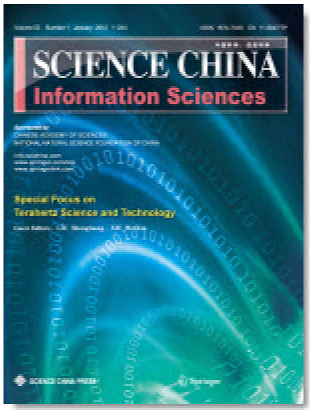


NAND-SPIN-Based Processing-in-MRAM Architecture for Convolutional Neural Network Acceleration

Journal:	*SCIENCE CHINA Information Sciences*
Manuscript ID	SCIS-2021-0973.R3
Manuscript Type:	Research Paper
Date Submitted by the Author:	31-Mar-2022
Complete List of Authors:	Zhao, Yinglin; Beihang University Yang, Jianlei; Beihang University Li, Bing; Capital Normal University Cheng, Xingzhou; Beihang University Ye, Xucheng; Beihang University Wang, Xueyan; Beihang University Jia, Xiaotao; Beihang University Wang, Zhaohao; Beihang University Zhang, Youguang; Beihang University ZHAO, Weisheng; Beihang University
Keywords:	processing-in-memory, convolutional neural network, NAND-like spintronics memory, nonvolatile memory, magnetic tunnel junction
Speciality:	Integrated Circuit and System Design < Microelectronics and Solid State Electronics

https://engine.scichina.com/doi/10.1007/s11432-021-3472-9

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

NAND-SPIN-Based Processing-in-MRAM Architecture for Convolutional Neural Network Acceleration

Yinglin ZHAO[1,5], Jianlei YANG[2*], Bing LI[3*], Xingzhou CHENG[2], Xucheng YE[2], Xueyan WANG[4], Xiaotao JIA[4], Zhaohao WANG[4], Youguang ZHANG[1] & Weisheng ZHAO[4]

[1] *School of Electronic and Information Engineering, Beihang University, Beijing* 100191*, China;*
[2] *School of Computer Science and Engineering, Beihang University, Beijing* 100191*, China;*
[3] *Academy for Multidisciplinary Studies, Capital Normal University, Beijing* 100048*, China;*
[4] *School of Integrated Circuit Science and Engineering, Beihang University, Beijing* 100191*, China;*
[5] *Qingdao Research Institute, Beihang University, Qingdao* 266104*, China*

Abstract The performance and efficiency of running large-scale datasets on traditional computing systems exhibit critical bottlenecks due to the existing "power wall" and "memory wall" problems. To resolve those problems, processing-in-memory (PIM) architectures are developed to bring computation logic in or near memory to alleviate the bandwidth limitations during data transmission. NAND-like spintronics memory (NAND-SPIN) is one kind of promising magnetoresistive random-access memory (MRAM) with low write energy and high integration density, and it can be employed to perform efficient in-memory computation operations. In this work, we propose a NAND-SPIN-based PIM architecture for efficient convolutional neural network (CNN) acceleration. A straightforward data mapping scheme is exploited to improve the parallelism while reducing data movements. Benefiting from the excellent characteristics of NAND-SPIN and in-memory processing architecture, experimental results show that the proposed approach can achieve ∼2.6× speedup and ∼1.4× improvement in energy efficiency over state-of-the-art PIM solutions.

Keywords Processing-in-memory, Convolutional neural network, NAND-like spintronics memory, Nonvolatile memory, Magnetic tunnel junction

Citation Yinglin ZHAO, Jianlei YANG, Bing LI, et al. A NAND-SPIN-Based Processing-in-MRAM Architecture for Convolutional Neural Network Acceleration. Sci China Inf Sci, for review

1 Introduction

Over the past decades, the volume of data required to be processed has been dramatically increasing [1]. As the conventional von Neumann architecture separates processing and data storage components, the memory/computational resources and their communication are in the face of limitations due to the long memory access latency and huge leakage power consumption. This phenomenon can be interpreted as memory and power walls [2]. Therefore, there is an urgent need to innovate the architecture and establish an energy-efficient and high-performance computing platform to break existing walls.

Processing-in-memory (PIM), a promising architecture diagram, has been proposed to overcome power and memory walls in recent years [3, 4]. Through the placement of logic units in the memory, the PIM architecture is considered an efficient computing platform because it performs logic operations by leveraging inherent data-processing parallelism and high internal bandwidth [5, 6]. However, the full exploitation of the bandwidth and the integration of computing cells within the memory result in a major circuit redesign and a significant chip area increase [7]. As CMOS technology is moving to its physical limitation [8], the realization of PIM generates increases design and manufacturing costs and sacrificed memory capacity to some extent, which is not conducive to obtaining cost-effective products.

[*] Corresponding author (email: jianlei@buaa.edu.cn, bing.li@cnu.edu.cn)

https://engine.scichina.com/doi/10.1007/s11432-021-3472-9

In recent years, many non-volatile memories (NVMs), such as resistive random-access memory (ReRAM) [9–11], phase change memory (PCM) [12, 13], and magnetoresistive random-access memory (MRAM) [14, 15], provide PIM with a new research platform. Among all emerging NVM technologies, MRAM has emerged as a promising high-performance candidate for the main memory due to its non-volatility, superior endurance, zero standby leakage, compatibility with the CMOS fabrication process and high integration density [16]. In particular, spin-transfer torque MRAM (STT-MRAM) and spin-orbit torque MRAM (SOT-MRAM) are two advanced types of MRAM devices [17]. However, the switching speed and energy consumption of STT-MRAM are limited by the intrinsic incubation delay, while SOT-MRAM exhibits a poor integration density because it contains two transistors in a standard bit cell [18]. In [19, 20], an emerging spintronics-based magnetic memory, NAND-like spintronics memory (NAND-SPIN), was designed to overcome the shortcomings of STT-MRAM and SOT-MRAM and pave a new way to build a novel memory and PIM architecture.

Convolutional neural networks (CNNs) have received worldwide attention due to their potential of providing optimal solutions in various applications, including popular image recognition and language processing [21]. As neural networks deepen, the high-performance computation of CNNs requires a high memory bandwidth, large memory capacity, and fast access speed, which are becoming harder to achieve in traditional architectures. Inspired by the high performance and impressive efficiency of PIM, researchers have attempted to implement in-memory CNN accelerators. For example, CMP-PIM involves a redesign of peripheral circuits to perform CNN acceleration in the SOT-MRAM-based memory [22]. STT-CiM [16] enables multiple word lines within an array to realize in-memory bit-line addition through the integration of logic units in sense amplifiers. However, their performance improvement brought about by PIM is offset by the shortcomings of the SOT/STT-MRAM mentioned above.

NAND-SPIN adopts a novel design that allocates one transistor for each magnetic tunnel junction (MTJ) and writes data with a small current, which means low write energy and high integration density. Despite its excellent potential, the PIM architecture based on NAND-SPIN is still scarce. In this study, we developed an energy-efficient memory architecture based on NAND-SPIN that can simultaneously work as an NVM and a high-performance CNN accelerator. The main contributions of this study are summarized as follows:

- Inspired by the outstanding features of NAND-SPIN devices, we developed a memory architecture based on NAND-SPIN. Through the modification of peripheral circuits, the memory subarray can perform basic convolution, addition and comparison operations in parallel.
- By breaking CNN inference tasks into basic operations, the proposed NAND-SPIN-based PIM architecture achieves a high-performance CNN accelerator, which has the advantages of in-memory data movement and excellent access characteristics of NAND-SPIN.
- We employed a straightforward data mapping scheme to fully exploit data locality and reduce data movements, thereby further improving the performance and energy efficiency of the accelerator.
- Through bottom-up evaluations, we show the performance and efficiency of our design with comparison to state-of-the-art in-memory CNN accelerators.

The remainder of this paper is organized as follows: Section 2 presents the background of MRAM and design motivation. Section 3 provides the details of the proposed architecture. Section 4 presents the acceleration methods for CNNs and introduces some optimization schemes. Section 5 describes the experimental platform and analyzes the simulation results. Section 6 concludes this paper.

2 Preliminary and Motivation

In this section, we present MRAM-related technologies, CNNs and existing in-memory computing designs.

2.1 MRAM

MTJs are the basic storage element in STT-MRAM and SOT-MRAM [17, 23]. As shown in Fig. 1a, an MTJ contains three layers: two ferromagnetic layers with a tunnel barrier sandwiched between them. The magnetization direction of the pinned layer is fixed and perpendicular to the substrate surface, while the magnetization direction of the free layer exhibits two stable states: parallel (P) or anti-parallel (AP) to that of the pinned layer. Due to the tunnel magnetoresistance (TMR) effect, when the magnetization directions of the two ferromagnetic layers are parallel (anti-parallel), the resistance of the MTJ is low (high). This state is used to represent the logic “0” (“1”) [24].

