRAM process technologies are not suitable to build complex logic circuits, with design complexities even for simple adders.**

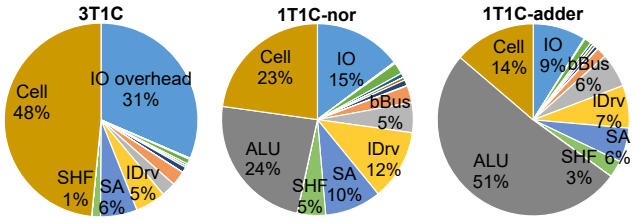

Figure 15: The area breakdown of three *DRISA* solutions.

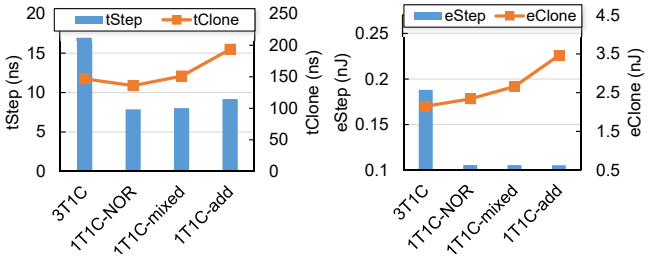

Figure 16: The latency and energy comparison among four *DRISA* solutions.

Figure 16 shows the latency (*tStep*) and energy (*eStep*) for a basic computing step (including opening operand rows, computing, closing operand rows, and writing back to result row). It also shows the latency (*tClone*) and energy (*eClone*) to copy one row across subarrays in the same bank. First, we observe that *3T1C* takes 112% more computing latency and 79% more computing energy, due to the longer BLs/WLs and larger cells. Second, the data movement latency/energy are dominated by the latency/energy on wires. Hence, designs with larger areas result in larger latency/energy. Third, a latency breakdown shows that the logics' latency takes less than 1% in all *DRISA* cases, except for *1T1C-adder* which spends 10% latency on the adder.

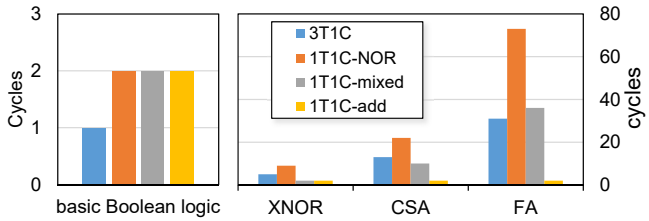

Figure 17: Cycles for frequently used operations.

Figure 17 shows the cycle count for frequently used operations. It shows that *1T1C-nor* is the slowest since for AND/OR the system requires copy-on-operation, which takes 3 more cycles per logic. *1T1C-mixed* appears to be better since every logic operation is based on the add-on logic gates. However, it still needs 2 cycles for each Boolean logic operation. *1T1C-adder* is clearly the fastest design.

In later sections, we will show effective performance of these four solutions for more comprehensive comparisons before we draw the conclusion in Section 7.

6.3 The CNN Case Study Results

We compare *DRISA* with the state-of-the-art solutions in the CNN inference acceleration case study.

• ***Baseline setup:*** We also compare with state-of-art accelerating platforms for the CNN application. They are described as follows.

ASIC: This is a DaDianNao-like [22] ASIC design but optimized for binary weight CNN with 8-bit activation data. There are two versions with either 8x8 tiles (33MB eDRAM) or 16x16 tiles (129MB eDRAM). An advanced on-chip data reuse scheme as in ShiDianNao [28] is adopted. The design is synthesized with Design Compiler [4] and scaled to 22nm. The eDRAM and SRAM are calculated from CACTI [65]. An in-house behaviorial-level simulator is built to evaluate the performance and energy given a certain CNN task.

GPU: We use two TITAN X (Pascal) [3]. Each GPU has 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance). *GPU-FP* is achieved by running Torch 7 [9] with cuDNN [7] using floating point data. We measure the power consumption with NVIDIA's system management interface [8]. The results are conservatively scaled by 50% to exclude the power cost by cooling, voltage regulators, etc. We then aggressively scale the *GPU-FP* result by ×4 for the quantized CNN[1], since it claims ×4 peak performance running with 8-bit integers instead of floating point data.

In the case study, we consider four CNN applications (including both convolution layers and fully connected layers): 8-layer AlexNet [56], 16-layer VGG-16, 19-layer VGG-19 [85], and 152-layer ResNet-152 [41]. Note that as another advantage, *DRISA* does not have refresh overhead. Even in the most complex CNN case, one iteration of the task is done within 8ms, which means every row have already been read and restored at least once within 64ms.

