DASM: Data-Streaming-Based Computing in Nonvolatile Memory Architecture for Embedded System

Liang Chang, *Student Member, IEEE*, Xin Ma, Zhaohao Wang, *Member, IEEE*, Youguang Zhang, *Member, IEEE*, Yufei Ding, Weisheng Zhao, *Fellow, IEEE*, and Yuan Xie, *Fellow, IEEE*

***Abstract*— Emerging nonvolatile memories (NVMs), including resistive RAM (RRAM), phase-change memory (PCM), and magnetic RAM (MRAM), have opened up new pathways for Computing-In-Memory (CIM). Those NVM technologies can achieve energy-efficient computational operations with only minor modification of the peripheral circuits. Despite many advantages provided by computational NVMs, parallelism is not sufficiently explored in such CIM designs. To break through this limitation on performance gain, we propose a data-streaming design for the NVM-based CIM (e.g., DASM) by leveraging the underlying parallelism in the hardware. DASM benefits from the massive parallelism of data-streaming computing, reduction in data movement of the CIM, and the nonvolatility of memory arrays. Specifically, data streaming operations can be implemented with CIM bitwise operations in both read-out and write-in procedures. In addition, we use the multilevel power gating for the memory array and connections to further boost the performance. Finally, we study a case of inference process for the quantized deep-neural-network-based on the DASM design. DASM architecture achieves 47.8×, 5.1×, 2.1× speedup compared to the NVIDIA Jetson TK1 embedded GPU board, Intel Xeon E5-2640 CPU, the state-of-the-art field-programmable gate array (FPGA) design, with much lower power consumption.**

***Index Terms*— Accelerator, binary neural network, data streaming, memcomputing, nonvolatile memory (NVM).**

Manuscript received November 12, 2018; revised February 17, 2019; accepted April 17, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61704005, Grant 61627813, and Grant 61571023; and in part by the Program of Introducing Talents of Discipline to Universities under Grant B16001. *(Corresponding authors: Zhaohao Wang; Weisheng Zhao; Yuan Xie.)*

L. Chang, and Y. Zhang are with the Fert Beijing Research Institute, School of Electronic and Information Engineering, Beihang University, Beijing 100191, China (e-mail: liang.chang@buaa.edu.cn; zyg@buaa.edu.cn).

X. Ma and Y. Xie are with the SEAL Laboratory, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: xinma@ece.ucsb.edu; yuanxie@ece.ucsb.edu).

Z. Wang and W. Zhao are with the Fert Beijing Research Institute, School of Microelectronics, Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University, Beijing 100191, China, and also with the Beihang-Geortek Joint Microelectronics Institute, Qingdao Research Institute, Beihang University, Beijing 100191, China (e-mail: zhaohao.wang@buaa.edu.cn; weisheng.zhao@buaa.edu.cn).

Y. Ding is with the Computer Science Department, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: yufeiding@cs.ucsb.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2912941

I. Introduction

RECENT developments of nonvolatile memories (NVMs) promote advanced Computing-in-Memory (CIM). By repurposing memory structures, certain NVMs have the capability of performing logic and arithmetic operations beyond data storage [1]–[3]. This modification allows computing all the instructions with/within the memory, where the abundant energy consumption from data movement can be saved. Compared with traditional CIM, the NVM-based CIM requires less modification on the memory array peripheral circuits and can utilize the in-memory analog signals with improved energy efficiency [4]. Therefore, CIM designs with NVMs turn out to be good candidates for breaking through the "power wall" and "memory wall" [5], [6].

Despite the above advantages, the parallelism is not fully explored for NVM-based CIM, hindering further performance improvements. Previous efforts enable CIM with parallelism by simultaneously activating multiple rows, subarrays, and banks in memory arrays for bitwise operations [7], [8]. However, high-level parallelism is still missing in the hardware design. We accomplish such parallelism by using data-streaming models, which is also successfully implemented in software programming for improved throughput and efficient parallelism [9], [10]. The idea of data-streaming models is to completely decentralize the dataflow processor, to eliminate all the large hardware structures, and to make superscalars nonscalable [11], [12]. These features of data-streaming models match well with that of CIM designs, leading to significant performance improvement.

The advantage of a data-streaming design with NVM-based CIM benefits from multifold sources. The first one is the massive data parallelism. NVMs are composed of several thousands of arrays. Each of these arrays is transformed into a single-instruction–multiple-data (SIMD) processing unit that can compute concurrently in the data-streaming models. The second source is a reduction in data movement. The intensive data-streaming operations can be implemented with the energy-efficient CIM operations within both the read-out and write-in phases. The third source is the nonvolatility of data storage. Power gating methods can be adapted to shut

down the memory arrays not immediately used in the data-streaming execution.

In this paper, we present the data-streaming computing NVM architecture, namely, DASM, which combines the data-streaming model with the CIM for improved throughput efficiency and reduced energy consumption. We choose the emerging NVMs as the computing memory of DASM, utilizing their nonvolatility and ability to conduct CIM. Explicitly, we reconfigure the NVMs with communication buffers to enable CIM logic operations within both the read and write memory accesses and design an additional power gating strategy to further save energy. Based on our study of the binary convolutional neural networks (BCNNs), DASM can achieve 47.8×, 5.1×, and 2.1× speedup, compared to that of the NVIDIA Jetson TK1 embedded GPU board, CPU, the state-of-art field-programmable gate array (FPGA) design. Moreover, the proposed DASM-BCNN improves the throughput efficiency up to 235.1 Img/Sec/Watt, in contrast to the throughput efficiency of 35.8 Img/Sec/Watt with the state-of-the-art FPGA design. Based on these, our contribution can be listed as follows.

1) We propose a methodology, namely, DASM, to design an NVM-based CIM architecture with the data-streaming model. DASM contains both local and global data streaming (GDS) patterns to improve the parallelism of the data computation and communication of the CIM architecture. Both read-out and write-in operations of the memory array are employed as the computation patterns. The computation and communication operations can overlap with the access operations hence to reduce the latency as well as the energy dissipation.
2) We design a CIM-architecture based on the DASM methodology for accelerating the computation of the BCNN called DASM-BCNN. We construct the computation pattern with computing memory arrays, communication buffers, and NVM-based reconfigurable logic. We develop multiple strategies to reduce energy consumption. Although the CIM operations significantly save the energy of data movement, we additionally leverage the nature of dataflow and NVMs by power gating the memory arrays during the execution.
3) We compare DASM-BCNN architecture to the state-of-art architectures and accelerators by using the CIFAR-10 BCNN model. Our experiment results show that the proposed DASM-BCNN can provide 5.1×, 2.1× and 2.5× speedup compared to CPU, FPGA, and CIM with digital process unit (DPU) implementations. Moreover, DASM-BCNN saves significant power in contrast to the GPU, CPU, and FPGA.

The rest of this paper is organized as follows. Section II presents the study of the NVM-based computing and data-streaming model. Section III introduces the streaming computation methodology and computation pattern based on NVM technology. In Section IV, we propose the architecture of DASM-BCNN. In Section V, we discuss the mapping method and programmability. The evaluation results and analyses are demonstrated in Section VI. Section VII concludes this paper.

II. Background and Related Work

This section briefly describes the limitation of the Von Neumann model and the opportunities to improve the parallelism of the hardware architecture using the "Memcomputing"[1] model, and data-streaming model. Afterward, we introduce NVM technologies.

A. Von Neumann Model and Memcomputing Model

Modern computer architecture based on the Von Neumann model has obtained significant success in the out-of-order multicore and multicore systems [15], [16]. However, as the technology scales down, both the multicore and the multicore systems meet several challenges such as memory wall, power wall, and even the "scaling wall" [17]–[19]. Moreover, the execution of the Von Neumann model contains several serial computing sections, which constrains the promotion of parallelism and the development of a superscalar processor. For example, sequential executions are guided by the program counter and control instructions. The following instructions can only be executed after finishing the previous load–store operation using the off-chip memory [20].

The Memcomputing model is motivated to surpass the Von Neumann model for improving performance and energy efficiency [14], [21]. The goal of the Memcomputing model is to build the UMM for in-data processing, where computations are performed in and with memory. The element of the computation (such as memory array, which we call mem-processor or MP in this paper) relies on memory units of functional polymorphism, where the computational memory can be reconfigured as many functions [21], [22]. The Memcomputing model is also of intrinsic parallelism [21]. It allows all the collected data to operate at the same time, in a certain cooperative fashion. The novel computing paradigm using processing embedded in memory has demonstrated many advantages, compared to the typical computing paradigm [14]. However, the computing paradigm is efficiently deployed in software rather than in hardware. Recently, the concept of CIM matches Memcomputing model regarding computing locally in the memory, but CIM does not completely satisfy the functional polymorphism and intrinsic parallelism. In this paper, we propose a cooperative fashion using CIM combined with a data-streaming model and reconfigurable logic to maximize the parallelism.

B. Data-Streaming Model

Data-streaming model attracts a renewed interest due to its ability to express parallelism efficiently at the software level [23], [24]. The main characteristic of the data-streaming model is that the execution of an operation depends only on the availability of its input data [10]. The computation consists of a phase whereby all parallel nodes (like MP) in the system perform some local computation, and cast (e.g., unicast and multicast) results to nearby nodes [24]. Afterward, the node is

[1]This is a single word. The concept of Memcomputing was proposed by [13], [14], which means using any form of memory (e.g., digital or analog memristor) to implement the universal memory machine (UMM).