The most popular STT-RAM cell structure is illustrated in Fig. 1b [25]. The MTJ pillar has a small area and can be integrated above transistors. Hence, the total cell area is determined only by the bottom transistors and leads to an expectation of achieving a high-density memory. However, the long write latency and high write energy hinder the broad application of STT-MRAM.

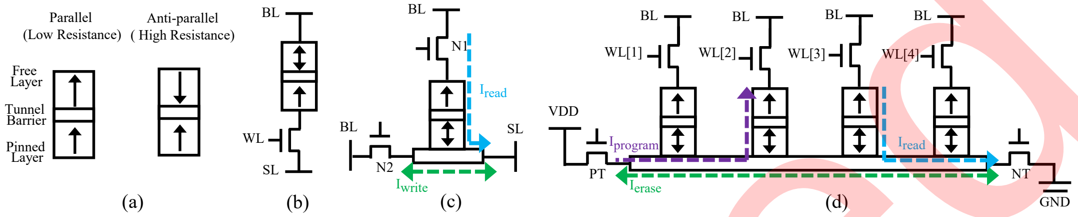


Figure 1 (a) Device structure of the MTJ in parallel and anti-parallel states. (b) 1T-1MTJ STT-MRAM cell. (c) Bit cell schematic of the standard 2-transistor SOT-MRAM. (d) Structure and operations of the NAND-like spintronic memory.

SOT-MRAM is a composite device of spin hall metal and MTJ [14], and Fig. 1c shows the basic bit cell of a standard SOT-MRAM. The access transistors, N1 and N2, connect the pinned layer of the MTJ and heavy metal strip with bit lines (BLs), respectively. The data can be written into and read out from the MTJ by referring to the green and blue currents flowing from the source lines (SLs) to BLs [26]. Although SOT brings the fast switching of magnetization, such a design faces the storage density challenge because it contains two transistors in a unit.

A multi-bit NAND-SPIN device is shown in Fig. 1d, in which the MTJs are organized similar to a NAND flash memory [19, 27]. The PMOS transistor (PT) and NMOS transistor (NT) work as the selection transistors for conducting paths to the VDD and GND, respectively. In the NAND-SPIN, the write operation requires two steps:

Step 1: Erase data in all MTJs, and initialize them into default AP states. In this step, two transistors, PT and NT, are activated, while all word line (WL) transistors are off. The generated current between VDD and GND can erase all MTJs in the heavy metal strip via the SOT mechanism.

Step 2: Program the selected MTJs by switching them into the P state. In this step, the corresponding WL and PT transistors are activated, and the currents flowing through the MTJs from free layers to pinned layers would switch the states of the MTJs to the P state via the STT mechanism.

Because NAND-SPIN uses MTJs as the basic storage element, it has high endurance, which is essential for memory cells. In addition, the compatibility with CMOS makes NAND-SPIN a high density memory, because it distributes MTJs over CMOS circuits. Compared with conventional STT-MRAM, NAND-SPIN only requires a small STT current to complete an efficient AP-to-P switching. The asymmetric writing scheme reduces the average latency and energy of write operations while achieving a high storage density, which unlocks the potential of MRAM-based architectures.

2.2 CNN

A CNN is a type of deep neural network, commonly used for image classification and object recognition. Typically, a CNN consists of three main types of layers, namely, convolutional layer, pooling layer and fully-connected layer [6, 28, 29].

In the convolutional layer, the kernels extract features from the input feature maps through convolution operations. The convolution operation applies a kernel to move across the input feature map and performs dot products between the inputs and weights. There are usually many input and output feature maps in a convolutional layer, which requires considerable convolution operations.

The pooling layer is used to reduce the input dimensions of the feature maps. Similar to the convolutional layer, the pooling operation slides a filter across the inputs and combines the neuron clusters into a single neuron. There are two types of pooling layers, namely max/min pooling and average pooling. Max/min pooling uses the maximum/minimum value of each cluster as the neuron of the next layer, while average pooling uses the average value.

The fully-connected layer connects all neurons from one layer to every activation neuron of the next layer, and it usually leverages a softmax activation function to classify inputs as the final outputs. Past studies have concluded that the fully-connected layer can be treated as another convolutional layer [30,31].

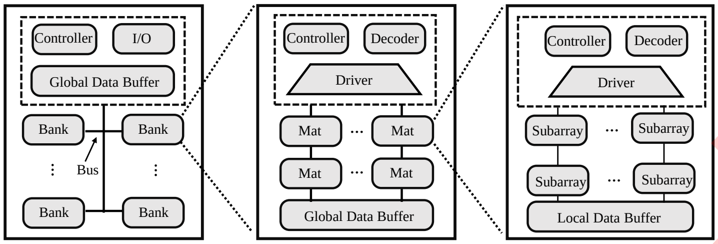


Figure 2 Hierarchical memory organization in the proposed architecture.

2.3 PIM Architectures

To reduce the cost of data movement, the PIM platform has been proposed for several decades [32–34]. Some proposals in the context of static RAM (SRAM) or dynamic RAM (DRAM) have been researched in recent years. For example, in [35], a grid of SRAM-based processing elements was utilized to perform matrix-vector multiplication in parallel. The design in [36] uses a CNN accelerator built with DRAM technology to provide a powerful computing capability and large memory capacity. However, their working mechanisms inevitably lead to multi-cycle logic operations and high leakage power.

Considering the possibility of using NVM as a substitute for the main memory, various works have been conducted to explore emerging PIM architectures. These works put forward a wide range of specialized operators based on NVM for various applications [37,38]. For instance, in [39], an interesting design was proposed to implement in-memory logic based on MTJs. Pinatubo optimized the read circuitry to perform bitwise operations in data-intensive applications [40]. Based on PCM, a equivalent-accuracy accelerator for neural network training is achieved in [13]. In addition, some designs modify memory peripherals to perform specific applications instead of general applications. In [41], a ReRAM crossbar-based accelerator was proposed for the binary CNN forward process. Moreover, PRIME shows a ReRAM-based PIM architecture in which a portion of a memory array can be configured as NN accelerators [42].

Although PIM-based designs effectively reduce data movements, the complex multi-cycle operations and insufficient data reuse are still hindrances to performance improvement. Different from previous designs, we not only used NAND-SPIN to build an in-memory processing platform, but optimized the storage scheme to minimize data duplication and provide large parallelism for in-memory processing.

3 Proposed Architecture

In this section, we first introduce the architecture design and the function of each component. Then, we show how to perform memory and logic functions based on the proposed architecture.

3.1 Architecture

The general memory organization is shown in Fig. 2. There are three levels in such a hierarchical organization: the bank, mat and subarray. The bank is a fully-functional memory unit and banks within the same chip share the I/O resources. The mat is the building block of bank, and multiple mats are connected with a global data buffer. The subarray is the elementary structure in our design, and multiple subarrays in a mat implement memory access or CNN acceleration in parallel. To coordinate those components, the controller generates control signals to schedule computations and communications. In particular, the local data buffer temporarily hold data sent from subarrays and the global buffer for alleviating data congestion. The mat level scheme and peripheral components is shown in Fig. 3a, and the subarray architecture based on NAND-SPIN is illustrated in Fig. 3b. Here, we mark a single NAND-SPIN device containing a group of 8 MTJs with a green ellipse. The specific structure of subarrays and the operation details of CNN acceleration are discussed later.