• ***Performance evaluation:*** Figure 18 shows the peak performance (w/ and w/o normalization by area) for all the solutions (We assume *DRISA* has the same power budget as GPUs). It shows that the best *DRISA* is still 54% slower than *GPU-INT*. Note that *DRISA*'s area is ~14% of GPUs, larger sized *DRISA* with more active subarrays provides higher performance. Therefore, area-normalized results (performance per area) turns to be a fairer performance metric. With this metric, *DRISA* (*1T1C-adder*) outperforms ASIC and GPU by 1.9× and 12.7×, respectively. Note that peak performance is only part of the picture, and the resource utilization shown later is another key factor.

Figure 19 shows the on-chip memory capacity and bandwidth (buffer bandwidth for ASIC and register file bandwidth for GPU are counted). First, it shows that *DRISA* has 387× more memory capacity and 54× more bandwidth than GPU and 6.8× and 15× more than ASIC solutions. This is because of the **in-situ computing architecture**. The majority of *DRISA* is DRAM cells for large capacity, and multiple subarrays are activated simultaneously for large bandwidth. Second, *3T1C* and *1T1C-adder* have lower capacity and bandwidth density among *DRISA* solutions, due to the large cell size and large add-on circuit overheads.

[1]No framework supports fixed point CNN on GPU yet, and the real scale ratio should be less than ×4 due to the unperfect utilization.

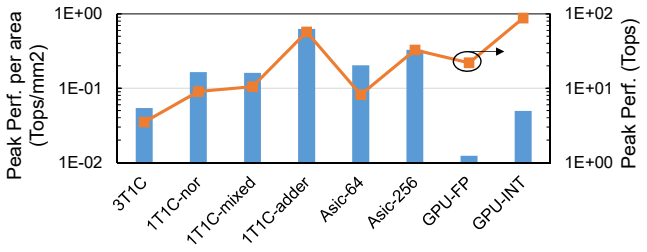

Figure 18: The peak performance (w/ and w/o normalization by area) comparison.

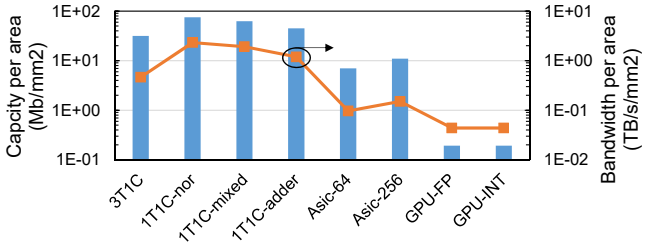

Figure 19: The on-chip memory capacity and peak on-chip bandwidth comparison (normalized by area).

Figure 20 shows the performance (frames per second) results on CNN applications with a batch size of 1/8/64, which are normalized with area. It shows that *DRISA* is 8.7× and 7.7× faster than the ASIC and GPU solutions, respectively. **It also shows a flipped result:** *1T1C-adder* with higher peak performance (Figure 18) is 12.4% slower than *1T1C-mixed*. This is because the computing and data movement costs are not balanced in *1T1C-adder*. Even though the computing is fast, the data movement turns out to be the bottleneck. For the same reason, an ASIC with more tiles is not necessarily better.

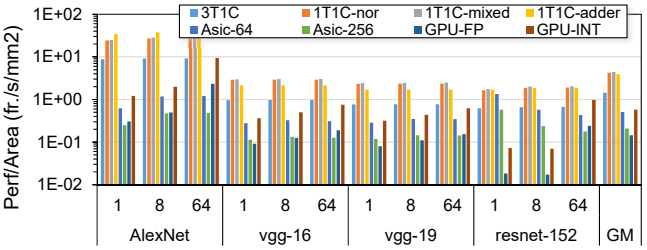

Figure 20: The performance comparison (normalized by area).

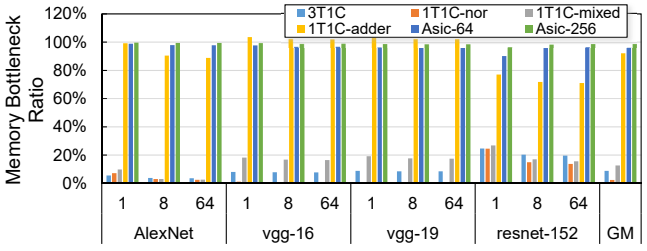

Figure 21: The memory bottleneck ratio (when computing has to wait for data).