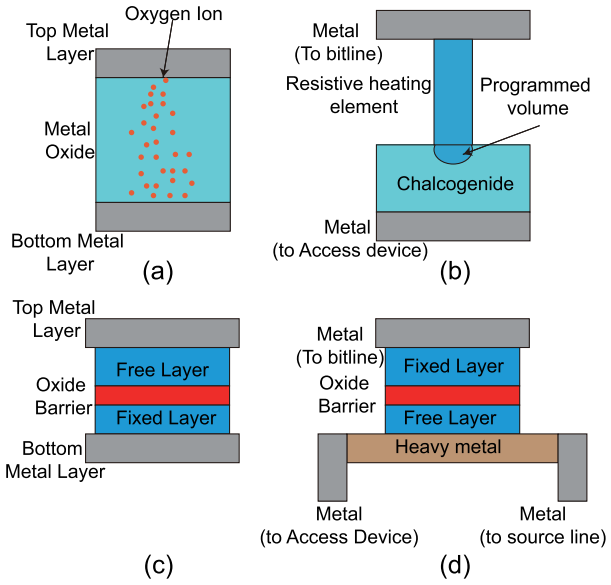


Fig. 1. Memory bit cell of the emerging NVMs. (a) RRAM. (b) PCM. (c) STT-MRAM. (d) SOT-MRAM.

ready to accomplish another stream computation. Such stream processing can significantly improve the throughput efficiency and decrease the latency for the computation [9], [25]. We integrate the data-streaming model into the memory array and Cluster of DASM, aiming to improve the execution efficiency and to reduce the communication latency.

C. Nonvolatile Memories

Emerging NVMs have been used to develop the working memory and CIM, thanks to their nonvolatility, low-power consumption, CMOS-compatibility [7], [26]. Fig. 1 presents the memory cells of different NVMs.

1) Resistive RAM (RRAM) works by changing the resistance of a dielectric solid-state material; an RRAM cell is composed of two metal layers on the top and bottom, which are separated by a metal oxide layer, as shown in Fig. 1(a) [27].
2) Phase-change memory (PCM) exploits the unique behavior of chalcogenide glass that can be switched between amorphous and crystalline phases [28]. The heat produced by the resistive heating element (generally made of TiN) is used either to make chalcogenide glass amorphous by quickly heating and quenching to switch the glass to a crystalline state by holding it in its crystallization temperature range for some time, as shown in Fig. 1(b).
3) Magnetic RAM (MRAM) is a sandwiched structure that includes three layers: free layer, oxide barrier, and the fixed layer (or named reference layer), as shown in Fig. 1(c). Both free and reference layers (FL and RL) are composed of ferromagnetic material, while MgO is usually used as the tunneling oxide barrier. The binary value is represented with the low- or high-tunneling magnetoresistance, which is governed by the parallel (P) or antiparallel (AP) magnetization alignments between FL and RL [29].

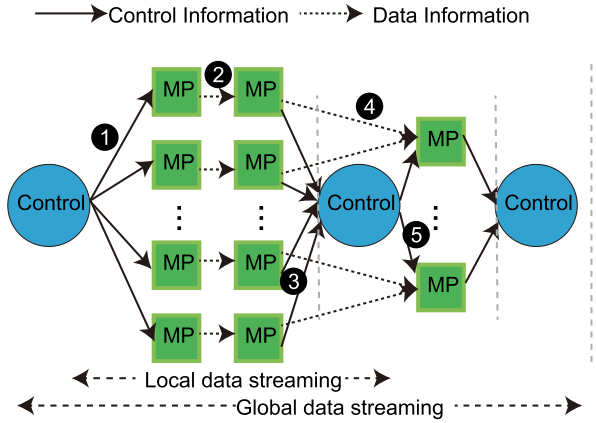


Fig. 2. Overview of DASM. The control unit and several MPs are organized as LDS. Several LDSs are organized as GDS.

4) Fig. 1(d) introduces an emerging MRAM, namely, spin–orbit torque (SOT)-MRAM, the memory bit value is stored in the magnetic tunneling junction (MTJ) above heavy metal (HM). SOT-MRAM utilizes the effects of SOTs to write the memory cell. An electrical current passing through the HM layer can effectively switch the magnetization of the FL as well as the bit value, utilizing the SOT generated by the spin Hall effect (SHE) or the Rashaba effect [26], [30], [31]. SOT-MRAM has many advantages compared to the conventional MRAM [spin-transfer torque (STT)-MRAM] such as separated read and write paths and lower write energy.

Such emerging NVMs can be used to design the MP of the Memcomputing model. In DASM, different MPs combine with the data-streaming model to improve parallelism and energy efficiency. However, the Dark Silicon may be unavoidable even in the CIM design with massive parallelism operations and running big data workloads [32], [33]. Fortunately, the power gating technology can be used in NVM technology, benefiting from the fact that the data stays inside the memory array for a long time without the power supply. We can turn off some parts of the DASM to save energy for the running parts, and hence mitigate the Dark Silicon problem. Note that our DASM architecture design is friendly to different types of NVMs introduced above, because these NVMs can be modified to perform different logic and arithmetic operations.

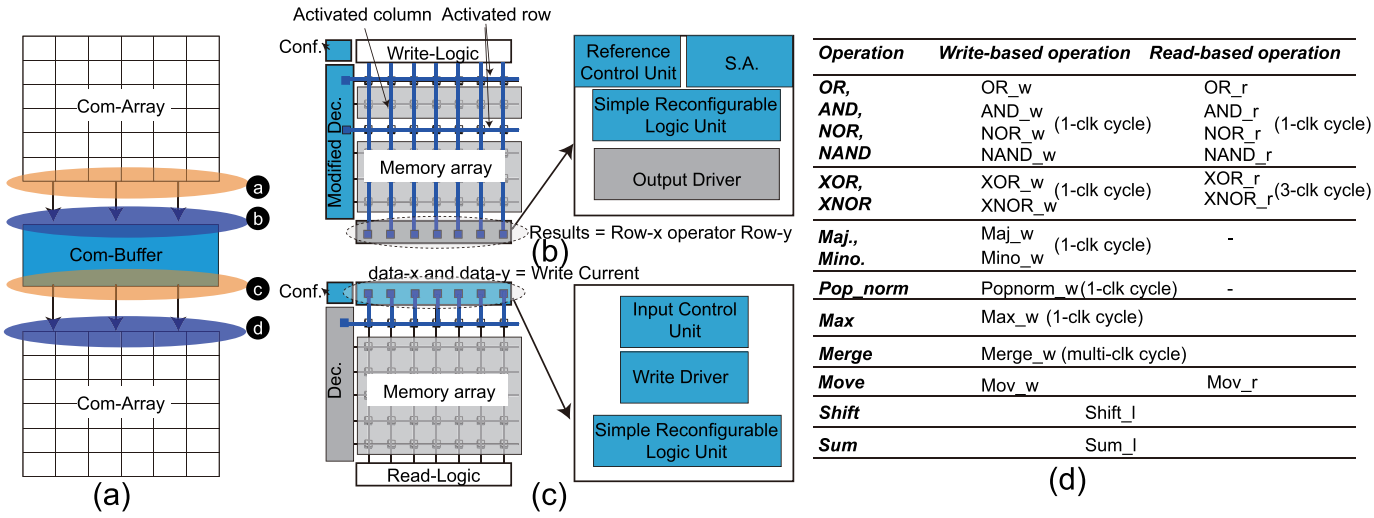
III. Streaming Computation Methodology

This section provides an overview of the DASM methodology and the data streaming pattern developed by the memory elements. With this data-streaming pattern, we introduce a study of the BCNN to reveal the computation pattern of DASM.

A. Overview of DASM

DASM can be viewed as a design methodology rather than a fixed design by itself, which makes a case for developing an accelerator using reconfigurable and plug-and-play building components. The MP can be memory elements such as NVM arrays and reconfigurable logic arrays.

Fig. 2 provides an overview of the DASM computation methodology including local data streaming (LDS) and GDS



Operation	Write-based operation	Read-based operation
OR, AND, NOR, NAND	OR_w, AND_w, NOR_w, NAND_w (1-clk cycle)	OR_r, AND_r, NOR_r, NAND_r (1-clk cycle)
XOR, XNOR	XOR_w, XNOR_w (1-clk cycle)	XOR_r, XNOR_r (3-clk cycle)
Maj., Mino.	Maj_w, Mino_w (1-clk cycle)	-
Pop_norm	Popnorm_w (1-clk cycle)	-
Max	Max_w (1-clk cycle)	
Merge	Merge_w (multi-clk cycle)	
Move	Mov_w	Mov_r
Shift	Shift_l	
Sum	Sum_l	

(d)

Fig. 3. Computation pattern and operations. (a) LDS among different computing memory arrays (Com-Array) and communication buffer (Com-Buffer). (b) and (c) Read-based and write-based array with its modification. (d) Operations supported by the read-based and write-based MPs. Glossary: Configure Unit (Conf.), SA, Decoder(Dec.).

computation strategies. The LDS consists of distributing the control information to the MP and data transfer between nearby MPs. First, each control unit distributes the task configuration and control information to the MPs and the connection between MPs in offline mode. Second, the MP starts to compute and transfers the data to the connected MP in online mode. Each read and write operation of the MP is also the computation operation form as a stream from one MP (read) to the connected MP (write).

The GDS includes several LDSs and networks to communicate between components such as MPs and control logic with consideration of the task topology, data mapping, and data dependence. Reference [34] motivates the concept of GDS, but DASM employs NVMs memory array as the process element (PE), and hence to further reduce the data movement between memory and PEs. Fig. 3(a) provides an example of the GDS; the control unit is used to collect information from the previous LDS and transfers data to the corresponding MPs [④ and ⑤]. The data are written into the MP, and the forward process is executed following the control information of the GDS. The length of the GDS can be optimized for higher MP utilization under a different task topology.