3.2 Microarchitecture

Fig. 4a describes the detailed structure and internal circuits of a block. The synergy of control signals carries out 3 logic functions: writing, reading and logic AND (for CNN acceleration mode). The writing process is divided into two stages: the stripe erase stage and the program stage. As illustrated in Section 2.1, the WE and ER are both activated in the erase stage to generate the SOT current, while the WE,

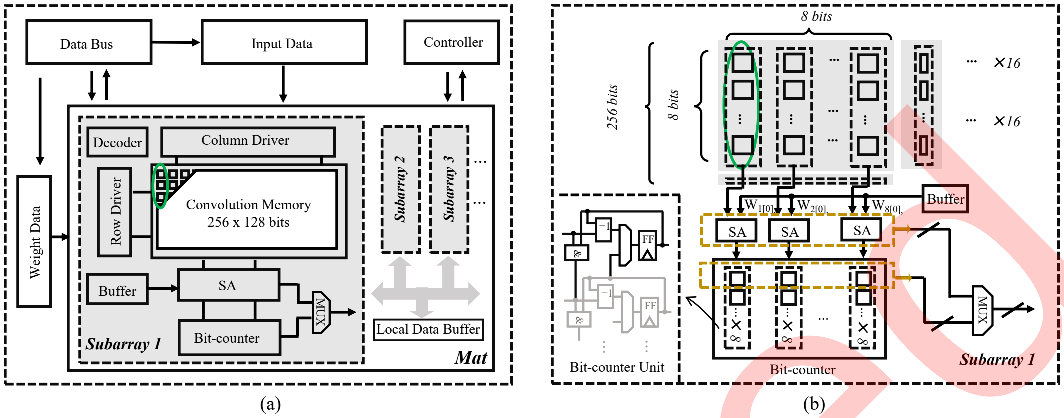


Figure 3 (a) Mat level scheme and peripheral components. (b) NAND-SPIN-based subarray architecture.

$C_x(x=[1,m])$ and corresponding $R_y(y=[1,n])$ are selected in the later program stage to produce the STT current. In regard to read operations, the REF, FU and corresponding $R_y(y=[1,n])$ are set high. Then, the SA is connected to the circuit for a reading operation. Besides, the setting for AND operations is similar to read operations, but the FU varies with the operand.

The SA is the central functional unit that performs read operations and AND operations, utilizing a separated PCSA (SPCSA) circuit (depicted in Fig. 4b) [43]. The SPCSA can sense the resistance difference between two discharge branches according to the discharge speed at two points (V_{ref} and V_{path}). Accordingly, R_{ref} refers to the resistance in the reference path, and is set to $(R_H+R_L)/2$ (R_H and R_L represent the resistance of an MTJ in AP and P states, respectively), and R_{path} represents the total resistance in another path.

An SA requires two steps to implement a single function. The first step is to charge V_{ref} and V_{path} by setting the RE low voltage. The second step is a reverse process that flips RE to discharge V_{ref} and V_{path}. The inverter connected to the point with a higher path resistance first flips and latches the state.

Note that we use a complementary method for data storage. For example, the MTJ in the AP state actually means storing binary data "0". Fig. 4c lists the possible conditions (DATA represents the actual binary data stored in MTJ1) and the outputs of the SA. Moreover, the transistor connected to the REF is turned on by default when the SA is working.

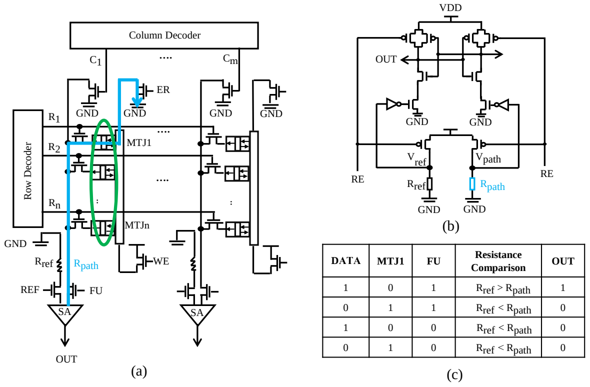
1). Memory Mode: Based on the subarray design described above, Fig. 5 and Table 1 describe the paths of the current flow and corresponding signal states respectively.

Erase operation: To erase the contents in a group of MTJs, the current is generated flowing through the heavy metal strip. As shown in Fig. 5a, the transistors in contact with heavy metal strips are activated by ER and WE, while the other transistors remain deactivated. Then, a path is formed between VDD and GND, and it generates the write current in the heavy metal strip to erase the MTJs to AP states.

Program operation: The program operation is the second step of data writing after the erase operation. A program operation requires a current from the free layer to the fixed layer in the MTJ, as shown in Fig. 5b. While programming data (represented as D in Table 1), the circuit should activate the transistor controlled by WE and the two transistors corresponding to a certain MTJ (for example, R_1 and C_1 for MTJ1 in Fig. 5b). Then, a path is formed between VDD and GND, which produces a current inducing the STT to switch the MTJ from AP to P.

Note that the state of an MTJ after finishing the two stages above is determined by the signals sent from decoders. The signals (R_1 to R_n) determine which row performs the program operation. The signals (C_1 to C_m) produced by the column decoder determine whether the program operation is completed. Noticing the mapping relationship above, we regard generated signals as a map to values that need to be written into MTJs. The signal $C_x(x=[1,m])$ equal to "1" results in a successful program operation as well as the AP-to-P switching in the MTJ. In contrast, the logic "0" in $C_x(x=[1,m])$ means a blocking current in the transistor connected with $C_x(x=[1,m])$, and the MTJ maintains the AP state. Fig. 6 demonstrates the timing diagram of an erase operation followed by a program operation.

Read operation: When performing a typical read operation, a current should be generated in the path connecting the SA and a certain MTJ, as shown in Fig. 5c. Similar to the program operation, the signals (R_1 to R_n) transmitted by row decoders decide which row of MTJs would be read out. Additionally,



DATA	MTJ1	FU	Resistance Comparison	OUT
1	0	1	$R_{ref} > R_{path}$	1
0	1	1	$R_{ref} < R_{path}$	0
1	0	0	$R_{ref} < R_{path}$	0
0	1	0	$R_{ref} < R_{path}$	0

Figure 4 (a) Schematic of the subarray architecture. (b) Schematic of the sensing circuit. (c) Possible conditions and outputs of the SA.

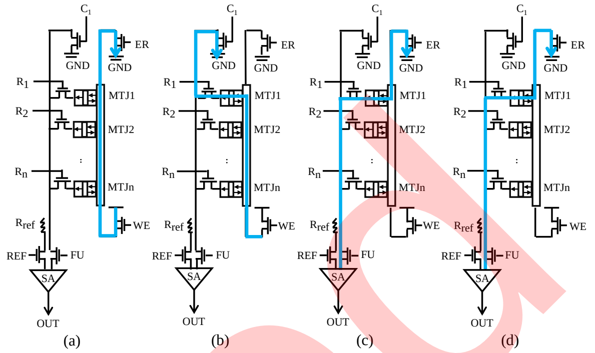


Figure 5 The current paths for (a) erase operation, (b) program operation, (c) read operation, (d) AND operation.

Table 1 Circuit signals for different operations

Operation	WE	ER	C_1	R_1	FU	REF	MTJ[1)]	MTJ[2)]	OUT
Erase	1	1	0	0	0	0	/	1	/
Program D	1	0	D	1	0	0	1	$\overline{D}$	/
Read	0	1	0	1	1	1	$\overline{D}$	$\overline{D}$	D
AND	0	1	0	1	W	1	$\overline{D}$	$\overline{D}$	W ‘AND’ D

ER, FU and REF need to be set to logic “1” during read operations, and then the states of MTJs can be indicated by outputs of SAs. An output “0” indicates that the MTJ has a high resistance (AP state) and stores logic “0”. Conversely, an output “1” refers to an MTJ storing “1” in the P state.

As our subarray structure is different from traditional architectures, the memory access scheme needs to be modified accordingly. In our design, the erase operation can reset a group of MTJs in a single NAND-SPIN device and is always followed by a set of program operations for writing data. However, a read operation does not involve other operations, which causes asymmetry in the read and write operations. In other words, the subarray writes a row of NAND-SPIN devices with an erase operation and N program operations ($M \times N$ bits in total, where M is the number of columns, N is the number of MTJs in a NAND-SPIN device, and $M \times N$ is 128×8 in our design) instead of writing a row of MTJs with a single write operation like the traditional architecture [31]. Nevertheless, the read operation reads a row of data out (128 bits in our design) at a time, the same as the traditional architecture.

Due to the introduction of an erase operation before program operations, the write operation latency would be increased. However, the SOT-induced erase operation could reset multiple MTJs on the same heavy metal strip, while the program operations set MTJs individually. Therefore, the time consumed by a erase operation is amortized. In addition, the SOT-induced erase operation is much faster than the program operation induced by STT, which further offsets the extra latency.

It should be noticed that the read disturb could be significantly mitigated in our design. As the P-to-AP switching is induced by SOT and the AP-to-P switching is based on STT, the read disturb margin is related to the read current and the P-to-AP STT switching current. Therefore, we can increase the P-to-AP STT switching current of MTJs by adjusting the HM dimension to mitigate read disturb issues and enhance the reliability.