Figure 21 shows the fraction of time when either on-chip or off-chip data movement blocks computing (data from GPU is not achievable). **It explains why *DRISA* performs better:** First, we observe *DRISA* (except for *1T1C-adder*) only spends ~10% time on memory access while others spend more than 90% time waiting for the loading data either from off-chip memory or on-chip caches[2]. The low memory bottleneck ratio is then transferred as a high resource utilization in Figure 22, which benefits from the in-situ computing architecture. Second, although both *1T1C-adder* and ASICs have more than 90% memory bottleneck ratio, *1T1C-adder* is still 7.8× faster than ASICs (Figure 20). This is because *DRISA*'s superiority also stems from its massive parallelism, not only its merging of computing and memory resources.

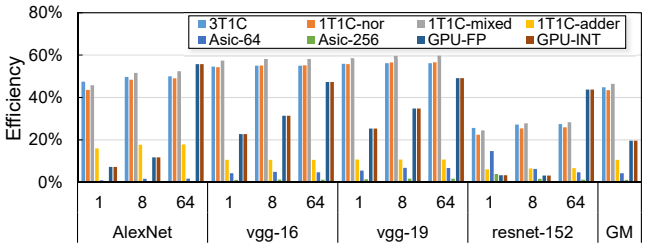

Figure 22: The resource utilization efficiency.

Figure 22 shows the resource utilization (in regard to the peak performance), which strengthens the conclusions drawn from Figure 21. *DRISA* (except for *1T1C-adder*) has an average of 45% utilization. The utilization is lower than 50% because it spends at least half of the resources on the data movement, while others are lower than 20%.

• ***Energy Evaluation:*** Figure 23 shows the area-normalized energy efficiency (frames per Joule, higher is better) comparison. First, we observe that GPUs are still the most energy-hungry solutions[3]. Second, *DRISA* is even 1.4× better than ASICs, thanks to the efficient in-situ computing architecture. In addition, DRAM process technologies have less leakage compared with the logic process, especially when considering memory cell's leakage(DRAM retention time is 64ms while eDRAM is ~100μs [64]). Third, *3T1C* is 1.94× better than *1T1C-adder*, because the logic implemented by DRAM process technologies hurts the energy efficiency.

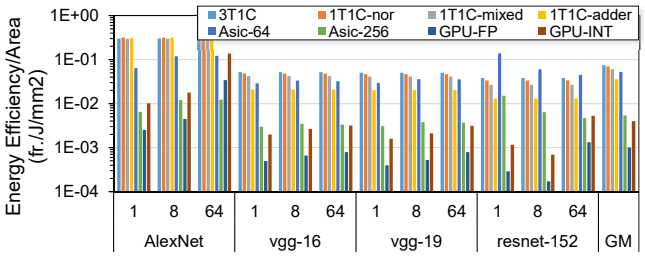

Figure 23: The energy efficiency comparison (normalized by area).

Figure 24 shows the percentage of energy spent on memory, which explains **why *DRISA* has better energy efficiency**. First, we observe that *DRISA* (except for *1T1C-adder*) spends 45% energy on memory, 1.15× smaller than others, thanks to the in-situ computing architecture. Second, *1T1C-adder* spends 92% energy on memory, due to the inefficient DRAM-implemented logics and longer wires induced by the large area overhead.

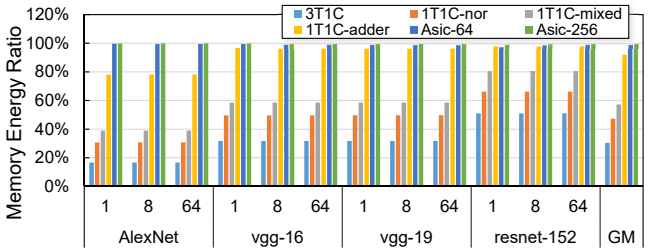

Figure 24: The data movement energy ratio

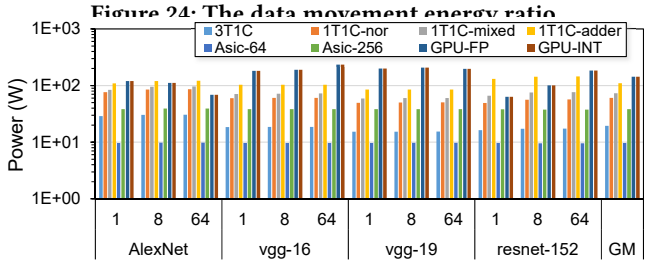

Figure 25: The power comparison.