Fig. 3(a) describes the flow of the computation, and we assume each of the memory arrays (including the buffer) with the computing capacity. Stage ⓐ: From the upper computing array (Com-Array), a read-based computing operation is executed as a bitwise operation such as AND, OR, and XNOR. Stage ⓑ: The data from the memory array are written into the communication buffer (Com-Buffer) with a reduction operation (or bitwise operation) such as majority/minority, max/min operations. In DASM, both the Com-Array and Com-Buffer are capable of computing and can be reconfigured to implement different operations. However, operations supported by Com-Array and Com-Buffer are different. They cannot be reconfigured to become each other since they have different peripheral circuits providing different operations. The Com-Array is used to store static data and the dynamic data which are performing the computation in/with the array. Although Com-Buffer works as a buffer performing computation and caching, the intermediate data for accelerating the computation of the Com-Array. We separated the framework of Com-Array and Com-Buffer for trading off the access latency against the capacity. The capacity of Com-Buffer is smaller than that of the Com-Array and Com-Buffer contains less reconfigurable logic compared to the Com-Array, leading to lower access latency and power consumption. In addition, Com-Buffer has more routing resources for available providing data streaming between different Com-Arrays. Stages ⓒ and ⓓ: the Com-Buffer and Com-Array are used to execute read-out and write-in operations, respectively. Finally, results are stored in the lower Com-Array. Therefore, the data movement between different MPs is also recognized as a computing process overlapping the data transfer and mapping operation.

B. Computation Blocks

Fig. 3(b) and (c) provides the computation blocks including NVM memory array, read and write logic, reconfigurable decoder, and configurable register units. Those components provide the functional polymorphism of the Memcomputing model. For the read-based computation block, we modify the reference control unit and the sense amplifier (SA) and add the simple reconfigurable logic unit (SRLU) to support the bitwise operation such as AND, OR, XOR, and XNOR operations. In order to perform those bitwise operations, two or more rows of the memory array are activated (on), and the modified SA reads the correct value by configuring the reference circuit [2], [35]. SRLU provides the necessary logic function such as shift logic.

For the write-based computation patterns [shown in Fig. 3(c)], we modify the input control unit and write driver, and add the SRLU to support the bitwise operation such as AND, OR, XNOR, Majority/Minority, and Max/Min. The Com-Array can support the threshold write operation since most of NVM technologies employ the current or voltage to program

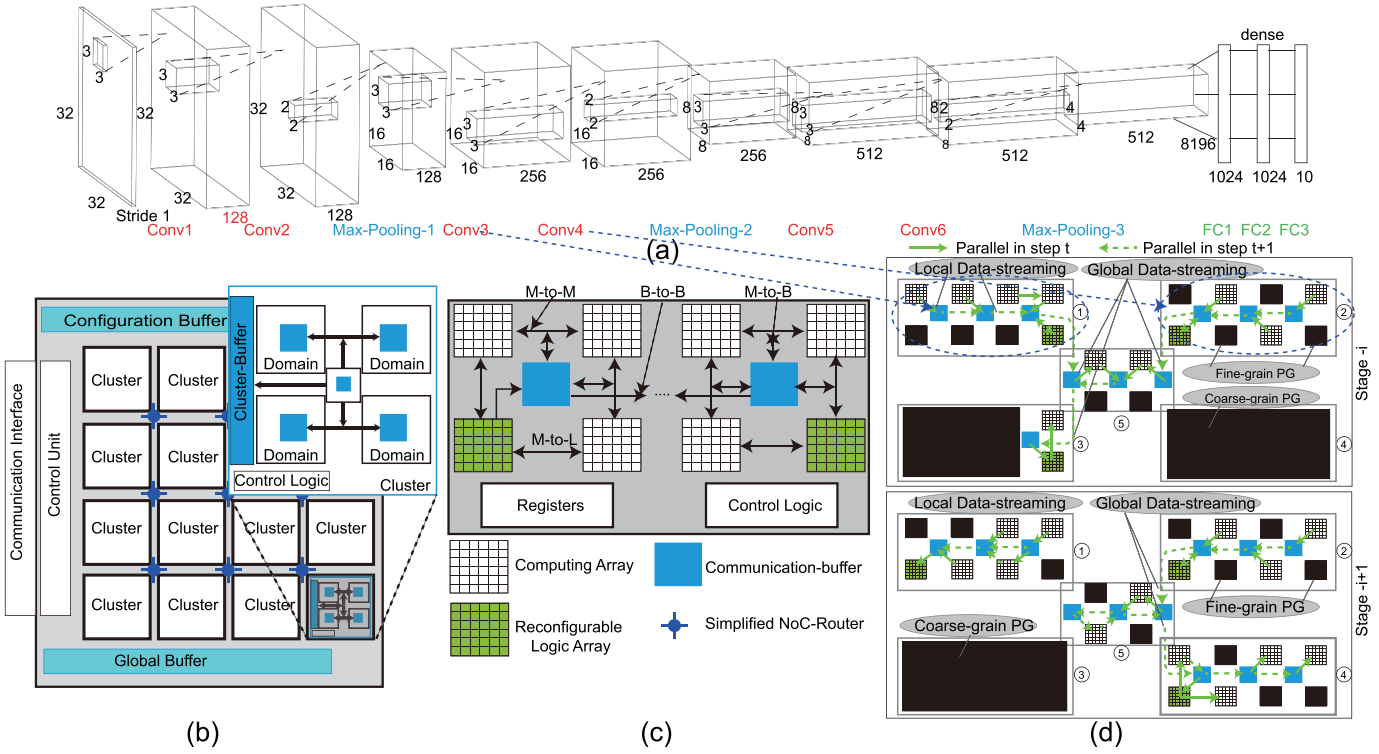


Fig. 4. Architecture of the DASM-BCNN. (a) Layers of the BCNN model for CIFAR-10 consists of CONV, POOL, and FC layers. (b) Organization of DASM consists of the Communication Interface, Control Unit, Configuration Buffer, Global Buffer, and some Clusters. (c) Connections of the Domain includes memory to buffer (M-to-B), a buffer to buffer (B-to-B), and memory to logic or logic to memory (M-to-L or L-to-M). (d) LDS and GDS with FPG and CPG.

values of the NVM bitcell (e.g., P and AP magnetization orientations of MRAM, high and low resistance states of RRAM) [36]. If the current and voltage are insufficient to modify the value of the NVM bitcell, the bitcell holds the original value (Section IV).

With the read- and write-based computation patterns, we can develop the streaming computation model overlapping memory access (write and read) and computing latency by combining the Com-Array described in Fig. 3(b). In addition, all data movements can also be recognized as computation of the intermediate data by the buffer or connection (Section IV), e.g., Computing on Road (CoR). Furthermore, the parallelism of the DASM can be improved by grouping the different streaming computation patterns (Section IV). Last but not least, we can use this streaming computation pattern to accelerate many applications such as CNN, and some data-flow benchmarks [34].

C. Study of BCNN and Computation Pattern

CNNs have demonstrated a great impact on image classification, and object detection [37], [38]. Conventionally, CNNs require millions of float-point parameters and operations. However, the deployment of CNNs on mobile phones and automobiles are limited by the computing power and memory space.

Potentially, the binary CNNs (BCNNs), using approximate binary weights and activations, hold the key to reduce the memory occupation and computation without sacrificing much accuracy in classification [39], [40]. Fig. 4(a) provides the CIFAR-10 BCNN model, including six convolutional layers (CONV), three pooling layers (POOL), and three fully connected (FC) layers. In BCNN, the complex multiply-accumulation operation of the CONV can be replaced by simple XNOR and pop-count operations, resulting in a considerable reduction of energy and area overhead [41]. Such bitwise operations of BCNN can further benefit from our DASM with the data-streaming computing pattern.

For CONV of CIFAR-10, the input is a 3-D feature map with a size of $N \times H \times L$ (e.g., CONV2 with $128 \times 32 \times 32$) filtered by weights with a size of $N \times M \times K \times K$ (e.g., $128 \times 128 \times 3 \times 3$). The weight filter slides with a stride S to filter the input feature map (ifmap) to produce an $M \times R \times C$ output feature map (ofmap), as shown in Fig. 4(a). In BCNN, both the input activation and weight parameters are constrained to $\{-1, 1\}$. The computation of CONV can be expressed as

$$\boldsymbol{y^b_{m,r,c}} = \text{Norm-Bin}\big(\text{pop-Count}\big(\text{XNOR}\big(w^b_{n,m,k}, x^b_{n,h,l}\big)\big)\big) \quad (1)$$

where $y^b_{m,r,c}$, $w^b_{n,m,k}$, $x^b_{n,h,l}$ are the layer output, binary weight, and binary input parameters, respectively. The normalization and binarization (Norm-Bin) are attached to the CONV layer [42], which are defined as

$$\boldsymbol{y} = \begin{cases} 1, & \text{if } x > 0 \\ -1, & \text{otherwise} \end{cases} \quad (2)$$

where the x and y are the input and output of the Norm-Bin layer, respectively.

In DASM, we can use the LDS strategy to map the data onto the different MPs using the computation pattern shown

in Fig. 3(b). For instance, ⓐ, ⓑ, and ⓒ present the XNOR operation, pop-count operation, and normalization operation, respectively. The ⓓ is a map operation for the next layer. We design a BCNN accelerator, e.g., DASM-BCNN, using the DASM methodology. The details are described in the following sections.

IV. DASM-BCNN Architecture Design

This section provides the architecture for the BCNN accelerator to study the detailed design of the DASM, namely DASM-BCNN.[2] Also, we introduce the detailed design including control flow, communication, and reconfiguration. The local and global streaming components with the power gating technology are introduced. Finally, we give an example for the read- and write-based computing operations.