2). CNN Acceleration Mode: In CNN acceleration mode, the AND logic is activated in SAs. As shown in Fig. 5d, the AND operation has the same current path as the read operation, and the difference between them lies in FU. FU is always at a high voltage during a read operation, while FU is used to represent one of the two source operands (represented as W in Table 1) during an AND operation. Another source operand is supposed to have been stored in the selected MTJ, and the SA finally obtains the AND operation result. Only when the MTJ is in a low resistance state (storing “1”), FU is under high voltage (indicating logic “1”), and the resistance of R_{path} is smaller than R_{ref}, the SA outputs “1”. Other situations result in R_{path} being larger than R_{ref}, and the SA outputs “0”. Fig. 7 demonstrates

[1)] The MTJ state before the operation
[2)] The MTJ state after the operation

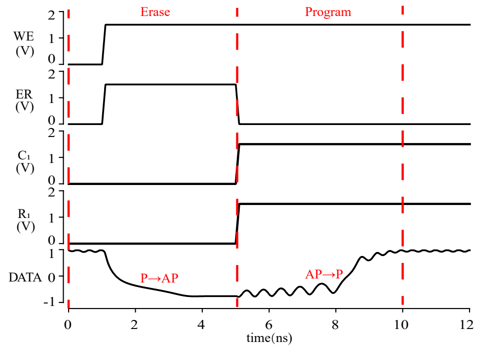


Figure 6 Timing diagram of erase and program operations.

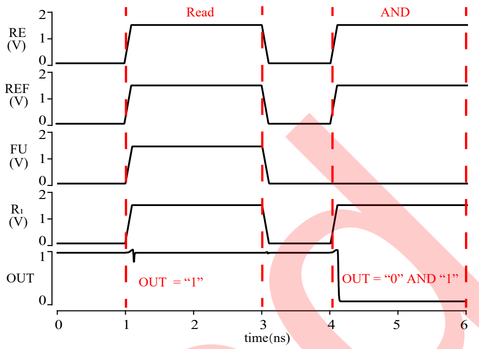


Figure 7 Timing diagram of read and AND operations.

the timing diagram of a read operation and an AND operation, assuming that D = "1" and W = "0".

While accelerating CNN inferences, data buses are used for transmission of weight and input data, both of which are considered as collections of source operands (especially for AND operations). The weight and input data need to be transferred into the buffers and convolution memories (CMs) before the CNN computation starts. The buffer is used for storing temporary weight data to reduce data movements and bus occupation. Moreover, the buffer is connected to the data bus through private data ports so that it does not occupy the bandwidth of the subarray. The bit-counter in each column could count the non-zero values of all AND operation results obtained in the corresponding SA. The multiplexers are used to output the data sensed in SAs during normal read operations or the data in the bit-counters (bit-by-bit for each unit) during convolution operations, as shown in Fig. 3.

4 Implementation

This section first introduces the complex computing primitives in CNN computation, and then shows how our architecture performs an inference task. As introduced above, the convolutional layer involves considerable convolution operations, and the pooling layer performs iterative addition, multiplication and comparison operations to implement average pooling or max/min pooling. Since AND is a universal logic gate, we use it to implement computing primitives together with bit-counters.

4.1 Building Blocks of CNN

Convolution: Convolution is the core operation of CNN, and it takes up most fraction of computation resources. We consider I (W) as an input (weight) fixed-point integer sequence located in an input (kernel) map [30]. Assuming that $I = \sum_{n=0}^{N-1} c_n(I)2^n$ and $W = \sum_{m=0}^{M-1} c_m(W)2^m$ where $(c_n(I))_{n=0}^{N-1}$ and $(c_m(W))_{m=0}^{M-1}$ are bit vectors, the dot product of I and W can be specified in Eq. 1.

$$I * W = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} 2^{n+m} bitcount(AND(c_n(I), c_m(W))). \tag{1}$$

Regarding the computationally expensive convolution operation as a combination of rapid and parallel logic AND, bit-count and shift operations, the PIM architecture commonly converts it into consecutive bitwise operations. Previously, some schemes first store the weight and input data in the same column, and then sense the bitwise operation outputs in modified circuits [16, 31]. However, those methods require additional data duplication and reorganization while the weight matrix slides, which aggravate the overhead as the time-consuming and power-consuming characteristics of the NVM.

To address this issue, we adopt a straightforward data storage scheme to reduce redundant access operations. We split both the input and weight data into 1-bit data. For example, an M-bit input matrix is converted to M 1-bit matrices and stored in M subarrays, and an N-bit weight matrix is decomposed into N 1-bit matrices and transmitted to each subarray for bitwise convolution. Fig. 8 illustrates the bitwise convolution of a 2×2 weight matrix and a 2×5 input matrix. In the first step, the first row of the input matrix in CM is activated, and the first row of the weight matrix in the buffer is connected to SAs in parallel for AND operations. The results are transferred to and counted in the bit-counter unit of each

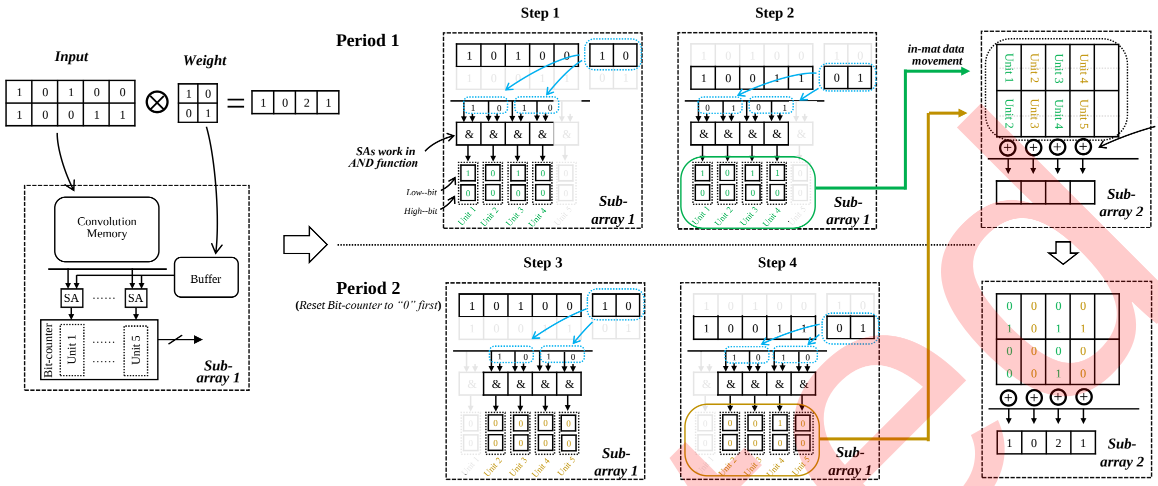


Figure 8 Bitwise convolution operation.

column. By repeating the above processes for the second row of matrices, the second step obtains the counting results in bit-counter units. Those units transfer their contents to Subarray 2 through in-mat data movement, and they would be reset to zero at the end of first period. The second period slides the weight matrix to the next position to work out another set of bit-counting results. Finally, Subarray 2 perform in-memory addition (will be discussed later) to get the bitwise convolution results.

Note that our design improves parallelism by greatly reusing the weights instead of duplicating the inputs in subarrays. In addition, the introduction of the buffer reduces the overhead of in-memory data movement. Requiring only one writing operation into the buffer, the 1-bit weight matrix would be used during the bitwise convolution operations of the entire 1-bit input matrix in this subarray, which significantly reduces data movements and dependence on the data bus. Since the buffer only needs to hold one bit of each weight matrix element, it does not require much capacity.

Addition: Unlike convolution, addition employs a data allocation mechanism that stores data element-by-element vertically [6]. Before addition starts, all bits of the data elements are transposed and stored in the CM. One type of conventional design paradigm generally selects two rows of data simultaneously and performs addition operation using a modified sense amplifier. However, the process variation may cause logic failures, making it hard to guarantee reliability. Our design uses bit-counters to count the non-zero data in each bit-position from the least significant bit (LSB) to the most significant bit (MSB). Fig. 9 shows the data organization and addition work steps of two vectors (vector A and B, both are 2-bit numbers). The numbers in circles indicate the execution order of the involved operations in each step. The two vectors that are going to be added together are put in the same column of the CM. There are 3 empty rows reserved for the sum results. In each step, the bits of the two vectors at the same bit-position are read out by read WLs (RWL) and bit-countered (BC) in bit-counter units. The LSBs of the count results are written back through a write WL (WWL), and the other bits of the count results are right-shifted as the initial state of the next step. As demonstrated in Fig. 9, the LSBs of the count results generated in the second and third steps are stored back as the second and third bits of the sum results. Moreover, the addition operation can be extended to the case where multiple source operands are added, as long as these operands are in the same column.