Figure 25 shows the power comparison. It shows that even though *DRISA*-based solutions activate multiple rows simultaneously, the power consumption is still within the power budget and 54% lower than GPUs. This is due to the power budget-aware active subarray number controlling, as described in Section 4.4. We also evaluated the power density with the Hotspot tool [43]. The core temperature is well under DRAM's 85°C constraint, and existing cooling solutions are sufficient [94]. In addition, when integrating *DRISA* with PCIe like the case of GPUs, the power delivery is not a problem. When integrating with DIMM, *DRISA* needs to shut down parts of the activate subarrays in order to stay within DIMM's power budget.

• ***Cost Analysis:*** Besides the performance and energy benefits, *DRISA* can also potentially offer lower cost. Not only does *DRISA* have high area efficiency, but also three more reasons contribute to lower cost. First, a DRAM process has only 3 to 4 metal layers [91], while GPUs or ASICs with logic processes usually have more than 10 layers. Second, *DRISA* has fewer pins since it does not require connections to large external memories. This could result in a reduction in packaging cost. Combining these two factors, an industry cost analysis tool [1] shows that *DRISA* can be ~6× more cost efficient (normalized by area) than GPUs. Third, it has fewer requirements for extra memory chips (like GDDR5), since *DRISA* itself is a memory.

7 DISCUSSION: WHICH DRISA IS BETTER

1T1C-adder is the least effective design. Though *1T1C-adder* has the best (3.84× better) peak performance, its effective performance is 11% lower than others because its resource utilization is 77% lower. It is also 40% less energy efficient. In addition, it requires the largest area with 51% overhead to build adders, which becomes more difficult to manufacture. *1T1C-adder*'s problems lie in (1) large energy/area overhead of building logics with DRAM process

[2]The large percentage is not surprising since the computing and memory access are pipelined, and the computing latency is usually hidden by the data movement.
[3]We conservatively take 50% of total GPU board power as that actually spent on the GPU chip and GDDR memories.

technologies and (2) unbalanced computing and data movement capability (data movement is costly due to longer wires caused by area overhead). As a conclusion, **building too large logic with DRAM process technologies is not feasible**.

On the opposite end of the spectrum from the *1T1C-adder*, *3T1C* has minimal extra logic circuits built in the DRAM process technologies. However, it is also not the most effective design. Though it has 5.7% better energy efficiency, it suffers from 68% lower effective performance. For the area, it is 49% less dense. *3T1C*'s problem is its large cell size. As a conclusion, it shows that **only relying on memory cells for computing is not feasible, either, due to significant performance loss**, though it brings the best energy efficiency.

1T1C-nor/mixed stand somewhere between *1T1C-adder* and *3T1C* and prove to be **the best designs**. They add a few logics in DRAM process technologies but not in excess. *1T1C-mixed* is 4.7% faster than *1T1C-nor* due to its flexibility to build logics, while *1T1C-nor*'s energy efficiency is 16% better due to more memory cell based computing.

8 RELATED WORK

• ***Processing-in-Memory Architectures:*** On one hand, there is plenty of recent work on PIM [10, 12, 16, 18, 20, 30, 32, 34, 36, 42, 46, 52, 53, 67, 73, 73, 75, 76, 88, 93, 95] that built lightweight processors, reconfigurable or application-specific logics in the logic die of HMC [74] or HBM [58]. For example, Active Memory Cube [67] is a representative design with HMC. *DRISA* is different from them in that it does not rely on 3D stacking. On the other hand, earlier PIM work has integrated logics directly in the 2D DRAM die. For example, IRAM [71] has put scalar processors in the DRAM die. These work has been criticized for the difficulty of integrating complex logic with DRAM technology. *1T1C-adder* shows that even including simple adders in DRAM process technologies induces large overhead and low effective performance. *DRISA* is different since its logic circuits are simple Boolean logic operations, which are easy to build with DRAM technologies. Recently, this approach has been revisited. Buffered Comparator [11, 46]) pairs lightweight comparator with DRAMs. *DRISA* is different since it grants every BL computing capabilities to achieve high performance. As a summary, PIM sticks with a main memory position that is optimized for memory density; however, *DRISA* is a processor/accelerator optimized for performance efficiency.