A. Accelerator Architecture

As illustrated in Fig. 4(b), the architecture of the DASM-BCNN contains Cluster, Simplified Network-on-Chip (NoC) (SNoC) router, Control Unit, Communication interface, and configuration and global buffers.[3] **SNoC** is used to connect nearby Clusters, which is optimized for the neural network (NN) accelerator [43]. **Control Unit** of the Cluster serves as two purposes. First, it distributes the configuration information and data to each Cluster in the offline mode. Second, it regulates the status and controls the dynamic information (message) flowing through each Cluster. **Communication interface** is used to connect the host processor such as a CPU. Also, it can be used to communicate with other DASMs that extend the scalability of the architecture. The configuration and global buffers are developed for transferring the configuration information and caching source/results data, respectively.

Cluster is a hierarchy structure including configuration registers, necessary control logic, Cluster buffer, and Domains. Different Domains are organized by the *H-tree* structure for the communication between using an additional simplified Domain whose size is smaller than the normal Domains. The Cluster-buffer is used to communicate the local cluster and nearby clusters, and data transfer between cluster and configuration/global buffers.

Domains consists of Com-Arrays, Com-Buffers, and reconfigurable logic array units (RLAUs) as shown in Fig. 4(c). In the study of BCNN, Com-Array is used to store the weight parameters and input activations, and provides the readout based XNOR operation; while Com-Buffer provides the reduction operation (e.g., pop-count) and caches the intermediate data for mapping into the subsequent Com-Array. The data-streaming patterns exist only between Com-Array and Com-Buffer. Among Com-Arrays (Com-Buffers), only normal memory access operations are developed by considering the parallelism. RLAU is designed by the lookup table(LUT) and NVM technologies serving as the DPU for those operations which are difficult or inefficient to implement with the memory array. Compared to SRLU shown in Fig. 3(b) and (c), RLAU contains more reconfigurable resources to perform the entire algorithm such as addition and multiplication. However, SRLU is used to provide simple logic functions such as shift or comparison. The SRLU can be developed by combining several simple logic units with function switching rather than LUT and complex routing resources. All those three components are known as the MP of the DASM and can be reconfigured to support different functions.

In DASM, the function of the computing memory array and the logic can be off-line reconfigured during the mapping phase (V). We assume that all storage elements inside the DASM are NVMs. With those designs and the reconfigurable information, DASM can support various algorithms with the data-streaming model. Moreover, MPs and each hierarchy are capable of computing, which can be grouped to improve the parallelism of the computation.

B. Control Flow

The DASM-BCNN is a memory-centric computing and reconfigurable architecture. We consider that a host is used to map the data set and configure the control information into the DASM-BCNN (through the communication interface). After finishing the mapping and configuration phases, the host sends an initial command (CMD) to start the DASM-BCNN from the beginning point. During the data-streaming processing of the DASM-BCNN, the host will not be involved since the DASM-BCNN has the control unit and automatic procedure. The global control begins to distribute CMDs to the activated Cluster. Each activated Cluster runs the data processing meanwhile the control unit (Ctrl) controls the Domain to perform the data-streaming procedure. We assume that all the automation of data streaming procedures has been programed into the configuration information.

C. Design of Communication and Reconfiguration

The data communication of DASM-BCNN is based on the data streaming and processing patterns. Fig. 4(c) illustrates the communication of Domain. The Com-Array can directly move data to another Com-Array (M-to-M) or transfer data to the buffer following the streaming pattern (M-to-B) as shown in Fig. 3(b). In addition, the nearby buffers can interconnect with each other and move data via the routing resource (B-to-B). The reconfigurable logic array provides necessary digital logic functions connected to Com-Array and Com-Buffer (M-to-L). Therefore, many streaming patterns can be developed according to different routing configurations. For example, the Domain ① shown in Fig. 4(d) illustrates the computation of CONV-3 layer for CIFAR-10 using M-to-B, B-to-B, and M-to-L streaming patterns.

The reconfiguration of DASM-BCNN happens on the MPs and connections between MPs. We can reconfigure the MPs to support different bit-wise operations by setting different reference values [Fig. 5(b)] or activating functional transistors [Fig. 5(c)]. Computation patterns of BCNNs decide the reconfiguration of connections. For example, Fig. 4(d) shows communications including M-to-B, B-to-B, and M-to-L. Therefore, each required connection is reconfigured in offline mode.

[2]Note that, DASM can provide more computation patterns for more applications. It is our future work to explore more computation patterns.

[3]We assume that all buffers in the DASM are developed by the NVM technology.

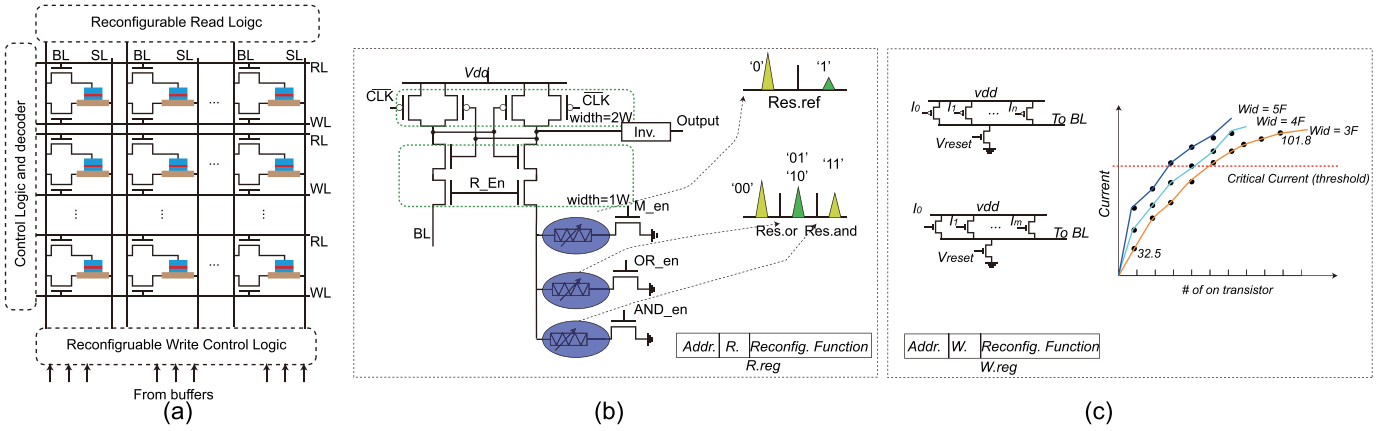


Fig. 5. Example of the SOT-MRAM as the MP. (a) Structure of the memory array including, bitcells, BL, readline (RL), wordline (WL), control logic and decoder, and reconfigurable read and write logic. (b) Modification of the read logic for the read-based memory computing and reconfigurable read register (R.reg). (c) Modification of the write logic for the write-based memory computing and reconfigurable write register (W.reg).

DASM-BCNN supports both offline and online modes for the reconfiguration. According to the control flow, there are configure phase and running phase involved. The off-line mode means that the DASM-BCNN is running in the configure phase and all Clusters can be reconfigured. In this mode, the host processor reconfigures the functions of each MPs via the configuration buffer. After the configure phase, the DASM-BCNN starts to work in the running phase, which is defined as the online mode. During this mode, the host processor continues to write the reconfiguration information into the Configure Buffer. The idle Clusters can be configured for the new function since only Global Buffer is required for writing back the results.

D. Local and Global Data Streaming Organization

DASM has a simple and somewhat different pipeline structure from a traditional processor. The typical five-stage: Instruction Fetch/Instruction Decode/Execution/Memory/Write Back (IF/ID/EX/MEM/WB) pipeline of a RISC-V processor [44], wherein the pipeline of DASM is just three stages: IF/ID/EX. Therefore, such a simple pipeline improves the execution efficiency of DASM-BCNN. We develop two data-streaming organizations: Local and Global, according to the hierarchy of DASM-BCNN.

1) Local Data Streaming and Fine-Grain Power Gating: Fig. 4 gives the example of the LDS and fine-grain power gating (FPG), in which the M-to-B, M-to-L, and B-to-B connections are configured as computation patterns. The *stage* represents a block of steps using different computing components, where many steps can result in one stage. The *step* means operations using the same components, which can consist of many operations (e.g., XNOR of Com-Array). In the Domain ①, the CONV3 is partitioned into two Com-Arrays forwarding the data to Com-Buffer (completing the XNOR, pop-count, and normalization operations). In the *Step t*, the data are transferred from Com-Array to Com-Buffer. Then, the data are moved from Com-Buffer to the nearby Com-Buffer, and the merged data flow to RLAU for computing and mapping to the next MPs in *Step t + 1*. We organize these processes as LDS performing in the same Domain. In addition, several MPs are not involved in the computation; we developed the FPG to shut down these MPs. Note that the configured data are still inside those MPs owing to the NVM technology. Therefore, those shutdown MPs can wake-up in the *stage i + 1* for the next data computation and communication. Fig. 4 also reveals that many Domains can work parallelly to execute different tasks.

2) Global Data Streaming and Coarse-Grain Power Gating: Fig. 4 also provides the example of GDS using a simplified Domain (SDomain) to communicate with different Domains. After the LDS process, the intermediate data can be moved to other Domains such as the Domain ① to ③ via ⑤. The intermediate data can be computed in the SDomain, which is a Computing-on-Road (CoR), delivering appropriate data to the next Domain [③]. The SDomain can be bypassed without processing the intermediate data. In the GDS, some Domains are not used for computation, for which we developed the coarse-grain power gating (CPG) to power off the whole Domain as suggested by Fig. 4(d) [④ and ③ of *Stage-i* and *Stage-i+1*, respectively].