Multiplication: Multiplication has a data allocation mechanism similar to addition. The difference between them lies in that the AND function is activated in SAs to generate bit multiplication results. We show how multiplication works using an example of a 2-bit multiplication in Fig. 10. The multiplication starts with initializing all bits of two vectors (A and B) in the CM and the buffer, and there are 4 empty rows reserved for the product results. The multiplication algorithm generates the product results bit-by-bit from the LSB to the MSB. In each step, each bit of the product is produced by bit-counting all the single-bit products that corresponding to this bit-position. For example, since the LSBs of the products are determined by the bit multiplication results of the LSBs of two vectors (A and B), the LSBs of two vectors A and B are read out simultaneously to perform bit multiplication in the first step. Considering two bits read out as operands, the SAs perform parallel AND operations and transfer the results to

https://engine.scichina.com/doi/10.1007/s11432-021-3472-9

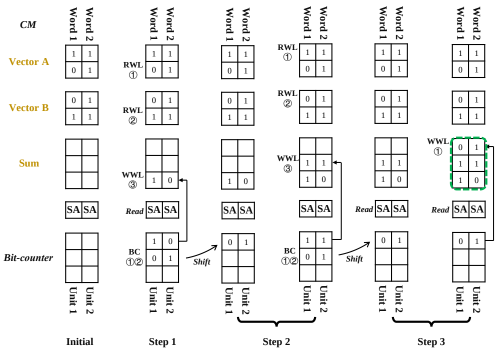


Figure 9 Computation steps of the addition operation.

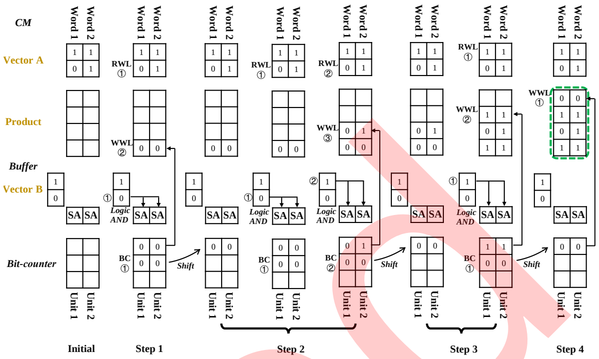


Figure 10 Computation steps of the multiplication operation.

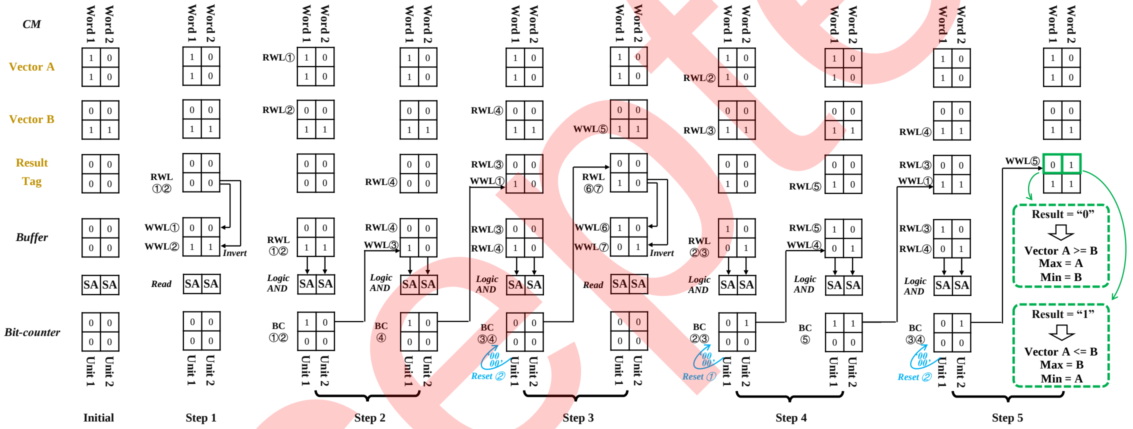


Figure 11 Execution steps of the comparison operation.

bit-counter units for counting. Then, the LSBs of those units report the LSBs of the product and are stored back in CM (product part) accordingly by a WWL operation. The other bits of the count results, which record the carry-in information, are right-shifted as the initial state of the next step. Obviously, the second step requires more cycles to count two partial AND operation results than the first step. It should be noted that the buffer capacity is limited, so it is not wise to set a different multiplier for the multiplicand in each column. Therefore, our architecture is suitable for multiplicative scaling with the same scale factor.

Comparison: Max/Min extraction is a common operation in the max/min pooling layer. We demonstrate how to compare two sets of data (vector A and B) and select the max/min using the method shown in Fig. 3. Initially, two vectors are stored bit-by-bit in the vertical direction along the BL. In addition, two extra rows of storage (Result and Tag) are both reset to 0, where Result row indicates the comparison results and Tag row is used as identifiers. In the first step, the row of Tag is read out by an RWL, and then two WWLs are activated to write the Tag row and its inverted values into the buffer. The second step activates two RWLs to read out the MSBs of the two vectors (A and B) on the same BL, and the SAs simultaneously perform AND operations considering the second row of the buffer as another operand. The outputs of SAs are subsequently bit-counted in the bit-counter. Then the LSB of each unit indicates the comparison result of two vectors. The LSB of the unit equaling 1 means that the two bits read out are different. Subsequently, we write the LSBs into the second row of the buffer and update the bit-counter with the 'AND' operation results between the first row of the buffer and the Tag row. Next, the LSBs of bit-count units are written into the Tag row, and all bit-counter units are reset to zero. In step 3, as shown in Fig. 3, two more AND operations are performed, where the MSBs (vector B), the Result row and the buffer are considered as operands. So far, the LSBs of bit-count units represent the comparison results only considering the first bit of each vector. We store the results in the Result row and start the next bit comparison process. The data in the Result and Tag rows are gradually updated as each bit is compared from MSB to LSB. If the final data located in the Result row is 1, vector A is

https://engine.scichina.com/doi/10.1007/s11432-021-3472-9

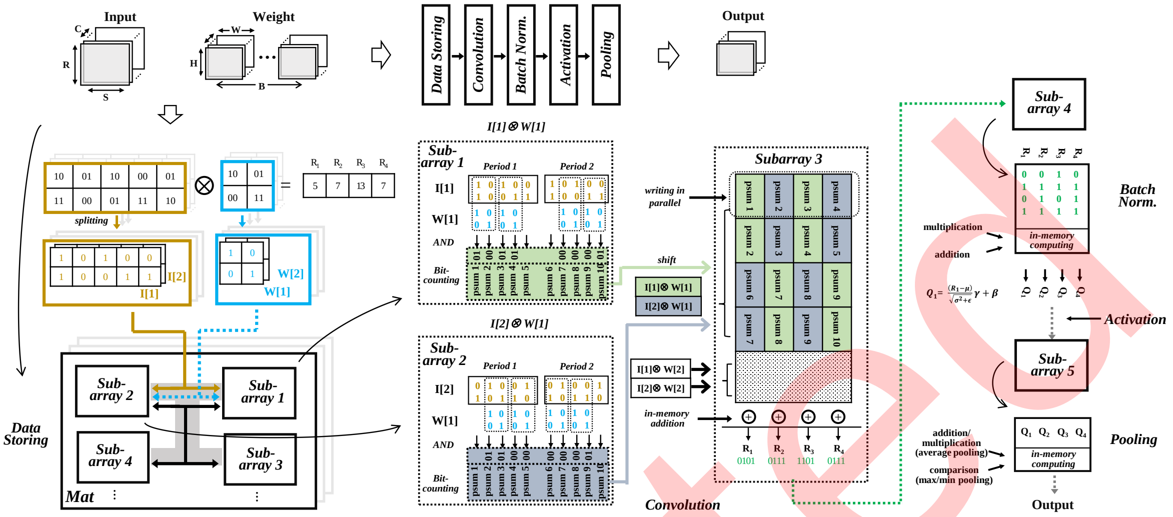


Figure 12 Data organization and computation steps of CNN.

greater than or equals to vector B, and A/B stands for the max/min of the two. Conversely, the binary data 0 means that B/A is the max/min.