• ***Revolutionary DRAM Designs:*** Recent research has evolved DRAM memory by adding additional functional capability [79]. Automata [27] has implemented a reconfigurable processor with a DRAM process. It computes with counters and finite-state machines, but *DRISA* computes with Boolean logic operations from BLs. Automata stores programmed states in the DRAM while streaming in the data, but *DRISA* is in-situ computing with all of the data stored within. Rowclone [81] has supported in-DRAM row-to-row copy with minor modification on existing DRAMs. Seshadri *et al.* [80, 82] has explored fast bulk bitwise operations in DRAMs. Compared with this work, *DRISA* has following new contributions. **(1)** *DRISA* supports more operations, e.g., NOR, shifting, and data movement, so that general purpose computing like addition is supported. **(2)** *DRISA* is a reconfigurable accelerator architecture, instead of a commodity DRAM design like BuddyRAM. Therefore, several optimizations, such as array reorganizations (Section 4.4) are proposed. Consequently, *DRISA* offers higher area/energy efficiency than prior architectures, including GPUs and ASICs. **(3)** *DRISA* provides a complete DRAM based processing architecture/solution and supports a larger range of applications (DNN/bio-informatics etc), rather than only for bitwise operations. DRAF [33] has proposed a DRAM-based FPGA, where DRAM cells are used as look-up tables. However, *DRISA* uses DRAM cells to store data.

• ***In-situ Computing Accelerators:*** Mikamonu [13–15] has proposed to compute with the NOR logic provided by the 3T1C DRAM cell or NVMs. However, their proposal does not have data movement mechanisms. Therefore, complex functions (like full adders) are not supported. Some work [19, 23, 35] has relied on emerging NVMs for computing, but *DRISA* takes use of the mature and cost-effective DRAM process technologies.

• ***Neural Network Accelerators:*** Lots of designs that accelerate NN applications with various platforms (ASIC, GPU, FPGA) [22, 39, 48, 83, 92] have been proposed. TrueNorth [29, 62, 63] computes with on-chip SRAM-based crossbars and counters. However, *DRISA* uses DRAM technologies and computes with reconfigurable Boolean logic operations. DaDianNao [22] has 36MB of on-chip memories, but *DRISA* has 512MB. EIE [39] is another design benefiting from data quantization. *DRISA* also adopts data quantization, but further stores all the intermediary data on-chip. NeuroCube [53] is a PIM architecture for NNs. Though its memory capacity is large, its performance is low (Figure 1). Most importantly, unlike all of these NN accelerating work, *DRISA* is a general-purpose reconfigurable processor. We just use NN acceleration as an application example.

9 CONCLUSION

To address the "memory wall" challenge, we propose a DRAM-based PIM design with simple Boolean logic operations to enable in-situ computing inside DRAM. To overcome the challenges induced by building accelerators with DRAM process technologies, we use simple Boolean logic operations to compute complex functions by running serially. The Boolean logic operations are provided either by the BLs themselves or by extra circuits added to the SAs. We compare four different *DRISA* designs and conclude that *1T1C-nor/mixed* are the best choices. We then present a case study where we evaluate CNN applications on *DRISA*. With the benefit of in-situ computing, *DRISA* shows 8.8× speedup and 1.2× better energy efficiency when compared with ASICs, and 7.7× speedup and 15× better energy efficiency than GPUs.

10 ACKNOWLEDGE

The authors would like to thank the anonymous reviewers and all members in SEAL lab, especially Peng Gu and Dylan Stow, for helping to improve this work. This work was funded by Samsung Semiconductor Inc, NSF 1719160, DoE DE-SC0013553 with disclaimer at http://seal.ece.ucsb.edu/doe/.

REFERENCES

[1] 2015. IC Cost and Price Model, Revision 1506, IC Knowledge LLC. (2015). http://www.icknowledge.com/

[2] 2016. 8Gb B-die DDR4 SDRAM. (2016). http://www.samsung.com/semiconductor/global/file/product/2016/06/DS_K4A8G085WB-B_Rev1_61-0.pdf

[3] 2016. NVIDIA TITAN X (pascal). (2016). http://www.geforce.com/hardware/10series/titan-x-pascal

[4] 2017. Design Compiler, Synopsys Inc. (2017).

[5] 2017. Intel Instruction Set Architecture Extensions. (2017). https://software.intel.com/en-us/intel-isa-extensions

[6] 2017. Micron Automata Processor. (2017). https://www.micronautomata.com/

[7] 2017. NVIDIA cuDNN. (2017). https://developer.nvidia.com/cudnn

[8] 2017. NVIDIA System Management Interface. (2017). https://developer.nvidia.com/nvidia-system-management-interface

[9] 2017. Torch 7. (2017). http://torch.ch/

[10] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *International Symposium on Computer Architecture (ISCA)*. ACM Press, New York, New York, USA, 105–117.

[11] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. 2016. AIM: Energy-Efficient Aggregation Inside the Memory Hierarchy. *ACM Transactions on Architecture and Code Optimization* 13, 4 (oct 2016), 1–24.

[12] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled Instructions: A Low-overhead, Locality-aware Processing-in-memory Architecture. In *International Symposium on Computer Architecture (ISCA)*. ACM, 336–348.

[13] A. Akerib, O. AGAM, E. Ehrman, and M. Meyassed. 2014. Using storage cells to perform computation. (dec 2014). US Patent 8,908,465.

[14] Avidan Akerib and Eli Ehrman. 2014. In-memory computational device. (nov 2014). US Patent App. 14/555,638.

[15] A. Akerib and E. Ehrman. 2015. Non-volatile in-memory computing device. (may 2015). US Patent App. 14/588,419.

[16] Berkin Akin, Franz Franchetti, and James C Hoe. 2015. Data Reorganization in Memory Using 3D-stacked DRAM. In *International Symposium on Computer Architecture (ISCA)*. ACM, 131–143.

[17] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *International Symposium on Microarchitecture (MICRO)]*. ACM, 1–13.

[18] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-Data Processing: Insights from a MICRO-46 Workshop. In *Micro, IEEE*, Vol. 34. IEEE, 36–42.

[19] Mahdi Nazm Bojnordi and Engin Ipek. 2016. Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1–13.

[20] Amirali Boroumand, Saugata Ghose, Brandon Lucia, Kevin Hsieh, Krishna Malladi, Hongzhong Zheng, and Onur Mutlu. 2016. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *Computer Architecture Letters* (2016), 1–1.

[21] Kevin Chen and Lior Pachter. 2005. Bioinformatics for whole-genome shotgun sequencing of microbial communities. *PLoS Comput Biol* 1, 2 (2005), e24.

[22] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *International Symposium on Microarchitecture (MICRO)*. IEEE, 609–622.

[23] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *international Symposium on Computer Architecture (ISCA)*, Vol. 44. 27–39.

[24] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:* 1602.02830 (2016).

[25] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *arXiv:* 1511.00363 (2015).

[26] Bill Dally. 2015. The Path to Exascale. http://images.nvidia.com/events/sc15/pdfs/SC5102-path-exascale-computing.pdf. (2015).

[27] Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. 2014. An efficient and scalable semiconductor architecture for parallel automata processing. In *Parallel and Distributed Systems, IEEE Transactions on*. IEEE, 99.

[28] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: shifting vision processing closer to the sensor. In *International Symposium on Computer Architecture (ISCA)*. ACM Press, New York, New York, USA, 92–104.

[29] Steve K. Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, Shyamal Chandra, Nicola Basilico, Stefano Carpin, Tom Zimmerman, Frank Zee, Rodrigo Alvarez-Icaza, Jeffrey A. Kusnitz, Theodore M. Wong, William P. Risk, Emmett McQuinn, Tapan K. Nayak, Raghavendra Singh, and Dharmendra S. Modha. 2013. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.

[30] A Farmahini-Farahani, Jung Ho Ahn, K Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *International Symposium on High Performance Computer Architecture (HPCA)*. 283–295.

[31] 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. 2016. A 14 nm 1.1 Mb Embedded DRAM Macro With 1 ns Access. *IEEE Journal of Solid-State Circuits* 51, 1 (jan 2016), 230–239.

[32] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical Near-Data Processing for In-memory Analytics Frameworks. *Parallel Archit. Compil. Tech. (PACT), 2015 IEEE Int. Conf.* (2015), 113–124.

[33] Mingyu Gao, Christina Delimitrou, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Christos Kozyrakis. 2016. DRAF: A Low-Power DRAM-Based Reconfigurable Acceleration Fabric. In *International Symposium on Computer Architecture (ISCA)*. IEEE, 506–518.

[34] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 126–137.

[35] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G Friedman. 2013. AC-DIMM: associative computing with STT-MRAM. In *nternational Symposium on Computer Architecture (ISCA)*. ACM, 189–200.

[36] Qi Guo, Tze-Meng Low, Nikolaos Alachiotis, Berkin Akin, Larry Pileggi, James C. Hoe, and Franz Franchetti. 2015. Enabling portable energy efficiency with memory accelerated library. In *International Symposium on Microarchitecture (MICRO)*. ACM Press, New York, New York, USA, 750–761.

[37] Linley Gwennap. 2015. Skylake speedshifts to next gear. *Microprocessor Report* 29, 9 (2015), 6–10.

[38] Fatih Hamzaoglu, Umut Arslan, Nabhendra Bisnik, Swaroop Ghosh, Manoj B. Lal, Nick Lindert, Mesut Meterelliyoz, Randy B. Osborne, Joodong Park, Shigeki Tomishima, Yih Wang, and Kevin Zhang. 2014. 13.1 A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology. In *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 230–231.