E. Example of Read-Based and Write-Based Computing

The fundamental component of the architecture is the computing memory array (which we named MP as aforementioned) which supports both read-computing and write-computing operations. Fig. 3(d) illustrates modifications of the read-based and the write-based memory arrays. Fig. 5(a) gives an example of SOT-MRAM-based NVM memory array, which includes bitcells, reconfigurable read and write logic, and control and decoder logic. In the reconfigurable read logic, we develop the reconfigurable registers (R. reg) supporting different configurations to modify the value of the reference cell, as shown in Fig. 5(b). Typically, the reference cell of NVM technology is used for comparison (to bitcell) in the SA. The value of the reference cell can be modified to different values resulting in different bitwise operations as demonstrated in Fig. 5(b).[4]

[4]We only give a few examples of the read circuit. The complex bitwise operations are also supported by the additional circuit as shown in recent research works [45], [46].

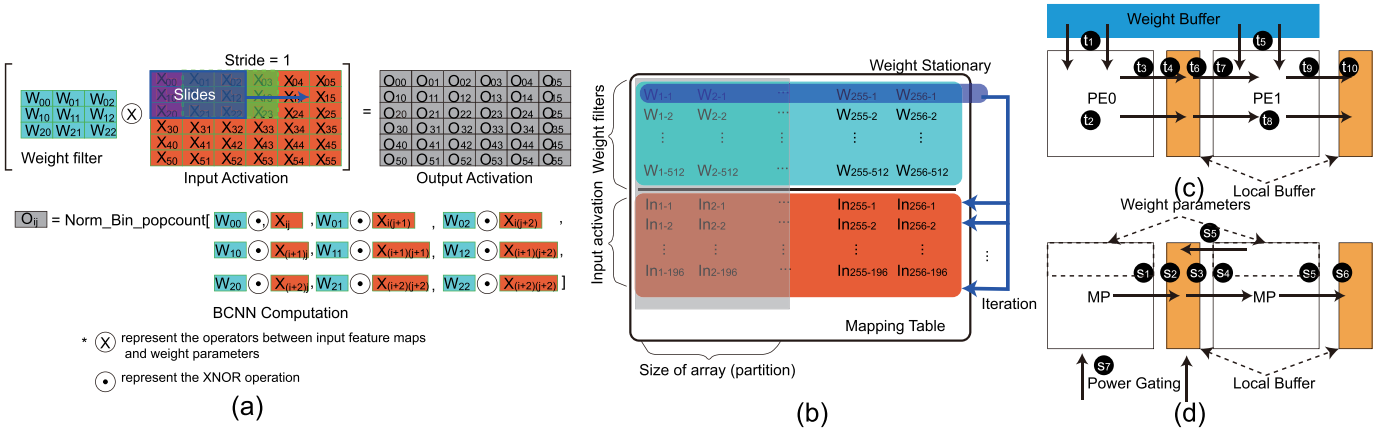


Fig. 6. Mapping of the MP (memory array). (a) Operation of the BCNN with weight filters sliding the input activation to produce the output activation. (b) Mapping table for parameters of weight filters and input activations. (c) Conventional mapping method of the CMOS PE with weight and local buffers. (d) Mapping of DASM with local buffer and power gating.

Fig. 5(c) indicates write-based computing, which employs different amounts of transistors to program the memory bitcell. The bitcell can be successfully programmed only by providing sufficient current that is larger than the critical current.[5] Both the number and width of the transistors impact the current added on the bitline (BL) as shown in Fig. 5(c). For example, the 3F (F is the feature size of the CMOS technology) and 9-bit nMOS transistor versions of write control logic can be used to design a 9-bit majority bitwise operation [36]. The write control logic supports the reconfiguration via the reconfigurable register (W.reg) for the different operations. Fig. 3(d) shows the operations supported by the read- and write-based computing memory arrays.

V. Mapping and Programmability of DASM-BCNN

This section provides the necessary mapping considerations of DASM-BCNN. The layer partition based on the mapping method is introduced. In addition, we briefly discuss the programmability of DASM-BCNN.

A. Mapping on Memory Array

The mapping method impacts the performance of the DASM-BCNN. In the mapping phase, we mainly consider several factors such as static and dynamic mapping, data reuse pattern, constraints of weight parameters, distribution of input activations, and the partition for the memory array.

1) Static Mapping: In this phase, the host processor maps weight parameters of all layers into the DASM-BCNN except the first layer. In addition, input activations of the second CONV layer are also mapped offline to the corresponding MPs.

2) Dynamic Mapping: Another mapping process happens in the running phase when the computation of one MP is completed. The results should be mapped to the next MP. For example, the output of the CONV layer is the input data of the next CONV layer or POOL layer. Therefore, the control and logic unit should support the dynamic mapping for the ifmaps.

[5]We agree that the thermal effect influences the device, but the critical current is a crucial factor for the switching of SOT-MRAM.

3) Analyze the Computation and Data Reuse Pattern: Fig. 6(a) provides the computing process of the BCNN, which contains the weight filter slides inside the input activation with a stride ($S = 1$ of this case study) to get the output activation. The process contains XNOR, pop-count, binarization, and normalization operation. During the process, both the weighting filter and input activation can be reused for the calculation. Here, in our example, the weight stationery is considered as the mapping method. Each weight row computes with all ifmaps rows to produce the output activations.

4) Weight Filter Mapping Construction: Fig. 6(b) shows the mapping table of CONV3 (CIFAR-10) for the weight filter (light blue). W_{i-j} represents the weight vector (here, 3×3 bits) for each feature, i and j denote the number of input channels (1 to 256) and output channels (1 to 512), respectively. We unfold all weight parameters following rules: the row for in-channel (ICH), the column for out-channel (OCH) mapping. For example, the weight dimension indicated in the figure is: $Row \times Column = ICH \times OCH \times K * K = 256 \times 512 \times 9$.

5) Input Activation Distribution: Fig. 6(b) indicates the input activation mapping method. We unfold all the input activation to fill the mapping table. The input activation is $ICH \times H \times L$ (e.g., $256 \times 16 \times 16$) and each ifmap is $(H-K+1) \times (H-K+1) \times K * K$ (e.g., 196×9) columns.

6) Partition the Mapping Table: We develop the mapping table according to the parameters of each layer, as shown in Fig. 6(b). Generally, the size of the memory array is smaller than the mapping table leading to layer partition including weight and input partitions. Fig. 6(b) provides a column-before-row method of partition, in which the column of the mapping table is partitioned before the row of the table (the gray window). With this method, the weight stationary can be used to improve iteration efficiency.

7) Mapping CONV and FC Layers: The DASM-BCNN architecture has the potential to improve data computing and data communication performance compared to the typical accelerator architecture. In the conventional PE-based NN accelerator architecture, as shown in Fig. 6(c), the weight parameters should load from the weight buffer (⑪ and ⑮) and communicate with host-side main memory to get enough data

Algorithm 1 Binary Neural Network Data Streaming of LDS Level

```
// The Convolutional Layer
// Let X denote the input vector of BNN
// Let W denote the weight vector of BNN
// Let Y denote the activation result
// Y = BNN_CONV(X, W) ==>
// Y =  Normalized (Popcount (XNOR(X, W)))
Conv_Layer (Y, X, W) begin
    1: input [n:0] X; input [m:0] W; output [p:0] Y;
    2: Wire [r:0] mem_read_w;
    3: Wire [r/m:0] buffer_write_w;
    4: Program_to_Array(X, W); //mapping the data
    5: for each i, j of the input and weight parameter do
    6:    mem_read_w=Mem_Read (x_i, w_j);                    ❶
          //read_based XNOR operation
    7:    buffer_write_w = mem_read_w/W_num;
          //W_num is the size of the W
    8:  Buffer_Write (buffer_write_w);                       ❷
          //write-based popcount operation
    9:  if(buffer_full) begin
   10:       Buffer_read (Buffer);
          // read the normalized results                     ❸
             Count = Count -1;
   11: end
   12: end for
end
```

Fig. 7. Algorithm of the CONV layer based on the mapping method.

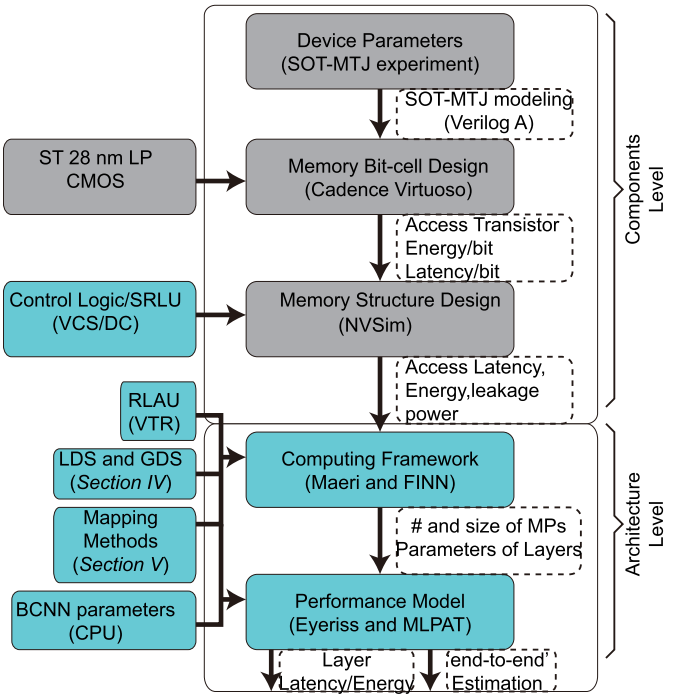


Fig. 8. Evaluation flow of the DASM-BCNN architecture. Gray box: works have been validated by the previous works. Light-blue box: contribution completed by this work.