4.2 CNN Inference Accelerator

In realistic scenarios of mainstream CNNs, it is hard to store all the data of one layer in a limited-capacity PIM platform. Therefore, reducing data duplication enables the memory array to accommodate more data. Fig. 12 shows the data organization and computation steps of CNNs. Initially, the input matrix is split and organized in different subarrays in a mat. To perform CNN inference tasks, the weight matrix is decomposed and transferred into multiple subarrays for parallel bitwise convolution. Although there is still massive necessary data movements, our design tends to exploit the internal data buses, which can reduce the dependence on the external buses. The operations of each layer are described below.

Convolutional layer: In this layer, the subarrays are configured to generate partial-sums through bitwise convolution operations. The partial-sums are summed, and then sent to the activation function. To maximize parallelism, we adopt a cross-writing scheme during convolution operations. This scheme guarantees that the bit-counting results produced by different subarrays during the same period are not crossed. For example, as shown in Fig. 12, during the Period 1, Subarray 1 and 2 obtain the bit-counting results, which are not crossed and therefore could be written into different columns of the Subarray 3. Thus, the partial-sums are written in parallel without cache operations. In addition, since the bit-counting results are read out bit-by-bit from LSBs to MSBs, the shift operation can be realized by simply writing them to different rows in the vertical direction in Subarray 3.

In CNN, calculations with high-precision numerical values require significant computational power and storage resources. Quantization is the transformation process of lessening the number of bits needed to represent information, and it is typically adopted to reduce the amount of computation and bandwidth requirement without incurring a significant loss of accuracy. Several works have shown that the quantization to 8-bit can achieve comparable prediction accuracy as 32-bit precision counterparts [30, 44]. In our design, we perform the quantization using the minimum and the maximum values of the given layer. The transformation, which quantizes the input Q_i to a k-bit number output Q_o, is as follows:

$$Q_o = round((Q_i - Q_{min})\frac{(2^k-1)}{Q_{max}-Q_{min}}). \tag{2}$$

Q_{max} and Q_{min} are the minimum and maximum values of the layer in the training phase. Therefore, the part $\frac{(2^k-1)}{Q_{max}-Q_{min}}$ could be calculated in advance, and this formula can be performed through in-memory addition and multiplication in subarrays.

Batch normalization is the following process that can recover the quantization loss and retain the accuracy of the model. The batch normalization transformation makes the data set have zero mean and one standard deviation [45], and given below:

Table 2 Simulation parameters

Spin Hall angle	0.3	Exchange bias	15 mT
Gilbert damping	0.02	TMR	120%
Resistance-area product	$5\ \Omega \cdot \mu m^2$	Tunneling spin polarization	0.62
Saturation magnetization	1150 kA/m	Heavy metal thickness	4 nm
Ratio of damping-like SOT to field-like SOT	0.4	Uniaxial anisotropy constant	$1.16\ \times 10^6 J/m^3$

$$I_o = \frac{I_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta, \tag{3}$$

where I_o and I_i denote the corresponding output and input of the transformation, respectively. σ and μ are two statistics of the training model, γ and β are trained parameters used to restore the representation power of the network, and ϵ is a constant added for numerical stability. The aforementioned parameters are calculated and stored in advance, so that the above formula can be parallel performed through in-memory addition and multiplication in subarrays, similar to quantization. In addition, the ReLU activation function is achieved by replacing any negative number with zero. The MSB of the input is read out first and used to determine whether to write zero.

Pooling layer: Average pooling and max/min pooling are the two main types of pooling layers. Average pooling computes the average of all input values inside a sliding window. We support average pooling by summing the input values in a window and dividing the sum by the window size. Max/min pooling calculates the max/min of all the inputs inside the window and is accomplished by iterative in-memory comparison. In each iteration, the input for the comparison is selectively copied from max/min in the previous iteration.

Fully-connected layer: It has been concluded that the fully-connected layers can be implemented by convolution operations using 1×1 kernels in networks [30,31]. Therefore, we treat the fully-connected layer as convolutional layer.

5 Evaluation

5.1 Platform Configurations

To compare our design with other state-of-the-art solutions, we adopted a device-to-architecture evaluation along with an in-house simulator to evaluate the performance and energy benefits. We first characterized the hybrid circuit using a 45nm CMOS PDK and a compact Verilog-A model that is based on the Landau-Lifshitz-Gilbert equation [19]. Table 2 lists some key device parameters used in our experiments. The circuit level simulation was implemented in Cadence Spectre and SPICE to obtain the performance parameters of basic logic operations. The results showed that it costs 180 fJ to erase an NAND-SPIN device with eight MTJs, with average 0.3 ns for each MTJ, and 840 fJ to program an NAND-SPIN device, with 5 ns for each bit. And the latency and energy consumption were 0.17 ns and 4.0 fJ for a read operation. The bit-counter module was designed based on Verilog HDL to obtain the number of non-zero elements. We synthesised the module with Design Compiler and conducted a post-synthesis simulation based on 45nm PDK. Secondly, we modified NVSim simulator [46], so that it calibrates with our design while performing access and in-memory logic operations. After configuring NVSim based on the previous results, the simulator reported the memory latency, energy and area corresponding to the PIM platform. Finally, for the architecture level simulation, we simulated the CNN inference tasks with an in-house developed C++ code, which simulates the data movement and in-memory computation in each layer.

5.2 Experimental Setup

Both the memory capacity and bandwidth can affect the peak performance of the CNN accelerator. We evaluated these impacts on the basis of fixed memory structure. In our design, we assumed that there are 4×4 subarrays with 256 rows and 128 columns in each mat, and 4×4 mats were considered as a group.

Obviously, enlarging the memory capacity brings a higher performance owing to the increase in the number of computation units. Fig. 13a indicates the relationship between the performance and memory

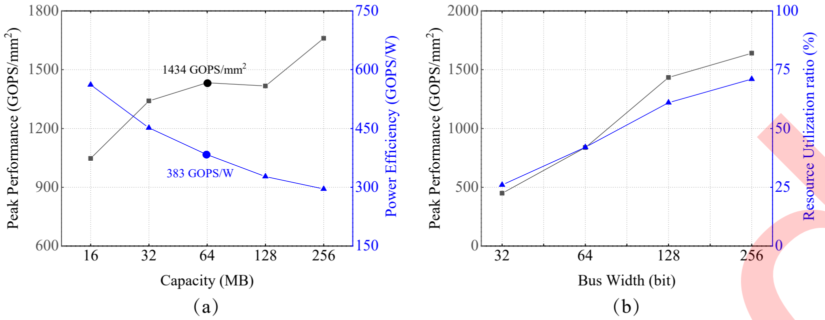


Figure 13 (a) The effect of the capacity on the peak performance and energy efficiency. (b) The effect of the bus width on the peak performance and resource utilization ratios.

capacity. We observed that the peak performance normalized to the area tended to increase slowly with the expansion of the memory capacity, and it reached a regional peak at 64 MB. Nonetheless, the power efficiency dropped because of the increasing energy consumption of peripheral circuits.

Due to the bandwidth limitation, the architecture exhibited a relationship between the performance and the bandwidth as shown in Fig. 13b. In addition, the weight data were transferred to subarrays through the bus and buffered in the buffer. Obviously, the peak performance normalized to the area rose linearly as the bandwidth increases. This mainly arises from that the higher bandwidth provided more data for computation units, which could also be verified from the view of hardware utilization ratios.

With reference to the above results, we configured our PIM architecture with a 64 MB memory array and a 128-bit bandwidth in subsequent simulations.

5.3 CNN Acceleration Performance

For comparison with state-of-the-art CNN accelerators, we regard the designs based on DRAM (DRISA in [36]), ReRAM (PRIME in [42]), STT-RAM (STT-CiM in [16], MRIMA in [31]), and SOT-RAM (IMCE in [21]) as counterparts. Among various benchmarks, we validated the AlexNet/VGG19/ResNet50 models on the ImageNet dataset for a comprehensive evaluation. At runtime, the execution of convolution accelerators depends on the reasonable data flows and the control signals. The inputs and weights of each model were transferred to and initialized in subarrays. The complex logic operations in each layer were decomposed into a series of simple logic operations which were performed sequentially. Temporary results at runtime were transferred to each other across the buses between modules. Considering the uniqueness of those CNN models in depth and structure, the architectures had unique timing control signals to schedule the computations and communications for different models. In addition, the accelerators would split multi-bit data for fine-grained computations, when there was a mismatch between the data matrices and subarrays in size.