[39] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *International Symposium on Computer Architecture (ISCA)*. IEEE, 243–254.

[40] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *arXiv:* 1510.00149 (2015).

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv:* 1512.03385 (2015).

[42] Byungchul Hong, Gwangsun Kim, Jung Ho Ahn, Yongkee Kwon, Hongsik Kim, and John Kim. 2016. Accelerating Linked-list Traversal Through Near-Data Processing. In *International Conference on Parallel Architectures and Compilation (PACT)*. ACM Press, New York, New York, USA, 113–124.

[43] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. 2006. HotSpot: a compact thermal modeling methodology for early-stage VLSI design. *IEEE ransactions on Very Large Scale Integration (VLSI) Systems* 14, 5 (2006), 501–513.

[44] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *arXiv:* 1609.07061 (2016).

[45] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:* 1502.03167 (2015).

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

[47] Jan Van Lunteren. 2016. Programmable Near-Memory Acceleration on ConTutto. In *OpenPower Summit*.

[48] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen. 2016. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13.

[49] Sung-Mo Kang and Yusuf Leblebici. 2003. *CMOS digital integrated circuits*. Tata McGraw-Hill Education.

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

[51] Brent Keeth, R. Jacob Baker, Brian Johnson, and Feng Lin. 2007. *DRAM Circuit Design: Fundamental and High-Speed Topics* (2nd ed.). Wiley-IEEE Press.

[52] Saugata Ghose Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand and Onur Mutlu. 2016. Accelerating Pointer Chasing in