(not shown; this is due to the capacity limitation of the buffer). The previous work has observed that the data movement occupies considerable computation latency and energy [47], [48], whereas those data movements are significantly saved in our architecture because of the NVM technology and the proposed mapping method. Fig. 6(c) reveals the layer mapping of DASM-BCNN. The mapping of memory array follows the mapping method as mentioned above, in which all weight parameters are distributed inside all the MPs. With this method, we can remove the weight buffer and reduce communication between weight buffers and MPs. In addition, the MP computing can overlap with the memory read operation with the LDS strategy ($t_2 + t_3$ to s_1). Then, the dynamic mapping operation overlaps with the write operation (ⓢ1, ⓢ2 and ⓢ4). Furthermore, we develop the power gating operation for deactivating the unused MPs following the mapping method to save energy. Fig. 7 provides the pseudocode of the CONV layer to highlight the data streaming operation.

B. Programmablity of DASM

The reusability and computation power of the DASM is the result of its compiler and the streaming algorithm developed for it. The compiler takes the architecture specifications of the target DASM on the one hand and the topology of the realization network on the other hand, and as its output, generating the streaming as parallel as possible. The output of the compiler includes hardware configuration, network communication, and mapping information. DASM-BCNN can be constructed from a collection of Verilog HDL templates, and all the reconfiguration blocks and registers can be configured to match the application. Programmability is looming as a practical issue in future large-scale computing, and DASM could hold the key to the active solution for them. In DASM, the host is responsible for running the operating system (OS) and sequential code, and DASM performs the parallel configured kernel. The host transfers the preprocessing information such as data sets and triggers DASM kernel execution. The detailed research of the compiler research is beyond the topic discussed in this paper and many recent research works have provided possible solutions [8], [49].

VI. Evaluation and Results

This section introduces the experiment setup and our considerations on the evaluation. We discuss the influence of memory size on latency and energy. The end-to-end estimation of DASM-BCNN is provided and compared to the state-of-art design.

A. Experimental Setup

In the study of the DASM-BCNN, we employ the SOT-MRAM to build the NVM MPs. We develop a components-to-architecture framework to evaluate the performance of the DASM-BCNN architecture as shown in Fig. 8. The evaluation of components level consists of developing device model, bitcell circuit, and memory array. For architecture, we develop a computing framework and a performance model.

In the device level, we abstract parameters of SOT-MTJ and employ a Verilog-A-based device model to validate the behavior of the read and write operations [50], [51].

In the circuit level, we develop a memory bitcell using the SOT-MTJ model and perform the SPICE simulation in Cadence virtuoso under STMicroelectronics CMOS 28-nm technology. We validate the behavior of the write- and read-based computing operations following the structure shown in Fig. 5(b) and (c). We get parameters including the size of access transistors, latency, and energy based on the simulation.

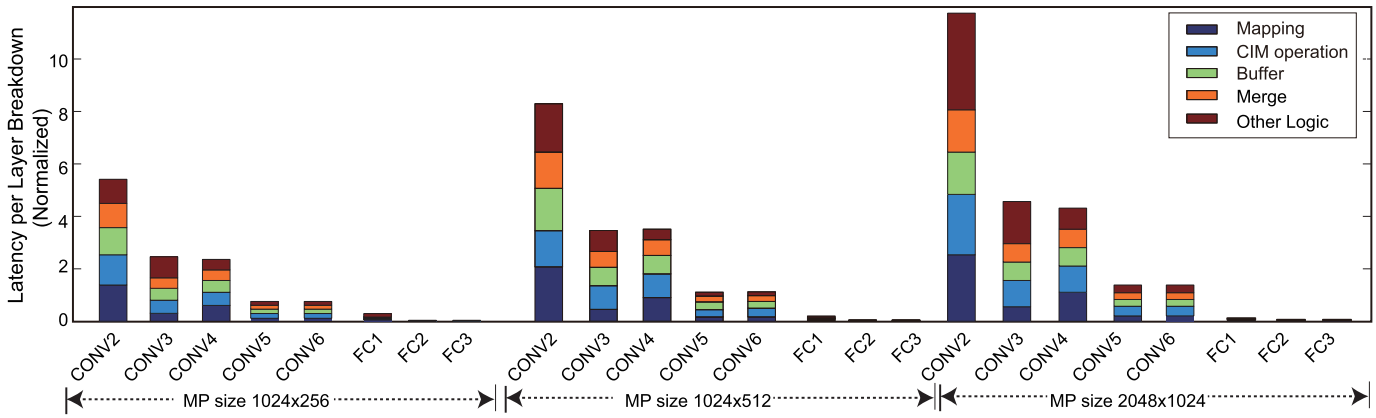


Fig. 9. Latency breakdown and latency of different layers for the DASM. The total latency consists of mapping, CIM operation, buffer, merge, and other logic latencies. The layer latency contains CONV (without CONV1), and FC layers latencies mapping on the MP-1024 × 256, MP-1024 × 512, and MP-2048 × 1024, respectively.

In the memory-array level, we evaluate the latency, energy, and leakage power of Com-Array and Com-Buffer with a modified NVSim simulator [52]. In the NVSim simulator, we add parameters of the bitcell and modify the structure of peripheral circuits based on requirements of the DASM-BCNN architecture as shown in Fig. 4. The latency and energy consumption of the control unit and the RLAU are evaluated using the Synopsys VCS and Design Compiler, and Verilog-to-Routing synthesis tool, respectively [53], [54].

In the architecture level, we employ the CIFAR-10 BCNN model whose network architecture is demonstrated in Fig. 4(a) and parameters come from [40], [42]. We develop a simulation model of the LDS and GDS for the latency and energy-efficiency analyses based on the computation framework provided by [42], [48] and [34]. For the BCNN computation, we modify the computation model provided by [34], [42],[6] and merge the analysis model with computation patterns and mapping methods developed in Sections IV and V, respectively. Parameters of BCNN are processed by CPU and mapped into the DASM-BCNN architecture. Fig. 8 provides our analysis framework, more detailed analysis and performance models have been validated in [48], [55] and [7].

Our baseline contains an embedded NVIDIA Jetson TK1 GPU board (mGPU), an NVIDIA Tesla K40 GPU (GPU), an Intel Xeon E5-2640 multicore processor (CPU), and the FPGA implementation on Xilinx Zynq-7000 SoC with an ARM Cortex-A9 embedded process (FPGA) [42]. We develop two versions of DASM-BCNN: one is for the low-power optimization with serial LDS (DASM-S), and the other is for the high-speed optimization with consideration of parallel LDS and GDS (DASM-P).

B. BCNN Layerwise Evaluation

The size of the NVM memory array is an important factor influencing the performance of DASM-BCNN. It influences the weight mapping, and input feature maps distribution approach and the number of the partition-merge operation. We evaluate the latency and energy breakdown of each of the CONV and FC layers mapping on the MP-1024 × 256, 1024 × 512, 2048 × 1024 MPs (the NVM memory array), respectively.

1) Latency Breakdown of CIFAR-10: Fig. 9 demonstrates the latency breakdown of each CONV and FC layer. The layer latency consists of data mapping latency for each MP (Mapping), read-computing latency (CIM operation), write-based computing (for buffer) and buffers latency (Buffer), data movement and data merge latency (Merge), and other logic such as latency of RLAU. Note that most of those operations are performed in parallel. For CONV layers, the latency of Mapping occupies a large proportion of the total latency since the mapping process contains buffer read, data management (for input feature maps distribution), and MP write operations. For the CONV2 and the MP-1024 × 512, Mapping occupies a quarter of the total latency. The computing operation including CIM operation, Buffer and Other Logic consumes most of the total latency for DASM-BCNN, which matches our analyses above. We also notice that the Merge operation, which includes most of data movement operation, cannot be neglected since it costs considerable latency as illustrated in Fig. 9. For CONV2 and the MP-1024 × 512, Merge costs 16.7% of total layer latency.

2) Latency Influence of Each Layer by Array Size: Fig. 9 gives the latency comparison of each layer influenced by the size of the MP (memory array). For each of the MP sizes, CONV layers cost the most of the total computation latency for the DASM-BCNN acceleration. Also, the first three CONV layers expend the majority of the computation latency due to the large input feature maps and iterations. The latency of FC layers is very low because of the parallelism of each MP and each LDS. We mainly analyze the latency of CONV layers as follows.

For CONV2, CONV3, and CONV4 layers, the MP size 2048 × 1024 of DASM-BCNN costs the highest latency, whereas the MP size 1024 × 256 costs the lowest latency. The

[6]Source code: http://synergy.ece.gatech.edu/tools/maeri/ and https://github.com/cornell-zhang/bnn-fpga.git. BNN-FPGA is a framework for FPGA design, and they provided detail computing functions for the BCNN. We add some functions based on the DASM-BCNN structure and mapping methods.