Energy efficiency: We obtained the energy efficiency normalized to area results in different bit-width (precision) configurations $\langle W : I \rangle$ in three models. As shown in Fig. 14, our design offered energy efficiency superior to those of the other solutions. In particular, the proposed method achieved 2.3× and 12.3× higher energy efficiency than DRAM- and ReRAM-based accelerators on average, mainly for four reasons: 1) Part of the energy-intensive calculation was converted to efficient AND and bit-count operations. 2) The introduction of the buffer reduced data movements and rewrite operations within the memory, which increased the data reuse while reducing the energy consumption. This also contributed greatly to the superiority of our method to the SOT-based solution ($\sim$2.6× energy savings on average). 3) By exploiting the characteristics of the SOT mechanism and implementing the customized storage scheme, our architecture achieved lower energy consumption for data writing than all counterparts, even STT-CiM ($\sim$1.4× energy savings). 4) The elimination of complex functional units, such as ADCs/DACs in the ReRAM crossbar, also resulted in favorable energy efficiency. Although there were some adders and bit-counters in our design, the scheme in which different significant bits were separately processed dramatically reduces the number of accumulations. This is also why the improvement in the energy efficiency of our design becomes increasingly evident when $\langle W : I \rangle$ increases.

Speedup: The performance of each accelerator in different bit-width (precision) configurations $\langle W : I \rangle$ is presented in Fig. 15. Among all solutions, our design obtained the highest performance normalized to area, with a 6.3× speedup over the DRAM-based solution and an approximately 13.5× speedup over

https://engine.scichina.com/doi/10.1007/s11432-021-3472-9

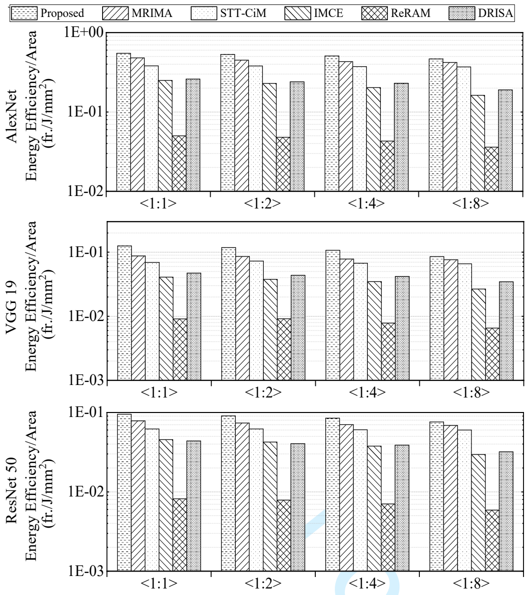


Figure 14 Comparison of the architecture efficiencies for different $\langle W : I\rangle$ ratios across various CNN models.

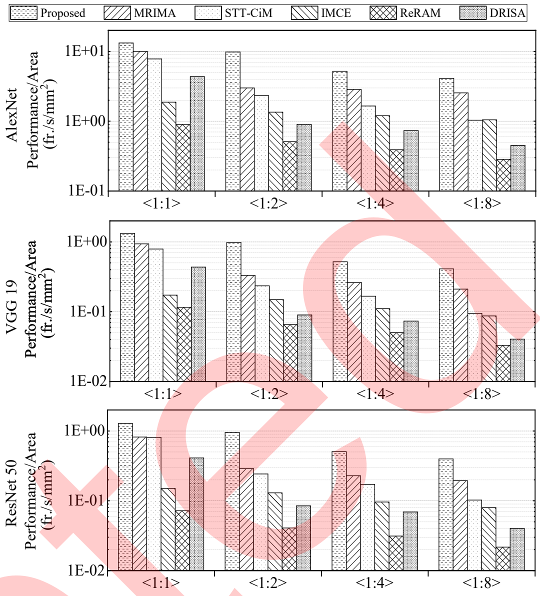


Figure 15 Comparison of the architecture performance for different $\langle W : I\rangle$ ratios across various CNN models.

Table 3 Comparison with related in-memory CNN accelerators

Accelerator	DRISA [36]	PRIME [42]	STT-CiM [16]	MRIME [31]	IMCE [21]	Proposed
Technology	DRAM	ReRAM	STT-RAM	STT-RAM	SOT-RAM	NAND-SPIN
Throughput (FPS)	51.7	9.4	45.6	52.3	21.8	80.6
Capacity (MB)	64	64	64	64	64	64
Area (mm^2)	117.2	78.2	57.7	55.6	128.3	64.5

the ReRAM accelerator. The improvement in our design comes from several aspects: 1) The parallel execution of logic operations and the pipeline mechanism for implementing accumulation fully utilized the hardware resources to complete efficient convolution calculation. 2) The participation of the buffer in PIM effectively reduced the in-memory data movements, data congestion, and bus competition, all of which reduce the waiting time. 3) There were no need for complex peripheral circuits in our design, such as ADCs/DACs in the ReRAM crossbar, which could reduce the area overhead to a certain extent. In addition, the results showed that our design is on average 2.6× and 5.1× faster than the STT-CiM and IMCE, mainly because of the efficient and parallel logic operations.

Table 3 shows the area efficiency comparison of related in-memory CNN accelerators. We observed that STT-CiM and MRIMA show better area efficiency, which mainly comes from the high integration density of STT-MRAM-based memory designs. The SOT-MRAM-based architecture took the largest area, even more than the DRISA solution that embeds complex logic circuits in chips as the result of two transistors in a single cell. The proposed NAND-SPIN accelerator was not the most area-efficient architecture, but it offered the highest throughput by exploiting the data locality and benefiting from excellent characteristics of NAND-SPIN devices in memory arrays.

Energy/Latency breakdown: Fig. 16 shows the latency and energy breakdown of our accelerator for ResNet50 model. In Fig. 16a, we observed that loading data and distributing them into arrays is the most time-consuming part, accounting for 38.4%. This was mainly because writing data into NAND-SPIN device took more time than reading. The time spending on convolution and data transfer took 33.9% and 4.8% of the time respectively. In addition, 13.2% of the time was spent on data comparison operations in the process of determining the maximum in pooling layers. The remaining parts were for batch normalization (4.4%) and quantization (5.3%).

As shown in Fig. 16b, the convolution, corresponding to numerous data reading and bit-counting operations, consumed the most energy up to 35.5%. Due to the high writing energy consumption of NAND-SPIN device, loading data consumed nearly 32.6% of the total energy consumption. Transferring data contributed to 4.9% of the energy consumption, and 15.4% of the energy was spent in pooling layers. The other parts included batch normalization (5.1%) and quantization (6.5%).

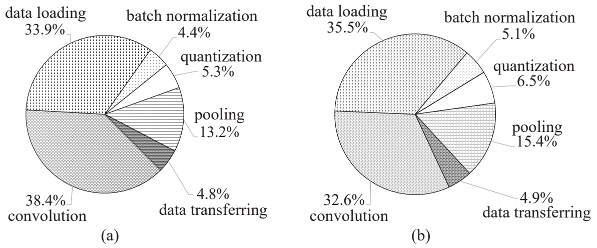


Figure 16 Breakdown of (a) latency and (b) energy.

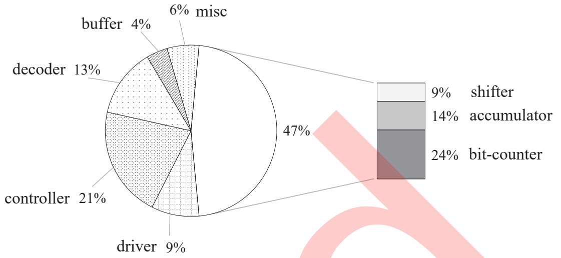


Figure 17 Area overhead breakdown.

Area: Our experiments showed that our design imposes 8.9% area overhead on the memory array. The additional circuits supported the memory to implement in-memory logic operations and cache the temporary data in CNN computation. Fig. 17 shows the breakdown of area overhead resulted from the add-on hardware. We observed that up to 47% area increase was taken by added computation units. In addition, approximately 4% was the cost of the buffer, and other circuits, such as controllers and multiplexers, incurred 21% area overhead.

6 Conclusion

In this paper, we propose a memory architecture that employs NAND-SPIN devices as basic units. Benefiting from the excellent characteristics such as low write energy and high integration density, the NAND-SPIN-based memory achieves a fast access speed and large memory capacity. With supportive peripheral circuits, the memory array can work as either a normal memory or perform CNN computation. In addition, we adopted a straightforward data storage scheme so that the memory array reduces data movements and provides high parallelism for data processing. The proposed design exploits the advantages of PIM and NAND-SPIN to achieve high performance and energy efficiency during CNN inferences. Our simulation results demonstrate that the proposed accelerator can obtain on average $\sim$2.3× and $\sim$1.4× better energy efficiency, and $\sim$6.3× and $\sim$2.6× speedup than the DRAM-based and STT-based solutions, respectively.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China (No. 62072019, 62004011, 62171013), the Joint Funds of the National Natural Science Foundation of China (No. U20A20204), and the State Key Laboratory of Computer Architecture (No. CARCH201917).