3D-Stacked Memory: Challenges, Mechanisms, Evaluation. In *International Conference on Computer Design (ICCD)*.
[53] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *International Symposium on Computer Architecture (ISCA)*. IEEE, 380–392.
[54] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM. In *International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 368–379.
[55] Ytong-Bin Kim and Tom W. Chen. 1999. Assessing merged DRAM/Logic technology. *Integration, the VLSI Journal* 27, 2 (1999), 179–194.
[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. 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.). Curran Associates, Inc., 1097–1105.
[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[58] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, Jang Hwan Cho, Ki Hun Kwon, Min Jeong Kim, Jaejin Lee, Kun Woo Park, Byongtae Chung, and Sungjoo Hong. 2014. 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)*. 432–433.
[59] Fengfu Li and Bin Liu. 2016. Ternary Weight Networks. *arXiv:* 1605.04711 (2016).
[60] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Design Automation Conference (DAC)*. ACM Press, New York, New York, USA, 1–6.
[61] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An Instruction Set Architecture for Neural Networks. In *International Symposium on Computer Architecture (ISCA)*. IEEE, 393–405.
[62] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S. Modha. 2011. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *Custom Integrated Circuits Conference (CICC)*. IEEE, 1–4.
[63] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra S Modha. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.
[64] Mu-Tien Chang, P. Rosenfeld, Shih-Lien Lu, and B. Jacob. 2013. Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 143–154.
[65] Norman P Muralimanohar, Naveen and Balasubramonian, Rajeev and Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP Lab.* (2009), 22–31.
[66] Prashant J. Nair, Dae-Hyun Kim, and Moinuddin K. Qureshi. 2013. ArchShield: architectural framework for assisting DRAM scaling by tolerating high error rates. In *International Symposium on Computer Architecture*. ACM Press, New York, New York, USA, 72–83.
[67] R Nair, S F Antao, C Bertolli, P Bose, J R Brunheroto, T Chen, C Cher, C H A Costa, J Evangelinos, B M Fleischer, T W Fox, D S Gallo, L Grinberg, J A Gunnels, A C Jacob, P Jacob, H M Jacobson, T Karkhanis, C Kim, J H Moreno, J K O'Brien, M Ohmacht, Y Park, D A Prener, B S Rosenburg, K D Ryu, O Sallenave, M J Serrano, P D M Siegl, K Sugavanam, and Z Sura. 2015. Active Memory Cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development* 59, 2/3 (mar 2015), 17:1–17:14.
[68] David Harris Neil Weste. 2006. *CMOS VLSI Design: A Circuits And Systems Perspective, 3/E*. Pearson.
[69] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. 2016. Recurrent Neural Networks With Limited Numerical Precision. *arXiv:* 1608.06902 (2016).
[70] J. M. Park, Y. S. Hwang, S. W. Kim, S. Y. Han, J. S. Park, J. Kim, J. W. Seo, B. S. Kim, S. H. Shin, C. H. Cho, S. W. Nam, H. S. Hong, K. P. Lee, G. Y. Jin, and E. S. Jung. 2015. 20nm DRAM: A new beginning of another revolution. In *2015 IEEE International Electron Devices Meeting (IEDM)*. 26.5.1–26.5.4.
[71] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *Micro, IEEE* 17, 2 (1997), 34–44.
[72] David A Patterson and John L Hennessy. 2013. *Computer organization and design: the hardware/software interface*. Newnes.
[73] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das. 2016. Scheduling techniques for GPU architectures with processing-in-memory capabilities. In *International Conference on Parallel Architecture and Compilation Techniques (PACT)*. 31–44.
[74] J Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *Hot Chips*, Vol. 23.
[75] Seth H. Pugsley, Jeffrey Jestes, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. Comparing Implementations of Near-Data Computing with In-Memory MapReduce Workloads. In *Micro, IEEE*, Vol. 34. IEEE, 44–52.
[76] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, A Buyuktosunoglu, A Davis, and F Li. 2014. NDC: Analyzing the Impact of 3D-Stacked Memory+ Logic Devices on MapReduce Workloads. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*.
[77] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv:* 1603.05279 (2016).
[78] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
[79] Vivek Seshadri. 2016. Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems. *arXiv:* 1605.06483 (2016).
[80] V Seshadri, K Hsieh, A Boroumand, D Lee, M A Kozuch, O Mutlu, P B Gibbons, and T C Mowry. 2015. Fast Bulk Bitwise AND and OR in DRAM. *Computer Architecture Letters* PP, 99 (2015), 1.
[81] Vivek Seshadri, Michael A. Kozuch, Todd C. Mowry, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, and Phillip B. Gibbons. 2013. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In *International Symposium on Microarchitecture (MICRO)*. ACM Press, New York, New York, USA, 185–197.
[82] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2016. Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM. *arXiv:* 1611.09988 (2016).
[83] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From High-Level Deep Neural Models to FPGAs. In *International Symposium on Microarchitecture (MICRO)*. IEEE.
[84] George Sideris. 1973. INTEL 1103-MOS memory taht defied cores. *ELECTRONICS* 46, 9 (1973), 108–113.
[85] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:* 1409.1556 (2014).
[86] Young Hoon Son, O. Seongil, Yuhwan Ro, Jae W. Lee, and Jung Ho Ahn. 2013. Reducing memory access latency with asymmetric DRAM bank organizations. *International Symposium on Computer Architecture (ISCA)* 41, 3 (jul 2013), 380.
[87] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *arXiv:* 1409.4842 (2014).
[88] Pedro Trancoso. 2015. Moving to Memoryland: In-memory Computation for Existing Applications. In *International Conference on Computing Frontiers*. ACM, 32:1—-32:6.
[89] G. Venkatesh, E. Nurvitadhi, and D. Marr. 2016. Accelerating Deep Convolutional Networks using low-precision and sparsity. *arXiv:* 1610.00324 (2016).
[90] Oreste Villa, Daniel R. Johnson, Mike Oconnor, Evgeny Bolotin, David Nellans, Justin Luitjens, Nikolai Sakharnykh, Peng Wang, Paulius Micikevicius, Anthony Scudiero, Stephen W. Keckler, and William J. Dally. 2014. Scaling the Power Wall: A Path to Exascale. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 830–841.
[91] Thomas Vogelsang. 2010. Understanding the Energy Consumption of Dynamic Random Access Memories. In *International Symposium on Microarchitecture (MICRO)*. IEEE, 363–374.
[92] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. 2015. Deep Image: Scaling up Image Recognition. *arXiv:* 1501.02876 (2015).
[93] Mahmut Kandemir Mustafa Karakoy Xulong Tang, Orhan Kislal. 2018. Data Movement Aware Computation Partitioning. In *International Symposium on Microarchitecture (MICRO)*.
[94] Yasuko Eckert Nuwan Jayasena and Gabriel Loh. 2014. Thermal Feasibility of Die-Stacked Processing in Memory. In *WoNDP: 2nd Workshop on Near-Data Processing, International Symposium on Microarchitecture*. IEEE.
[95] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: Throughput-oriented Programmable Processing in Memory. In *International Symposium on High-performance Parallel and Distributed Computing*. ACM, 85–98.
[96] Tao Zhang, Ke Chen, Cong Xu, Guangyu Sun, Tao Wang, and Yuan Xie. 2014. Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. In *International Symposium on Computer Architecture (ISCA)*. 349–360.
[97] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv:* 1606.06160 (2016).