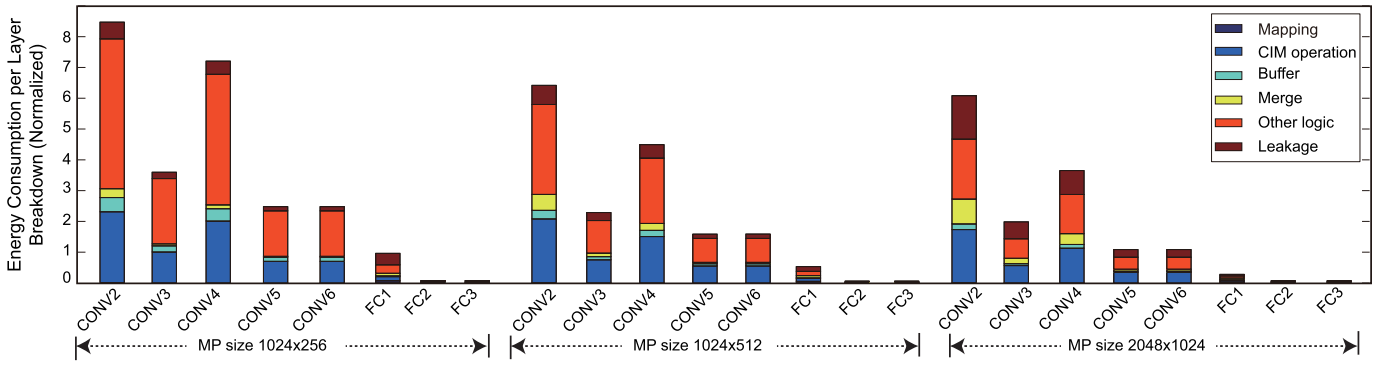


Fig. 10. Energy breakdown and energy consumed by different layers of the DASM-BCNN. The total energy consists of mapping, CIM operation, buffer, merge, other logic, and leakage energies. The layer energy contains CONV (without CONV1), and FC layers energy mapping on the MP-1024 × 256, MP-1024 × 512, and MP-2048 × 1024, respectively.

TABLE I

PERFORMANCE OF DASM COMPARED TO THE STATE-OF-THE-ART DESIGN FOR THE CIFAR-10 BNN MODEL

	Execution time per image (ms)						
Item	**mGPU**	**CPU**	**GPU**	**FPGA**	**CIM-DPU**	**DASM-S**	**DASM-P**
Conv1	-	0.68	0.01	1.13	0.68	0.68	0.68
Conv2-5	-	13.2	0.68	2.68	5.35	6.78	2.02
FC1-3	-	0.92	0.04	2.13	1.23	1.49	0.19
Total	90	14.8	0.73	5.94	7.26	8.95	2.89
Speedup	1 ×	6.1 ×	123 ×	15.1 ×	12.4 ×	10.1 ×	47.8 ×
Power(W)	3.6*	95*	235*	4.7*	2.1	0.62	1.48
Image/sec/W	3.09	0.71	5.83	35.8	68.1	182.7	235.1

- represents a value we could not measure.
* are sourced from data sheets and reference [42].

reason is that each operation latency (read and write) increases as the size of the column becomes larger. In addition, more MPs represent more parallelism since we can construct more LDS to improve operational speed. However, a large memory array only improves the column parallelism of the memory array reducing the LDS parallelism. Furthermore, larger MPs require a larger Other logic for satisfying the requirement of throughput. However, a larger logic unit increases computing latency.

3) Energy Breakdown of CIFAR-10: Fig. 10 provides the energy breakdown of each CONV and FC layer. The layer energy contains Mapping, CIM operation, Buffer, Merge, Other Logic, and Leakage energy. In each layer, the energy consumed by CIM operation and Other Logic becomes the two largest parts of the total energy. The RLAU of the Other Logic consumes considerable energy on the data movements rather than computation. However, most of the energy of CIM operation pay for computation operations. Consequently, DASM prefers using more CIM operation than logic operations performed by the DPU. In addition, the leakage energy of the memory array and logic is a nonnegligible portion of the layer energy.

4) Energy Influence of Each Layer by Array Size: Fig. 10 also demonstrates the energy consumption by each layer of different MP sizes. Similar to the latency evaluation, CONV layers consume most of the total computation energy. Among those three MP sizes, the MP size of 2048 × 1024 DASM-BCNN consumes the least energy since fewer operations are required between different MPs compared to other counterparts. However, we notice that the leakage consumption of MP-2048 × 1024 is larger than MP-1024 × 256 and MP-1024 × 512. The reason is that the mapping data of some layers do not completely occupy the corresponding MPs. For instance, CONV2 of MP-1024 × 512 requires three MPs (memory array) to map the data with 95% array occupation, where the array occupation is 60% of MP-2048 × 1024 with two MPs (memory array).

5) Discussion: Based on the above observations, we set the MP (memory array) to MP-1024 × 512 with a balance between latency and energy consumption. First, the latency of the MP-1024 × 512 is slightly higher than that of the MP-1024 × 256 but in an acceptable degree. Second, most of the memory array of the MP-1024 × 512 is occupied by weight and input feature map parameters with only minor waste.

C. End-to-End Estimation Result and Design Comparison

Table I demonstrates the execution time for several layers and the total execution time of the CIFAR-10 BNN model on the DASM-BCNN and other conventional architectures. We also build a conventional CIM-based accelerator with DPU for the pop-count operation based on the method referred to [56]. All three CIM-based accelerators (CIM-DPU, DASM-S, and DASM-P) achieve a better performance regarding the execution time and power consumption. For execution time, CIM-DPU, DASM-S, and DASM-P can achieve the speedup of 12.4×, 12.4×, and 47.8×, respectively, compared to the embedded GPU board (**mGPU**). The DASM-P also achieves 5.1×, 2.1×, and 2.51× speedup compared to CPU,

FPGA, and CIM-DPU, respectively. GPU is still the fastest architecture for the BCNN because more resource is used to perform the computation. DASM holds the potential to further reduce the execution time by adding data replication and more parallel MPs. Nevertheless, we prefer DASM to work in a low power scenario. Afterward, CIM-based accelerators outperform conventional architectures concerning power consumption. Those CIM-based accelerators cost lower power consumption since all the operations are inside and with the memory array and buffer based communication network. FPGA, CIM-DPU, and DASM-P can obtain 98%, 99%, and 99% power savings, compared to the GPU. CIM-based accelerators also cost less power consumption than the FPGA accelerator, thanks to the lower data movement between accelerator and memory. In addition, DASM has simpler routing resources than conventional FPGA. Among them, DASM-P achieves 68.5% and 29% power savings compared to FPGA and CIM-DPU, respectively. The power optimization version of DASM, namely, DASM-S, only consumes 0.62 W for the implementation of the data streaming computation pattern and power gating. As a result, the throughput of DASM can go up to 235.1 Image per second per Watt as shown in table I. This number is much higher than the other technology design, thanks to the data-streaming execution model of the DASM and the NVM technology.

VII. Conclusion

CIM architecture can solve the bandwidth problem of conventional architecture as well as reduce the energy consumption of data movement. However, the idea of CIM architecture is limited by the Von Neumann model which is an intrinsic serial computing model. In this paper, we propose the DASM methodology to build the architecture based on the Memcomputing model with the data-streaming computation pattern. We implement DASM-BCNN architecture with our proposed methodology. DASM-BCNN is a standalone system, which employs the NVM technology to build MPs including the memory array, buffers, and RLAU. In our study of the DASM-BCNN architecture, both the LDS and GDS strategies are suitable for operations of BCNN, hence to improve the energy-efficiency of the acceleration. Based on the comparison, DASM-BCNN can achieve throughput-efficiency up to 182.7 and 235.1 Image per second per Watt with DASM-S and DASM-P.

DASM is still in its infant stage, which naturally confronts this new design with challenges, problems, and barriers to meet the requirement of UMM [21]. Many commonly used operations are difficult to implement with only memory elements. For example, matrix multiplication with current CIM architecture consumes more energy than the typical algorithm logic unit. Consequently, more powerful computing operations are anticipated to be explored to move more instructions into the DASM.

Acknowledgment

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

References

[1] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, Jun. 2018.

[2] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. 53rd Annu. Des. Autom. Conf.*, Jun. 2016, p. 173.

[3] K. Cao *et al.*, "In-memory direct processing based on nanoscale perpendicular magnetic tunnel junctions," *Nanoscale*, vol. 10, no. 45, pp. 21225–21230, 2018.

[4] W. Zhao *et al.*, "Synchronous non-volatile logic gate design based on resistive switching memories," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 61, no. 2, pp. 443–454, Feb. 2014.

[5] R. Balasubramonian *et al.*, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, Jul./Aug. 2014.

[6] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2018, pp. 1–14.

[7] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, Jun. 2016

[8] P. Srivastava *et al.*, "Promise: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 43–56.

[9] T. Akidau *et al.*, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[10] A. Alexandrov, A. Salzmann, G. Krastev, A. Katsifodimos, and V. Markl, "Emma in action: Declarative dataflows for scalable data analysis," in *Proc. Int. Conf. Manage. Data*, Jul. 2016, pp. 2073–2076.

[11] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," *ACM SIGARCH Comput. Archit. News*, vol. 3, no. 4, pp. 126–132, Jan. 1975

[12] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "Wavescalar," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2003, p. 291.

[13] Y. R. Pei, F. L. Traversa, and M. Di Ventra. (2017). "On the universality of memcomputing machines." [Online]. Available: https://arxiv.org/abs/1712.08702

[14] H. Manukian, F. L. Traversa, and M. Di Ventra. (2018). "Accelerating deep learning with memcomputing." [Online]. Available: https://arxiv.org/abs/1801.00512

[15] M.-H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1599–1612, Sep. 2017.

[16] H. Tabani, J.-M. Arnau, J. Tubella, and A. Gonzalez, "A novel register renaming technique for out-of-order processors," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2018, pp. 259–270.

[17] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, 1995.

[18] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, May/Jun. 2012.

[19] D. Fan *et al.*, "Smarco: An efficient many-core processor for high-throughput applications in datacenters," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2018, pp. 596–607.

[20] M. Bakhshalipour and H. Sarbazi-Azad. (2018). "Parallelizing bisection root-finding: A case for accelerating serial algorithms in multicore substrates." [Online]. Available: https://arxiv.org/abs/1805.07269

[21] F. L. Traversa and M. Di Ventra, "Universal memcomputing machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2702–2715, Nov. 2015.