References

1 Shafique M, Hafiz R, Javed M U, et al. Adaptive and energy-efficient architectures for machine learning: Challenges, opportunities, and research roadmap. In: Proceedings of IEEE Computer society annual symposium on VLSI, Bochum, 2017. 627–632

2 Luo L, Zhang H, Bai J, et al. SpinLim: Spin orbit torque memory for ternary neural networks based on the logic-in-memory architecture. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2021. 1865–1870

3 Cai H, Guo Y, Liu B, et al. Proposal of analog in-memory computing with magnified tunnel magnetoresistance ratio and universal STT-MRAM cell. 2021. ArXiv: 2110:03937

4 Liu J, Zhao H, Ogleari M A, et al. Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In: Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture, Fukuoka, 2018. 655–668

5 Song L, Zhuo Y, Qian X, et al. GraphR: Accelerating graph processing using ReRAM. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, Vienna, 2018. 531–543

6 Eckert C, Wang X, Wang J, et al. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In: Proceedings of ACM/IEEE 45th Annual International Symposium on Computer Architecture, Los Angeles, 2018. 383–396

7 Hao Y, Xiang S, Han G, et al. Recent progress of integrated circuits and optoelectronic chips. Science China Information Sciences, 2021, 64: 1–33

8 Papandroulidakis G, Serb A, Khiat A, et al. Practical implementation of memristor-based threshold logic gates. IEEE Transactions on Circuits and Systems I: Regular Papers, 2019, 66: 3041–3051

9 Xue C X, Chen W H, Liu J S, et al. 24.1 a 1mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In: Proceedings of IEEE International Solid-State Circuits Conference, San Francisco, 2019. 388–390

10 Li B, Song L, Chen F, et al. ReRAM-based accelerator for deep learning. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition, Dresden, 2018. 815–820

11 Yuan Z, Liu J, Li X, et al. NAS4RRAM: neural network architecture search for inference on RRAM-based accelerators. Science China Information Sciences, 2021, 64: 1–11

12 Kim T, Lee S. Evolution of phase-change memory for the storage-class memory and beyond. IEEE Transactions on Electron Devices, 2020, 67: 1394–1406

13 Ambrogio T, Narayanan P, Tsai H, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. Nature2018, 558: 66–67

14 Guo Z, Yin J, Bai Y, et al. Spintronics for energy-efficient computing: An overview and outlook. Proceedings of the IEEE, Proceedings of the IEEE, 2021, 109: 1398-1417

15 Apalkov D, Dieny B, Slaughter J. Magnetoresistive random access memory. Proceedings of the IEEE, 2016, 104: 1796–1830

16 Jain S, Ranjan A, Roy K, et al. Computing in memory with spin-transfer torque magnetic RAM. IEEE Transactions on Very Large Scale Integration Systems, 2017, 26: 470–483

17 Wang M, Cai W, Zhu D, et al. Field-free switching of a perpendicular magnetic tunnel junction through the interplay of spin-orbit and spin-transfer torques. Nature Electronics, 2018, 1: 582–588

18 Cai W, Shi K, Zhuo Y, et al. Sub-ns field-free switching in perpendicular magnetic tunnel junctions by the interplay of spin transfer and orbit torques. IEEE Electron Device Letters, 2021. 42: 704–707

19 Wang Z, Zhang L, Wang M, et al. High-density NAND-like spin transfer torque memory with spin orbit torque erase operation. IEEE Electron Device Letters, 2018, 39: 343–346

20 Shi K, Cai W, Zhuo Y, et al. Experimental demonstration of NAND-like spin-torque memory unit. IEEE Electron Device Letters, 2021, 42: 513–516

21 Angizi S, He Z, Parveen F, et al. IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference, Jeju, 2018. 111–116

22 Angizi S, He Z, Rakin A S, et al. CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator. In: Proceedings of the 55th Annual Design Automation Conference, San Francisco, 2018. 1–6

23 Cai H, Liu B, Chen J, et al. A survey of in-spin transfer torque mram computing. Science China Information Sciences, 2021, 64: 1–15

24 Fong X, Kim Y, Venkatesan R, et al. Spin-transfer torque memories: Devices, circuits, and systems. Proceedings of the IEEE, 2016, 104: 1449–1488

25 Rho K, Tsuchida K, Kim D, et al. 23.5 a 4Gb LPDDR2 STT-MRAM with compact 9f2 1T1MTJ cell and hierarchical bitline architecture. In: Proceedings of IEEE International Solid-State Circuits Conference, San Francisco, 2017. 396–397

26 Peng S, Zhu D, Li W, et al. Exchange bias switching in an antiferromagnet/ferromagnet bilayer driven by spin–orbit torque. Nature Electronics, 2020. 1–8

27 Yu Z, Wang Y, Zhang Z, et al. Proposal of high density two-bits-cell based NAND-like magnetic random access memory. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, 68: 1665–1669

28 Shafiee A, Nag A, Muralimanohar N, et al. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: Proceedings of ACM/IEEE 43rd International Symposium on Computer Architecture, Seoul, 2016. 14–26

29 Yang J, Fu W, Cheng X, et al. S2Engine: a novel systolic architecture for sparse convolutional neural networks. IEEE Transactions on Computers, 2021

30 Zhou S, Wu Y, Ni Z, et al. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2016. arXiv:1606.06160

31 Angizi S, He Z, Awad A, et al. MRIMA: An MRAM-based in-memory accelerator. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39: 1123–1136

32 Ghose S, Boroumand A, Kim J S, et al. Processing-in-memory: A workload-driven perspective. IBM Journal of Research and Development, 2019, 63: 3:1–3:19

33 Imani M, Gupta S, Kim Y, et al. Floatpim: In-memory acceleration of deep neural network training with high precision. In: Proceedings of ACM/IEEE 46th Annual International Symposium on Computer Architecture, Phoenix, 2019. 802–815

34 Wang X, Yang J, Zhao Y, et al. Triangle counting accelerations: From algorithm to in-memory computing architecture. IEEE Transactions on Computers, 2021

35 Chen Y H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 2017, 52: 127–138

36 Li S, Niu D, Malladi K T, et al. DRISA: A DRAM-based reconfigurable in-situ accelerator. In: 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2017. 288–301

37 Wang X, Yang J, Zhao Y, et al. TCIM: Triangle counting acceleration with processing-in-MRAM architecture. In: Proceedings of the 57th ACM/IEEE Design Automation Conference, San Francisco, 2020. 1–6

38 Yang J, Wang P, Zhang Y, et al. Radiation-induced soft error analysis of STT-MRAM: A device to circuit approach. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 35: 380–393

39 Cai W, Wang M, Cao K, et al. Stateful implication logic based on perpendicular magnetic tunnel junctions. Science China Information Sciences, 2022, 65: 1–7

40 Li S, Xu C, Zou Q, et al. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: Proceedings of the 53rd Annual Design Automation Conference, Austin, 2016. 1–6

41 Tang T, Xia L, Li B, et al. Binary convolutional neural network on RRAM. In: 2017 22nd Asia and South Pacific Design Automation Conference, Tokyo, 2017. 782–787

42 Chi P, Li S, Xu C, et al. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. ACM SIGARCH Computer Architecture News, 2016, 44: 27–39

43 Zhang D, Zeng L, Gao T, et al. Reliability-enhanced separated pre-charge sensing amplifier for hybrid CMOS/MTJ logic circuits. IEEE Transactions on Magnetics, 2017, 53: 1–5

44 Colangelo P, Nasiri N, Nurvitadhi E, et al. Exploration of low numeric precision deep learning inference using Intel FPGAs. In: Proceedings of the 26th Annual International Symposium on Field-Programmable Custom Computing Machines, Boulder, 2018. 73–80

45 Ding P L K, Martin S, Li B. Improving batch normalization with skewness reduction for deep neural networks. In: Proceedings of the 25th International Conference on Pattern Recognition, Milan,2021. 7165–7172

46 Eken E, Song L, Bayram I, et al. NVSim-VX^s: An improved NVSim for variation aware STT-RAM simulation. In: Proceedings of the 53nd ACM/EDAC/IEEE Design Automation Conference, Austin, 2016. 1–6