[22] W. Kang, L. Zhang, J.-O. Klein, Y. Zhang, D. Ravelosona, and W. Zhao, "Reconfigurable codesign of STT-MRAM under process variations in deeply scaled technology," *IEEE Trans. Electron Devices*, vol. 62, no. 6, pp. 1769–1777, Jun. 2015.

[23] H. Esmaeilzadeh, P. Saeedi, B. N. Araabi, C. Lucas, and S. M. Fakhraie, "Neural network stream processing core (NnSP) for embedded systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May. 2006, p. 4.

[24] S. Venkataraman *et al.*, "Drizzle: Fast and adaptable stream processing at scale," in *Proc. 26th Symp. Oper. Syst. Princ.*, Oct. 2017, pp. 374–389.

[25] D. Sun, H. Yan, S. Gao, X. Liu, and R. Buyya, "Rethinking elastic Online scheduling of big data streaming applications over high-velocity continuous data streams," *J. Supercomput.*, vol. 74, no. 2, pp. 615–636, Feb. 2018.

[26] Z. Wang *et al.*, "High-density NAND-like spin transfer torque memory with spin orbit torque erase operation," *IEEE Electron Device Lett.*, vol. 39, no. 3, pp. 343–346, Mar. 2018.

[27] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[28] H.-S. P. Wong *et al.*, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[29] M. Wang *et al.*, "Current-induced magnetization switching in atom-thick tungsten engineered perpendicular magnetic tunnel junctions with large tunnel magnetoresistance," *Nature Commun.*, vol. 9, no. 1, p. 671, 2018.

[30] M. Wang *et al.*, "Field-free switching of a perpendicular magnetic tunnel junction through the interplay of spin–orbit and spin-transfer torques," *Nature Electron.*, vol. 1, no. 11, pp. 582–588, 2018.

[31] Z. Wang *et al.*, "Proposal of toggle spin torques magnetic RAM for ultrafast computing," *IEEE Electron Device Lett.*, vol. 40, no. 5, pp. 726–729, May 2019.

[32] M. Li, W. Liu, L. Yang, P. Chen, and C. Chen, "Chip temperature optimization for dark silicon many-core systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 5, pp. 941–953, May 2018.

[33] V. Y. Raparti and S. Pasricha, "PARM: Power supply noise Aware Resource Management for NoC based multicore systems in the dark silicon era," in *Proc. 55th Annu. Des. Autom. Conf.*, Jun. 2018, pp. 1–6.

[34] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2018, pp. 461–475.

[35] S. Angizi, Z. He, and D. Fan, "Pima-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Proc. 55th Annu. Des. Autom. Conf.*, Jun. 2018, p. 162.

[36] L. Chang, X. Ma, Z. Wang, Y. Zhang, W. Zhao, and Y. Xie, "CORN: In-buffer Computing for binary Neural network," in *Proc. Des. Autom. Test Eur.*, 2019, pp. 1–6.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, Jun. 2017

[38] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power CNN-based machine learning applications," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2018, pp. 488–490.

[39] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (2016). "XNOR-Net: Imagenet classification using binary convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1603.05279

[40] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y.Bengio. (2016). "Binarized Neural networks: Training deep neural networks with weights and activations constrained to $+1$or-1." [Online]. Available: https://arxiv.org/abs/1602.02830

[41] L. Jiang, M. Kim, W. Wen, and D. Wang, "XNOR-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-IO2 DRAMs," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, Jul. 2017, pp. 1–6.

[42] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. 2017 ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 15–24.

[43] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking NoCs for spatial neural network accelerators," in *Proc. 11th IEEE/ACM Int. Symp. Netw.–Chip*, Oct. 2017, p. 19.

[44] K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep., 2015.

[45] C. Eckert *et al.*. (2018). "Neural cache: Bit-serial in-cache acceleration of deep neural networks." [Online]. Available: https://arxiv.org/abs/1805.03718

[46] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. 55th Annu. Des. Autom. Conf.*, Jun. 2018, p. 105.

[47] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[48] Y.-H. Chen, J. Emer, and V. Sze. (2018). "Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks." [Online]. Available: https://arxiv.org/abs/1807.07928

[49] Y. Ji, Y. Zhang, W. Chen, and Y. Xie, "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2018, pp. 448–460.

[50] L. Chang, Z. Wang, Y. Gao, W. Kang, Y. Zhang, and W. Zhao, "Evaluation of spin-Hall-assisted STT-MRAM for cache replacement," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2016, pp. 73–78.

[51] L. Chang *et al.*, "Prescott: Preset-based cross-point architecture for spin-orbit-torque magnetic random access memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2017, pp. 245–252.

[52] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[53] (2001). *Compiler, Design and User, RTL and Guide, Modeling*. [Online]. Available: http://www.synopsys.com

[54] J. Rose *et al.*, "The VTR project: Architecture and cad for fpgas from verilog to routing," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Feb. 2012, pp. 77–86.

[55] T. Tang and Y. Xie, "Mlpat: A power, area, timing modeling framework for machine learning accelerators," in *Proc. DOSSA Workshop*, New York, NY, USA: 2018, pp. 1–3.

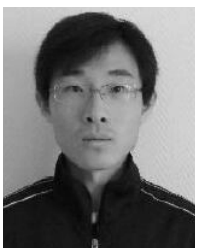
[56] S. Angizi, Z. He, F. Parveen, and D. Fan, "Imce: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proc. 23rd Asia South Pacific Des. Autom. Conf.*, Jan. 2018, pp. 111–116.

Liang Chang (S'15) received the B.S. degree from Chengdu Information and Technology University, Chengdu, China, in 2011 and the M.S. degree from Beihang University, Beijing, China, in 2014. He is working toward the Ph.D. degree in spintronics at the Fert Beijing Institute, Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University, and the School of Electronic Information and Engineering, Beihang University.

His current research interests include reconfigurable circuit design and advanced computer architectures based on emerging nonvolatile devices.

Xin Ma received the B.S. degree in physics from the University of Science and Technology of China, Hefei, China, in 2009 and the Ph.D. degree in physics from The College of William and Mary, Williamsburg, VA, USA, in 2014.

He is currently a Postdoctoral Researcher at the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include designing in-memory computing with emerging nonvolatile memory technologies.

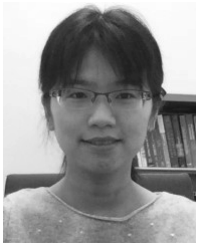


Zhaohao Wang (S'12–M'16) received the B.S. degree from Tianjin University, Tianjin, China, in 2009, the M.S. degree in microelectronics from Beihang University, Beijing, China, in 2012, and the Ph.D. degree in physics from University of Paris-Sud, Orsay, France, in 2015.

He is currently an Assistant Professor at the School of Microelectronics, Beihang University. His current research interests include the modeling of emerging nonvolatile nanodevices, design of new nonvolatile memories and logic circuits, and advanced computer architectures.

Youguang Zhang (M'13) received the M.S. degree in mathematics from Peking University, Beijing, China, in 1987 and the Ph.D. degree in communication and electronic systems from Beihang University, Beijing, in 1990.

He is currently a Professor at the Fert Beijing Institute, Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University, and also at the School of Electrical and Information Engineering, Beihang University. His current research interests include microelectronics and computer architectures.

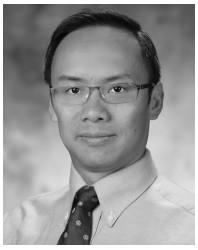


Yufei Ding received the B.S. and M.S. degrees in physics from the University of Science and Technology of China and The College of William and Mary, in 2009 and 2011, respectively, and the Ph.D. degree in computer science from North Carolina State University, in 2017. She joined the Department of Computer Science, University of California at Santa Barbara, as an Assistant Professor, in 2017. She has been actively publishing in major venues in both computer systems and data analytics areas, such as ASPLOS, PLDI, OOPSLA, VLDB, ICDE, and ICML. Her research interests reside at the intersection of compiler technology and (big) data analytics.

Weisheng Zhao (M'07–SM'14–F'19) received the M.S. degree in electrical engineering from the ENSEEIHT Engineering School, Toulouse, France, in 2004 and the Ph.D. degree in physics from the University of Paris-Sud, Orsay, France, in 2007.

From 2008 to 2009, he was at the Embedded Computing Laboratory, CEA. From 2009 to 2013, he was a Tenured Research Scientist at CNRS. He is currently a Professor at the Fert Beijing Institute, Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University, Beijing, China, and also at the School of Microelectronics, Beihang University. He has authored or coauthored more than 120 scientific papers. His current research interests include the spintronics and its related device, circuit, and architecture investigations.

Dr. Zhao is the Associate Editor of the IEEE TRANSACTIONS ON NANOTECHNOLOGY and *IET Electronics Letters*.



Yuan Xie (M'02–SM'09–F'15) received the Ph.D. degree from the Electrical Engineering Department, Princeton University, Princeton, NJ, USA, in 2002.

From 2002 to 2003, he was at IBM, Armonk, NY, USA. From 2012 to 2013, he was at the AMD Research China Laboratory, Beijing, China. From 2003 to 2014, he was a Professor at Pennsylvania State University, State College, PA, USA. He is currently a Professor at the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, electronic design automation, and VLSI design.

Dr. Xie served as the TPC Chair for HPCA 2018. He is currently the Editor-in-Chief of *ACM Journal on Emerging Technologies in Computing Systems* (JETC), the Senior Associate Editor (SAE) of *ACM Transactions on Design Automation for Electronics Systems* (TODAES), and an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. He is an expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame.