# A Survey on Memory-centric Computer Architectures

Gebregiorgis, A.B.; Du Nguyen, H.A.; Yu, J.; Bishnoi, R.K.; Taouil, M.; Franky, Catthoor; Hamdioui, S.

**DOI**
10.1145/3544974

**Publication date**
2022

**Document Version**
Final published version

**Published in**
ACM Journal on Emerging Technologies in Computing Systems

# A Survey on Memory-centric Computer Architectures

ANTENEH GEBREGIORGIS, HOANG ANH DU NGUYEN, JINTAO YU,
RAJENDRA BISHNOI, and MOTTAQIALLAH TAOUIL, Delft University of Technology
FRANCKY CATTHOOR, Inter-university Micro-Electronics Center (IMEC)
SAID HAMDIOUI, Delft University of Technology

Faster and cheaper computers have been constantly demanding technological and architectural improvements. However, current technology is suffering from three technology walls: leakage wall, reliability wall, and cost wall. Meanwhile, existing architecture performance is also saturating due to three well-known architecture walls: memory wall, power wall, and instruction-level parallelism (ILP) wall. Hence, a lot of novel technologies and architectures have been introduced and developed intensively. Our previous work has presented a comprehensive classification and broad overview of memory-centric computer architectures. In this article, we aim to discuss the most important classes of memory-centric architectures thoroughly and evaluate their advantages and disadvantages. Moreover, for each class, the article provides a comprehensive survey on memory-centric architectures available in the literature.

CCS Concepts: • **Hardware** → **Memory and dense storage**; *Spintronics and magnetic technologies;*

Additional Key Words and Phrases: Computation-in-memory, computer architectures, resistive computing, classification

**ACM Reference format:**
Anteneh Gebregiorgis, Hoang Anh Du Nguyen, Jintao Yu, Rajendra Bishnoi, Mottaqiallah Taouil, Francky Catthoor, and Said Hamdioui. 2022. A Survey on Memory-centric Computer Architectures. *ACM J. Emerg. Technol. Comput. Syst.* 18, 4, Article 79 (October 2022), 50 pages.
https://doi.org/10.1145/3544974

## 1 INTRODUCTION

For several decades, CMOS down-scaling and architecture improvements have doubled computer performance following Moore's law [28, 80, 90]. However, existing technology suffers from three main walls: leakage wall, reliability wall, and cost wall [75, 189], while computer architectures also face three walls: memory wall, power wall, and **instruction-level parallelism (ILP)** wall [62, 152, 192]. To address these walls, a lot of novel technologies and architectures are under

The work presented in this article is part of the project "Computation-in-memory architecture based on resistive devices" (MNEMOSENE), which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780215.
Authors' addresses: A. Gebregiorgis, H. A. Du Nguyen, J. Yu, R. Bishnoi, M. Taouil, and S. Hamdioui, Delft University of Technology, Mekelweg 4, 2628CD, Delft, The Netherlands; emails: {A.B.Gebregiorgis, H.A.DuNguyen, J.Yu-1, R.K.Bishnoi, M.Taoui, S.Hamdioui}@tudelft.nl; F. Catthoor, Inter-university Micro-Electronics Center (IMEC), Leuven, Belgium; email: Francky.Catthoor@imec.be.

1550-4832/2022/10-ART79 $15.00
https://doi.org/10.1145/3544974

research to improve the performance [89, 123, 141]. As a result, an enormous amount of architectures have been proposed so far. Therefore, a comprehensive survey on those architectures is needed to maintain a systematic view on these architectures.

Since the first Von-Neumann architecture in early 1950s, computer architectures have been evolved to various complex organizations, including pipelined, superscalar, multicore, and so on [69, 80, 96]. Since the energy and performance costs to move data between the memory subsystem and the CPU dominated the total costs of computation, architects and designers are forced to find breakthrough in computers architecture. Memory-centric computing paradigms such as **Processing-in-memory (PIM)** have evolved as promising solution to circumvent the aforementioned challenges [102, 135]. PIM, a concept of integrating memory and processing units, so-called **Logic-in-Memory (LIM)**, was invented in 1970 [178]; however, it was only applied to cache/on-chip memory and was soon abandoned due to the reduced performance during non-local memory accesses [150]. Since 2000, big data and embedded applications have been on the rise and demand new computing systems with not only higher performance but also energy efficiency. To fulfill these requirements, several architectures were explored with the concept of LIM applied for main memories, and regarded as PIM architectures (i.e., FlexRAM [99], DIVA [47], intelligent RAM [105] etc.). However, the PIM architectures were also shortly dismissed due to the limitations of embedded DRAM technology [91, 100, 101]. From the year 2008, emerging non-volatile memory technologies (e.g., memristor) have revived the concept of Processing-In-Memory under the new name In-Memory Computing [43] or Computation-In-Memory [76]. The novel architectures together with new memory technologies promise a lot of potential in terms of area, performance, and energy-efficiency improvements [44, 74, 76, 125, 169–171]. In this regard, there are several works that can be labeled as In-memory computing, such as **Computation-In-Memory (CIM)** [51, 60], ReVAMP [17], Pinatubo [117], and so on. All these architectures have common and distinct features, and these properties were addressed at least partly in the community. This leads to confusion in differentiating these architectures and hence, limiting the exploration potential of novel architectures.

To assess the developments in Computation-In Memory (a.k.a. memory-centric architectures) and determine the innovation potentials, it is essential to classify and evaluate the existing memory-centric architectures. In this regard, our previous work presented a classification methodology and broad overview of memory-centric computing [140]. As follow-up of that earlier work, this article refines the classification metrics and presents a comprehensive set of classification metrics for memory-centric architectures. The metrics are then used to classify the existing memory-centric architectures. Then, for each class, based on the classification metric, a comprehensive survey on existing memory-centric architectures available in the literature is provided. Therefore, the main contribution of this article is a comprehensive survey on existing memory-centric architectures by thoroughly discussing and evaluating the advantages and disadvantages of different memory-centric architectures belonging to the main classes of the classification. Thus, the article contributions are summarized as follows:

- Presenting main characteristics and working principles of existing memory-centric architectures.
- Discussing and evaluating main advantages and disadvantages of each architecture.
- Comprehensive survey and comparison of different existing architectures and provide an outlook on the characteristics of the future architectures.

The rest of this article is structured as follows: Section 2 summarizes the metrics used for the classification, the overview of different classes, namely, **Computation-In Memory Array (CIM-A), Computation-In Memory Periphery (CIM-P)**, and **Computation-Out-of-Memory (COM)**.

Section 3 presents summary of architectures in the CIM-A class. Similarly, Section 4 presents architectures in CIM-P class followed by the COM class architectures in Section 5. Each section describes and evaluates the architectures in their respective classes qualitatively. Section 6 discusses the prospect and challenges of memory-centric architectures. Section 7 concludes the article.

## 2 CLASSIFICATION CRITERIA FOR MEMORY-CENTRIC ARCHITECTURES

Evaluation and classification of modern computing systems is a complex process, as several metrics can be used to classify and evaluate computing systems [165]. Among the various metrics, performance, computing power, and resource utilization-based classification and evaluation of computing systems has been widely used in the literature [85, 165]. However, classification of computing systems based on computation location and computing resource technology, such as memory technology, has not been explored so far due to the universal adoption of Von-Neumann architecture and CMOS technology dominance. Memory is the main storage unit in any computing platform, as shown in Figure 1(a); it can include only memory core with memory arrays and its supporting peripheral circuits, or memory core with extra logic circuits, which is called memory **System-in-Packages (SiP)**. The computations are performed traditionally using computation cores, however, they can also be performed using extra logic circuits, peripheral circuits, and memory array of the memory SiP. With the abundance of Memory-centric architectures, it is becoming increasingly important to define efficient classification and evaluation metrics that can be used to analyze different memory-centric architectures. For this purpose, in our earlier paper, we developed two classification metrics, namely, computation position/location and memory technology. These classification metrics are summarized in the following subsections.

### 2.1 Computation Position

Computation position defines where the result of the computation is produced. A computation includes a primitive logic function (e.g., logical operations) or arithmetic operation (e.g., addition, multiplication). The possibilities of computation position can be seen at the four circles in Figure 1(a). If the result is produced within the memory core (labeled as 1 in Figure 1(a)), then the computer architecture is referred to as **Computation-In-Memory Array (CIM-A)**; if the result is produced within the memory array periphery (labeled as 2 in Figure 1(a)), then the architecture is referred to as **Computation-In-Memory Periphery (CIM-P)**; otherwise, the result is produced outside the memory core (labels 3 and 4 in Figure 1(a)) and the architecture is referred to as **Computation-Out-of-Memory (COM)**.

- **CIM-A:** In CIM-A, the computation result is produced within the memory array (noted as position 1 in Figure 1(a)). Note that this is different from a standard write operation. Typical examples of CIM-A architectures use memristive logic designs such as MAGIC and imply [104, 108]. CIM-A architectures mostly require a modification at the cell-level to support such logic design, as the conventional memory cell dimensions and their embedding in the bit- and wordline structure do not allow them to be used for logic. In addition, modifications in the periphery are sometimes needed to support the changes in the cell. Therefore, CIM-A architectures can be further subdivided into two groups: (1) basic CIM-A, where only changes inside the memory array are required. An example of basic CIM-A is an architecture that performs computations using implication logic [112]; (2) hybrid CIM-A, where, in addition to major changes in the memory array, minimal to medium changes are required in the peripheral circuit. An example of hybrid CIM-A is an architecture that performs computations using MAGIC [108]. In this case, multiple memory rows are written simultaneously; due to the high write currents, modifications are required to the cell and medium changes in the peripheral circuits are needed to activate the multiple rows.

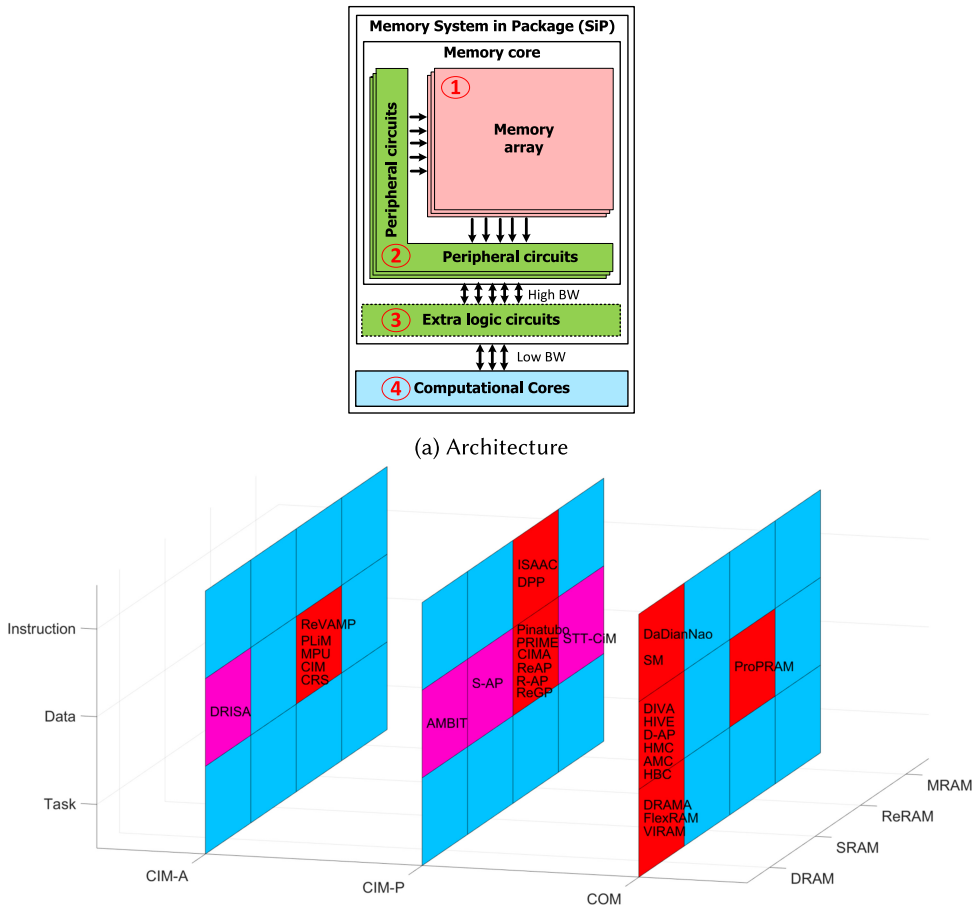



Fig. 1. Computer architecture and proposed classification.

- **CIM-P:** In **CIM Periphery (CIM-P)** architectures, the computation result is produced within the peripheral circuitry (noted as position 2 in Figure 1(a)). Typical examples of CIM-P architectures contain logical operations and vector-matrix multiplications [39, 117]. CIM-P architectures typically contain dedicated peripheral circuits such as DACs and/or ADCs [63, 164] and customized sense amplifiers [117, 203]. Note that more radical changes in the peripheral circuit can be made in the future (e.g., changing in control voltages leads to radical changes in voltage drivers and sense amplifiers, or including a full functional processor inside memory banks). Even though the computational results are produced in the peripheral circuits for CIM-P, the memory array is a substantial component in the computations. As the peripheral circuits are modified, the currents/voltages applied to the memory array are typically different than in the conventional memory. Hence, similar to the CIM-A sub-class, the CIM-P architectures are also further divided into two groups: (1) basic CIM-P, where only change in the peripheral is required, which means the current levels should not be affected. An example of basic CIM-P is Pinatubo logic [117]; (2) hybrid CIM-P, where the majority of the changes take place in the peripheral circuit and minimal to medium changes

in the memory array. An example of hybrid CIM-P is ISAAC [164]. ISAAC activates all rows of a memory array at the same time during read operations to perform a matrix vector multiplication using an ADC readout circuit. This architecture accumulates currents in the bitline that impose higher electrical loading in the memory array; hence, not only is the periphery circuit heavily modified but also the cell requires changes due to the high bitline current.

- **COM:** In **Computation-Out-of-Memory (COM)** architectures, the computation is performed in the extra logic circuit available inside the memory SiP (noted as position 3 in Figure 1(a)). If the computation is performed by off-memory computational cores (noted as position 4 in Figure 1(a)), then the architecture is similar to the classical Von-Neumann architecture and hence, it is not discussed in this article, as the article focuses on memory-centric architectures.

### 2.2 Memory Technology

Memory technologies can be classified as charge-based memories and non-charge-based memories. In charge-based memories such as **Dynamic Random Access Memory (DRAM), Static Random Access Memory (SRAM)**, and Flash, information is stored through the presence of charge [67, 121, 122, 134]. Whereas, the non-charge-based memories include different types of storage elements distinguished by their physical mechanism; these include resistive [16, 23, 59, 159, 196], "magnetic" memories [18, 20–22, 37, 68, 138, 159] and other types of memories such as molecular memories [71, 114, 115, 155] and mechanical memories [31, 73]) that can currently not be used for computing and are not discussed further in this classification.

*2.2.1 Charge-based Memories.* The SRAM and DRAM memories are largely adopted by the semiconductor industries. Both of these memories are volatile in nature, which means they require power supply to maintain their state. A six transistor bit-cell design is commonly used in SRAM, whereas DRAM bit-cell comprises a capacitor and a transistor. Although SRAM has faster accesses, its bit-cell size is much larger and consumes more leakage than DRAM. Despite the fact that DRAM has a significant advantage in terms of density, it requires periodic refresh to retain its data. Due to their volatility, both of these memories are facing serious power dissipation problems. On the contrary, Flash is a non-volatile memory that uses a floating gate transistor that has a charge trapping mechanism. Since flash uses only a single transistor, its density is significantly higher than that of DRAM. However, it requires high voltage and considerably long duration to write a value. Moreover, it has limited endurance due to gate oxide degradation under strong electric field, meaning it can be only employed for applications where a few write operations are required.

*2.2.2 Non-charge-based Memories.* RRAM, MRAM, and PCAM store information in the form of resistance states; these devices are thus also termed as memristors. These devices can be programmed in high resistance or low resistance states using reset or set electrical pulses. The RRAM cell consists of a top-electrode and a bottom-electrode that is separated by an oxide layer. Based on the formation/ disruption of a **Conductive Filament (CF)**, the resistive switching of RRAM devices takes place. The size of the CF determines the resistance state of the device. When a suitable positive voltage is applied, the breakage of ionic bonds increases the size of the CF, leading to a low resistance state of the device. However, when a suitable negative voltage is applied, some ions move back into the oxide region, thus reducing the size of the CF, resulting in a high resistance state. RRAMs are capable to perform multi-level bit storage.

In MRAM technologies, the value is stored in a **Magnetic Tunnel Junction (MTJ)** cell that consists of an oxide layer that is sandwiched between two ferromagnetic layers. Out of these two ferromagnetic layers, one is reference layer in which the magnetization is always fixed. The other one is known as free layer, whose magnetic orientation can be freely rotated, depending on the

Table 1. Design Metrics for Various Memory Technologies
(Data Obtained from References [147, 158])

| Comparison metric | SRAM (6T) | DRAM (1T1C) | Flash (1T) | RRAM (1T1R) | MRAM (1T1R) | PCRAM (1T1R) |
|---|---|---|---|---|---|---|
| Size ($F^2$) | 120–150 | 10–30 | 10–30 | 10–30 | 10–30 | 10–30 |
| Volatility | Yes | Yes | No | No | No | No |
| Write energy | ~fJ | ~10fJ | ~100pJ | ~1pJ | ~1pJ | ~10pJ |
| Write speed | ~1ns | ~10ns | 0.1–1ms | ~10ns | ~5ns | ~10ns |
| Read speed | ~1ns | ~3ns | ~100ns | ~10ns | ~5ns | ~10ns |
| Endurance | $10^{16}$ | $10^{16}$ | $10^4$–$10^6$ | $10^7$ | $10^{15}$ | $10^{12}$ |
| Scalability | medium | medium | medium | high | high | high |

direction of current flowing through it. When the magnetic orientation of these two layers are parallel and anti-parallel to each other, the cell exhibits low and high resistance states, respectively. Reading a value from the MTJ cell applies the principle of Tunneling Magneto-Resistance effect.

PCRAM exploits the large resistance contrast between the amorphous and crystalline states of a chalcogenide glass. The change from a high resistive amorphous phase to less resistive crystalline phase is induced by heating the material above its crystallization temperature for a certain duration. The reverse switching can be realized by melting and quenching the material using a reset electrical pulse. Due to these switching mechanisms, PCRAM devices have the capability to achieve a number of intermediate distinct states, enabling the feature of multi-bit storage similar to RRAM.

The unique characteristics associated with each memory technology are illustrated in Table 1. Besides to the regular memory operations, a range of in-memory logic and arithmetic operations can also be performed using these memory technologies [3, 58, 76, 172].

### 2.3 Computation Parallelism

Computation parallelism defines the level of parallelism that can be exploited in a computer system; i.e., task-, data-, and/or instruction-level parallelism. In task-level parallelism, a system has multiple independent control units and data memories; examples include multi-threading [54, 185] and multicore systems [69]. In data parallelism, a system has a single control unit used to apply the same instruction concurrently on a collection of data elements; examples include data elements with constant sizes (e.g., vector and array processor [52, 61]) and varying sub-word sizes (e.g., **SWAR (SIMD Within A Register)** processor [154]). In instruction-level parallelism, a system has a single control unit used to execute various instructions concurrently; examples include intra-instruction (e.g., pipe-lined processor [184]) and inter-instruction (e.g., VLIW processor [198]) parallelism. Additionally, they can be combined together as in speculative processor [124].

Based on the above discussed metrics, 36 classes can be differentiated by combining computation location with memory technology as shown in Figure 1(b); among those classes, 11 classes are occupied by the existing architectures, which are located in red and pink planes (see Figure 1(b)). The red plane demonstrates that a lot of work has been done for that particular class. The pink plane demonstrates a moderate number of work has been done. The cyan plane demonstrates either unexplored classes due to the lack of attention from the research community or non-existing due to current restrictions of memory technologies. The developments in memory-centric computing are shown in the timeline of Figure 2; this shows the trend of computing moving from processor-centric to memory-centric architectures (CIM-A, and CIM-P). In the figure, a larger circle indicates that more work has been proposed in that year. As it can be seen from the figure, the concept of merging computation and memory was introduced back in 1970.

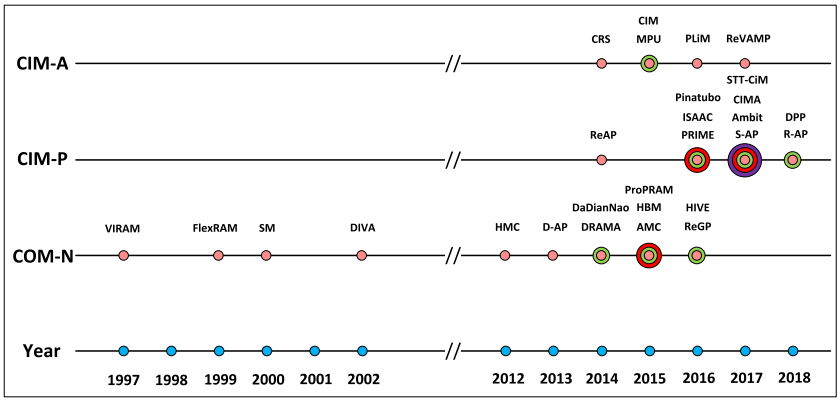



Fig. 2. Evolution timeline of memory-centric architectures.

Table 2. Comparison among Architectures of CIM-A Classes

| | Hierarchy level | Computations | | Memory Technology | Overheads | | Sneak path current | Destructive read | Require read-out* | Copy scheme | Evaluation | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Logic style | Available functions | | Periphery | Controller | | | | | Simulator | App. |
| **DRISA-3T1C** [116] | Accelerator | DRAM | Boolean | DRAM | Modif. | Simple | No | No | No | Both | CACTI-3DD, in-house | CNN |
| **CRS** [167] | Accelerator | CRS | Logical, + | RRAM | Conv. | Complex | No | Yes | Yes | Indirect | Hspice | No |
| **CIM** [51, 74, 76] | Accelerator | Varied | Logical, +, x | RRAM | Conv. | Varied | Yes | No | Varied | Both | Analytical | Parallel adder and multiplier |
| **PLiM** [5] | Main memory | Majority | Majority gates | RRAM | Conv. | Complex | Yes | No | Yes | Both | Analytical | Encryption |
| **MPU** [83] | Main memory | MAGIC | Logical, +, x | RRAM | Conv. | Simple | Yes | No | No | Both | Analytical | Image processing |
| **ReVAMP** [17] | Main memory | Majority | Majority gates | RRAM | conv. | Complex | Yes | No | Yes | Both | Analytical | EPFL benchmarks |

+: n-bit addition, Conv.: Conventional, (*): Required read-out during computations.
x: n-bit multiplication, Modif.: Modified, App.: Applications and benchmarks.

In the following sections, we will discuss the existing architectures classified based on their computation position. Please note that each computation position-based classes are consisting of architectures implemented using different memory technologies.

## 3 COMPUTATION-IN-MEMORY - ARRAY (CIM-A)

The CIM-A class contains mostly resistive computing architectures that use memristive-based logic circuits [50] to perform computations and **resistive RAM (RRAM)** as memory technology. Few architectures have been proposed in this category. Table 2 shows a brief comparison among the architectures that will be explained in each subsection.

On one hand, all these architectures have several common advantages:

- Low memory access/bandwidth bottleneck due to computing inside the memory.
- High data parallelism due to the possibility of performing concurrent operations inside the crossbars.
- Low leakage due to the usage of non-volatile memory technology and small footprint when compared to conventional memory technologies, but only in the case of very large memory array.

On the other hand, they all share several limitations:

- High computing latency per access due to the high latency of writing memristors and the need of multiple write steps to perform Boolean functions. Note that despite a high computing latency, the performance can be still high when sufficient parallelism is exploited.
- Higher endurance requirement due to the need of multiple write steps to perform Boolean functions.
- The cell designs are mostly modified to make the computing feasible.

The following subsections discuss the details and characteristics of each architecture. In this subsection, DRAM-based architecture is presented first, and the remaining RRAM-based architectures are discussed in a chronological order of their date of publication. This ordering technique is also reflected in Table 2.

### 3.1 DRISA-3T1C: A DRAM-based Reconfigurable In Situ Accelerator with 3 Transistors and 1 Capacitor (3T1C) Design

DRISA-3T1C was proposed in 2016 by S. Li et al. from University of California [116]. It is a DRAM-based architecture that exploits data parallelism by performing NOR gate inside DRAM cells [166]. The architecture consists of a DRAM memory organized in a hierarchy of banks, sub-arrays, and mat; each levels is controlled by their corresponding controllers as shown in Figures 3(a), (b), and (c). The banks are connected through **global bus (gbus)**, while communication among subarrays is carried out using **bank buffers (bBuf)**. The mats perform both data storage and computations.

The memory mats consist of cell regions for both data and computations and peripheral circuits including **calc-SA (Sense Amplifier)**, intra/inter-lance shifter, and lane forwarding unit. The cell regions contains multiple DRAM cells that consist of three transistors connected to form a NOR gate and one capacitor to store the data value (see Figure 3(d)). To perform computations, two DRAM cells (Rs and Rt) to be activated simultaneously and one DRAM cell (Rr) to store the computation result (as shown in Figure 3). Read voltages are applied to the sources DRAM cells (Rs and Rt) through the wordline (rWL), while write voltage is applied to the result DRAM cell (Rr). The voltage collected by the **sense amplifier (SA)** is used to control the transistor in the Rr DRAM cell as shown in Figure 3(d). Due to the NOR organization of these transistors, a NOR operation is realized and produces results in DRAM cells. The SA (also called calc-SA) cooperates with extra logic circuitry such as SHF and FWD to perform complex functions such as addition, copy, and inner product. DRISA-3T1C has the following advantages on top of the general advantages of CIM-A architectures:

- The latency of NOR primitive functions is fixed.
- The data transfer may include both direct and indirect schemes.
- The architecture does not suffer destructive read as in the case of CRS architecture [167], hence the write energy might be less due to the absence of write-after-read.
- The controller is simpler than for the CRS architecture, as each operation consists of a fixed number of steps while fewer control voltage values are used.
- The architecture uses DRAM technology, which has several benefits, such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology, and tools.

In spite of those advantages, DRISA-3T1C also has its own set of limitations that impact its effectiveness. The limitations of DRISA-3T1C include:

- The latency of complex functions varies, depending on the functional complexity, as each function needs to be converted into multiple NOR gates.

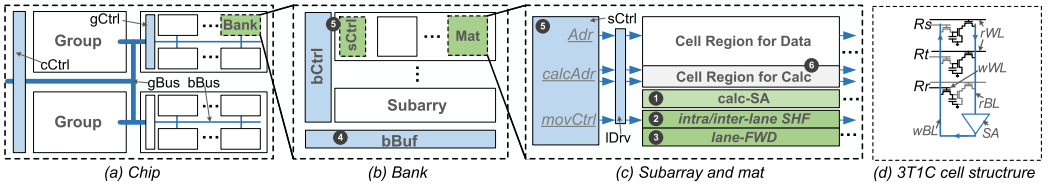



Fig. 3. A DRAM-based Reconfigurable In Situ Accelerator (DRISA) [116].

- The architecture uses DRAM technology, which suffers from low performance, high energy consumption, large footprint, and is difficult to scale down.

The architecture is simulated and evaluated against GPU TITAN X [146] using four CNN applications, including 8-layer AlexNet [106], 16-layer VGG-16 [168], 19-layer VGG-19, and 152-layer ResNet-152 [79].

### 3.2 CRS: Complementary Resistive Switch Architecture

**Complementary Resistive Switch (CRS)** architecture was proposed in 2014 by A. Siemon et al. from RWTH Aachen University [167]. It is a memristor-based architecture that exploits data-level parallelism using implication logic. The architecture shown in Figure 7 consists of multiple crossbars and a control unit. The crossbar stores and performs logic operations using CRS cells; a CRS cell consists of two resistive switches or resistive RAMs. The control unit distributes signals to the intended addresses (wordlines and bitlines) to perform operations on the crossbars.

The crossbar is controlled by a sequence of operations including: **write-in (WI), read-out (RO), write-back (WB)**, and **compute (CP)**. Before the operations can be performed, the crossbar part used for computation is entirely reset to a logic value 0. The WI operation writes a logic value into a memristor. The RO operation reads a logic value from a cell; the logic output value is determined by the sense amplifier. The RO operation is destructive and changes the value of the memristor to logic value 1. The task of the WB operation is to recover the destroyed value. Finally, the CP instructions are used to execute the implication logic gates [118, 167]. The data transfer between CRS cells is carried out through the control unit using a RO and WB operations; in other words, the control unit reads a value of the source CRS cell and writes it into the destination cell. In addition to the general advantages of CIM-A architectures, CRS has the following advantages:

- It is less impacted by the sneak path currents due to the usage of CRS cells. The cell's resistance is always equivalent to high resistance, hence, sneak path currents are eliminated. However, variations in resistances will make such paths practically unavoidable unless a 1T2R cell is used.
- CRS logic requires fewer cells to perform computations than **Fast Boolean Logic (FBL)**, which is required to express the sum-of-product format.
- It is possible to use sufficiently large crossbar arrays, meaning effective overhead due to control circuits can be minimized.

However, it also has the following limitations:

- The latency of the primitive functions varies and requires extra read-out instructions to determine the voltages that have to be applied.
- The RO operation is destructive, hence, a WB operation is required after each RO operation, which increases the latency and energy of computations.
- The data transfer method is indirect, as it is based on the read-out and write-back scheme. As all cells have high resistance, direct copying of cells in the crossbar is not applicable.

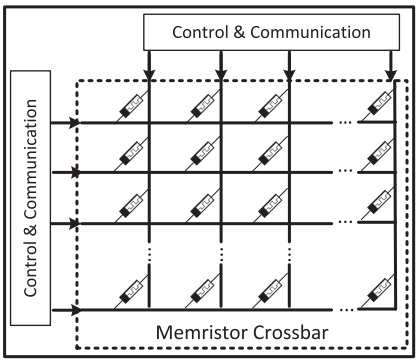



Fig. 4. The Computation-In-Memory Architecture (CIM) [51].

- The control unit imposes a high overhead, as it is responsible for both controlling the crossbar (requiring a large number of states) and transferring data (which involves the usage of buffers/registers to store temporary values).
- The area of CRS cells is larger than those based on single memristor cells.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to implication logic.

This architecture was only evaluated at circuit level using adders. Therefore, it is hard to make general conclusions on the performance and the applicability of this architecture.

### 3.3 CIM: Computation-In-Memory

CIM was proposed in 2015 by H. A. Du Nguyen et al. from Delft University of Technology [51, 74, 76]. It is a memristor-based architecture that exploits data-level parallelism by using any memristor logic style; the authors have showed the potential of this architecture using **Fast Boolean Logic (FBL)** [202] and implication logic [167]. The architecture consists of a memristor crossbar and a control and communication block as shown in Figure 4 [74]. The memristor crossbar stores data and performs computations. The control and communication block applies appropriate voltages to the memristor crossbar.

The architecture uses state machines stored in the *control and communication block* to compute and transfer data in the crossbar. Once triggered, the state machine applies an appropriate sequence of control voltages to the rows and columns of the memristor crossbar. Depending on the memristor types, the data transfer occurs directly inside the crossbar (for single RRAM cells) or indirectly through the *control and communication block* outside the crossbar (for CRS cells) using the CRS read-out/write-back scheme. In addition to the general advantages of CIM-A architectures, CIM comes with the following set of advantages:

- The architecture can accommodate any type of memristor logic design due to the flexibility of the control and communication block.
- In case FBL is used, the latency of primitive functions (i.e., addition, multiplication) is a constant number.
- The data transfer using both direct and indirect schemes has been intensively explored in References [49, 200].
- The control block of FBL is less complex than the control block of implication logic due to a fixed number of write steps and a simpler control voltage scheme [202].
- Compared to CRS architecture, CIM architecture has significant area and write energy advantage, as these values are much less at a single respective cell level.

However, it also has the following limitations:

- The architecture has to deal with sneak path currents in case a single RRAM cell (0T1R) is used, as multiple rows and columns are activated simultaneously. Possible solutions to alleviate the problem consist of isolating each FBL circuit, or the usage of a transistor-memristor (1T1R) structure to actively control each memristor using a transistor [126, 204], or isolated/half select voltages [27, 201].
- In case FBL is used, typically a lot of cells are required due to LUT-based computing.
- In case CRS cells are used, the same drawbacks of CRS architecture apply, i.e., a larger cell area, the control unit imposes a high overhead, as the controllers are responsible for both controlling the crossbar and transferring data, and it requires additional compiling techniques and tools to convert conventional Boolean logic functions to implication logic.

The potentials of the architecture are demonstrated using a case study of a binary-tree-based parallel adder and multiplier [51, 77]; the architecture is compared with a conventional multicore architecture. CIM architecture achieves at least one order of magnitude improvement in terms of delay, energy, and area.

### 3.4 PLiM: Programmable Logic-in-Memory Computer

Similar to the DRAM-based bit-serial addition using majority logic [5], PLiM was proposed in 2016 by P. Gaillardona et al. from EPFL [65]. It is a memristor-based architecture that exploits data parallelism using majority logic [166] to perform elementary Boolean logic such as ***OR*** and ***AND*** operations within the memory array. The architecture consists of a resistive memory organized in banks and a **Logic-in-Memory (LiM)** controller block as shown in Figure 5. The memory is a memristive crossbar that stores both instruction and data. The LiM controller is composed of a number of registers and a **finite state machine (FSM)**. The controller functions as a simple processor; it fetches instructions from the memory array, decodes and executes the operation inside the memory.

The LiM controller operates in two modes: conventional memory read/write mode and in-memory instruction mode. In the read/write mode, the FSM is deactivated, and the memory array is read or written in the same manner as a standard memory. In the in-memory instruction mode, FSM is activated, and an instruction is performed using majority logic gates inside the memory. Once the FSM is enabled, the following operations are performed: First, the FSM resets all registers in the LiM controller. Second, an instruction is read from the address in the **program counter (PC)** and decoded to obtain the addresses of the two operands and output; the addresses of the two operands are stored in registers @A and @B, while the output address is stored in register @Z. Third, the values of the two operands are read using the addresses in registers @A and @B; the obtained values are stored in registers @A and @B, respectively. Fourth, depending on the logic values (0 or 1) of the operands, appropriate voltages are applied to the crossbar at address @Z to perform a majority logic gate. Finally, the PC is incremented by one. On top of the general advantages of CIM-A architectures, PLiM has the following additional advantages:

- The data transfer may include both direct and indirect schemes.
- The write energy and area of a memristor cell is smaller as compared to a CRS cell.

Similarly, it also has its own limitations that are stated as follows:

- The latency of majority primitive functions varies, depending on the functional complexity, and some read-outs are required to determine the voltage values to be applied.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.

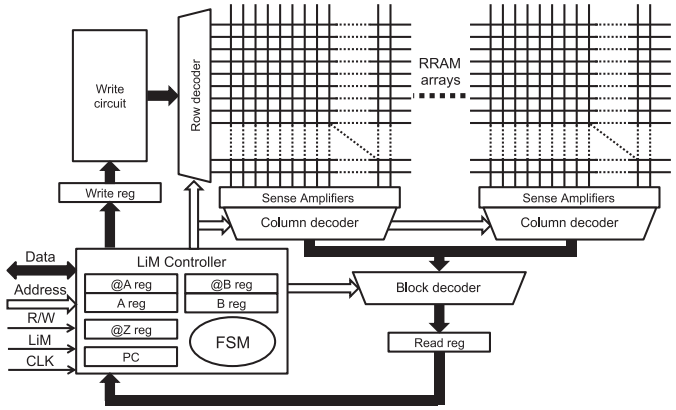



Fig. 5. The Programmable Logic-in-Memory Computer (PLiM) [65].

- The LiM controller is complex, as it has to determine the control voltage values based on the operands' values.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to majority logic gates.

The architecture is evaluated with a PRESENT Block Cipher algorithm [24], which encrypts a 64-bit plain text with an 80 or 128-bit key. The algorithm is compiled into a sequence of majority logic gates and executed on PLiM. Unfortunately, the results show that PLiM's performance is almost a factor of two slower than a 180 nm FPGA implementation [24].

### 3.5 MPU: Memristive Memory Processing Unit

MPU was proposed in 2016 by R. Ben Hur et al. from Technion-Israel Institue of Technology [83]. Similar to the DRAM-based architecture proposed in Reference [66] and SRAM-based solutions in References [1, 53, 64, 130, 156, 194], MPU is a memristor-based architecture that exploits data-level parallelism using **Memristive-Aid loGIC (MAGIC)** [108]. The architecture consists of a conventional processor, MPU controller, and a memristive memory as shown in Figure 6. The processor contains a control unit, an arithmetic and logic unit, and a memory controller. The MPU controller includes a Processor-In/Out block to interface to the conventional CPU, control blocks to execute specific commands (arithmetic, set/reset, read and write block) and an output multiplexer to apply appropriate voltages to specific rows/columns of the memristive memory. The conventional processor sends an instruction to the memristive memory using its own memory controller and the MPU controller. The memory controller of the processor recognizes the memristive memory instructions in a similar manner as conventional memory operations, while the MPU controller determines whether to treat the memristive memory as a storage element or a processing element. Based on that, the MPU controller applies read/write signals or a sequence of signals to perform logical or arithmetic operations.

The MPU controller uses the Processor-In unit to divert the instructions to specific blocks (such as arithmetic, read and write blocks) responsible for the execution of those operations. Each block determines the appropriate voltages that have to be applied to the memristive memory. The set/reset, read and write block have a latency of 1 cycle, while the arithmetic block requires multiple cycles to execute a vector operations using MAGIC logic [108]. Data movements in the crossbar are performed directly using *copy* (double negation) or single NOT operations. In addition to the general advantages of CIM-A architectures, MPU has the following advantages:

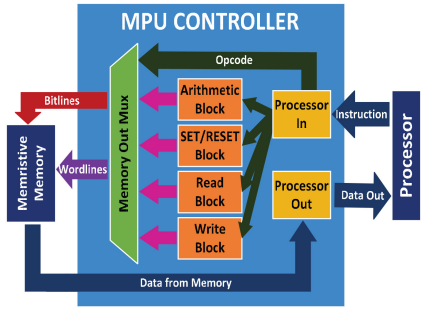



Fig. 6. The Memristor Memory Processing Unit (MPU) [83].

- The latency of MAGIC primitive functions is fixed.
- The data transfer may include direct (based on copying) and indirect (based on read-out/write-back) schemes.
- The MPU controller is simpler than for the CRS architecture, as each operation consists of a fixed number of steps while less controlling mechanisms are used.
- The write energy of a single MAGIC cell is smaller in comparison to those of a CRS cell.
- MAGIC requires in comparison to FBL fewer cells to perform computations.
- MPU can perform many MAGIC gate operations in parallel, which makes it very efficient in terms of throughput.
- MPU is compatible with the conventional Von-Neumann machines, meaning it can perform load/store/compute) operations.

In spite of the above-mentioned benefits, MPU also has its own set of limitations, which include:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- The control voltages used in MAGIC have to satisfy the constraint $2V_{reset} < V_w < V_{set}$, where $V_{set}$ is the minimum voltage required to switch a memristor from $R_H$ to $R_L$, and $V_{eset}$ is the minimum voltage required to switch a memristor from $R_L$ to $R_H$; in other words, it requires that the memristors have a higher $V_{set}$ than $V_{reset}$, leading to an unbalanced hysteresis loop. This limits the types of memristors that can be used for MAGIC.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to MAGIC gates.

The potential of the architecture is demonstrated by performing a logical bit-wise OR operation of two 8-bit vectors in 20 steps. The latest research has shown that MAGIC can be used for several arithmetic operations, such as addition, multiplication, and so on [72, 87, 109]. In this context, there is an architecture named SIMPLE MAGIC that automatically generates a defined sequence of atomic memristor-aided logic NOR operations [84]. In the same line, a SIMPLER flow is developed that additionally optimizes the mapping executions that reduce complexity and improve throughput [15]. Note, a lot of research on cell minimization and cell reuse has been done in the context of MAGIC and other basic cells.

### 3.6 ReVAMP: ReRAM-based VLIW Architecture

ReVAMP was proposed in 2017 by D. Bhattacharjeeet al. from Nanyang Technological University [17]. It is a memristor-based architecture that exploits data parallelism using majority logic. The architecture consists of an **Instruction Fetch (IF), Instruction Decode (ID)**, and **Execute**

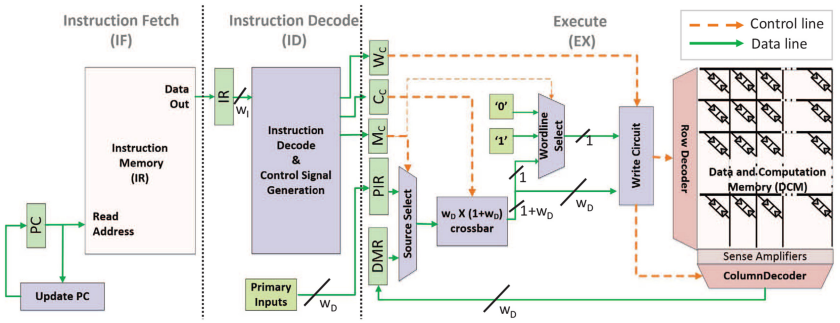



Fig. 7. ReRAM-based VLIW architecture (ReVAMP) [17].

**(EX)** stage. The IF block fetches instructions from the Instruction Memory using the **program counter (PC)** as address and stores it in the **Instruction Register (IR)**. The ID block decodes the instruction and generates control signals that are placed in the control registers of the EX block. The EX stage finally executes the instruction.

The IF and ID stages are similar to those of the traditional five-pipelined RISC architectures. The IF stage includes an **Instruction Memory (IM)** and a **Program Counter (PC)**. The ID stage contains registers (IR and Primary Inputs) and an Instruction Decode and Control Signal Generation. The EX stage consists of several registers (i.e., **Data Memory Register (DMR), Primary Input Register (PIR), Mux control ($M_c$) register, Control ($C_c$) register, Wordline ($W_c$)** register), as well as a crossbar interconnect, wordline select multiplexer, data Source Select multiplexer, and a Write circuit to control the crossbar that stores data. Once an instruction is fetched and decoded in IF and ID, respectively, the control registers in EX stage are filled with suitable values. These values control the multiplexers that are responsible for applying the right control signals to the crossbar. Depending on the operation, primary inputs from PIR or data retrieved from the crossbar stored in DMR can be used for the next operation. The crossbar interconnect permutes the inputs and control signals (indicated by $C_c$) to generate the voltages that need to be applied to the memory crossbar. The Write circuit applies these voltages to the targeted wordline address (indicated by $W_c$). In addition to the general advantages of CIM-A architectures, ReVAMP has the following advantages:

- The data transfer may include direct (within the crossbar based on copying resistance values) and indirect (based on read-out/write-back) schemes.
- The crossbar is based on only one device per cell, resulting in a more compact architecture as compared with other architectures that make use of two devices per cell (i.e., **Complementary Resistive Switch (CRS)** [167]).

However, it also has the following limitations:

- The latency of majority primitive functions varies, depending on the functional complexity; in addition, before any operations are applied to the cells, these cells first have to be read-out to determine the appropriate control voltages.
- The architecture has to deal with sneak path currents. Possible solutions to alleviate the problem consist of isolating each tile/crossbar, or using a transistor-memristor (1T1R) structure to actively control each memristor using a transistor [126, 204], or using isolated/half select voltages [27, 190, 201].
- The EX stage is complex, as it integrates control signals for memory and computations. Therefore, it is not easy to pipeline this architecture, as the EX stage will consume more time than the other stages; i.e., the stages IF, ID, and EX are not balanced.

- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to majority logic gates.

The architecture is simulated and evaluated using EPFL benchmarks [10] and compared against PLiM [65], which is based on a resistive memory with the same logic style. The compiler level of execution for such architecture can be performed as demonstrated in Reference [175].

## 4 COMPUTATION-IN-MEMORY - PERIPHERALS (CIM-P)

The CIM-P class consists of architectures that perform computations during read-out operations (i.e., two or more word lines are activated simultaneously) using special peripheral circuitry. Earlier memory-centric works explored the potential of shifting data-intensive computations to the memory system with/without reconfigurable logic includes Active Pages [26, 95, 149, 180]. These works utilized conventional DRAM and magnetic bubble memories to realize memory-centric architecture. However, as there are less restrictions on the functionality of the cell, various memory technologies can be used in this category, such as DRAM, SRAM, and non-volatile memory technologies. For instance, a medium number of architectures have been proposed in this category. Table 3 shows a brief comparison among the architectures that will be explained in each subsection.

On one hand, these architectures have several common advantages:

- Low memory access/bandwidth bottleneck, as the results are produced in the peripheral circuitry that is connected directly to the memory array.
- High parallelism due to the the possibility of performing multiple concurrent operations.
- High performance, as computations are performed in a single read step.
- Relatively simple controllers, as the operations are constructed in a similar manner as for conventional memory (read/write) operations.
- Higher compatibility with available memory technologies, because redesigning cells would induce a huge cost for the vendors.
- Lower endurance requirement, as operations are based on reading instead of writing [203].

On the other hand, they all share the following limitations:

- Overhead to align data; note that each operation requires the data to be aligned in the memory. Therefore, if the operands are not located in the same crossbar, then data transfer operations are required.
- Additional write overhead when the results have to be stored back into the memory. Note that the outputs are produced as voltages in the peripheral circuit, and therefore, if the results have to be stored back in the memory, then extra write operations would be necessary.
- Parallelism is possible, but it is achieved at the cost of area and power overhead.

Similar to CIM-A architectures, the following subsections discuss the details and characteristics of each CIM-P architecture. In this section, architectures that utilize charge-based memory technology (DRAM and SRAM) are presented first, followed by the discussion of architectures based on non-charge-based memory technology (e.g., Resistive, Magnetic, etc.). Both charge and non-charge-based memory technology architectures are discussed in chronological order of their date of publication. This ordering technique is also reflected in Table 3.

### 4.1 Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Ambit was proposed in 2017 by V. Seshadri et al. from Carnegie Mellon University [163]. Ambit is a DRAM-based architecture that performs in-memory instructions using modified peripheral

Table 3. Comparison among Architectures of CIM-P Classes

| | Hierarchy level | Computations | | Memory Technology | Overheads | | Sneak path current | Destructive read | Required read-out* | Copy scheme | Evaluation | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Logic style | Available functions | | Periphery | Controller | | | | | Simulator | App. |
| **Ambit** [163] | Accelerator | Bool. | Logical | DRAM | Modif. | Simple | No | No | No | Both | Rambus | Bitwise |
| **S-AP** [181] | Accelerator | Bool. | Logical,+ | SRAM | Modif. | Simple | No | No | No | Both | VASim | ANMLZoo &Regex |
| **ReAP** [205] | Accelerator | CAM | LUT-based | RRAM | Modif. | Simple | Yes | No | No | Both | Analytical | Arithmetic |
| **Pinatubo** [117] | Main memory | Bool. | Logical | RRAM | Modif. | Simple | No | No | No | Both | In-house | Bitwise |
| **ISAAC** [164] | Accelerator | NN. | MM. | RRAM | Modif. | Simple | Yes | No | No | Both | Analytical | CNN& DNN |
| **PRIME** [39] | Main memory | NN. | MM. | RRAM | Modif. | Simple | Yes | No | No | Both | Analytical | Arithmetic |
| **ReGP** [131] | Main memory | Bool. | Logical,+,x | RRAM | Modif. | Medium | No | No | No | Indirect | Analytical | MM. |
| **CIMA** [50] | Accelerator | Bool. | Logical | RRAM | Modif. | Simple | Yes | No | No | Both | Analytical | Bitwise |
| **STT-CiM** [92] | Accelerator | Bool. | Logical,+ | MRAM | Modif. | Simple | Yes | No | No | Both | STT-CiM Sim. | (1) |
| **DPP** [63] | Accelerator | Bool. | Logical,+,x | RRAM | Modif. | Simple | Yes | No | No | Both | TensorFlow, CACTI | PARSEC |
| **R-AP** [206] | Accelerator | MM. | Logical,+,x | RRAM | Modif. | Simple | No | No | No | Both | Hspice | No |

+: n-bit addition, CNN: Convolutional Neural Network, (*): Required read-out during computations.
x: n-bit multiplication, Modif.: Modified, App.: Applications and benchmarks.
LUT: Lookup Table, MM.: Matrix Multiplication, Analytical: Analytical model.
NN.: Neural Network, Bool.: Boolean, MRAM: Magnetic Random Access Memory.
STT-CiM Sim.: STT-CiM device to architecture evaluation framework.
(1): string matching, text processing, low-level graphics, data compression, bio-informatic, image processing, and cryptography.

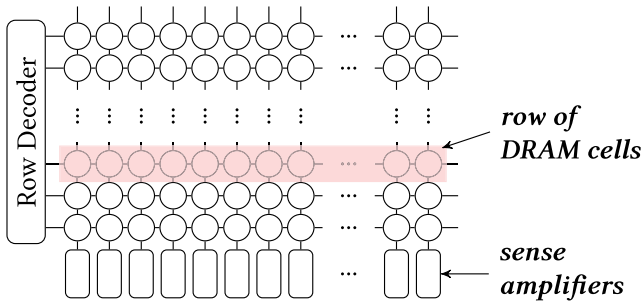



Fig. 8. In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology (Ambit) [163].

circuits to exploit data-level parallelism, which is achieved by computing all bits in a row(s) in parallel. This architecture can be plugged into a computer system as an accelerator in a similar manner as a GPU. The architecture consists of an Ambit controller and a 3D-stacked DRAM memory with modified sense amplifiers as shown in Figure 8. After receiving an instruction from the host processor, Ambit determines whether a normal memory operation or an in-memory instruction should be performed. After performing the required operations, the results are transferred back to the host processor for further processing. Depending on the type of operation, the Ambit controller activates single or multiple rows in the DRAM memory. AMBIT architecture can perform any bit-wise operations at column level using basic set of operations such as AND, OR, and NOT. The currents are summed up based on the values stored in the DRAM cells and converted into a digital value using the modified sense amplifiers. To transfer data, Ambit enables *row copy* (RowClone [162]) operations to directly move data inside DRAM memory. Moreover, an indirect scheme can be used as well by having the Ambit controller performing *read* and *write* operations.

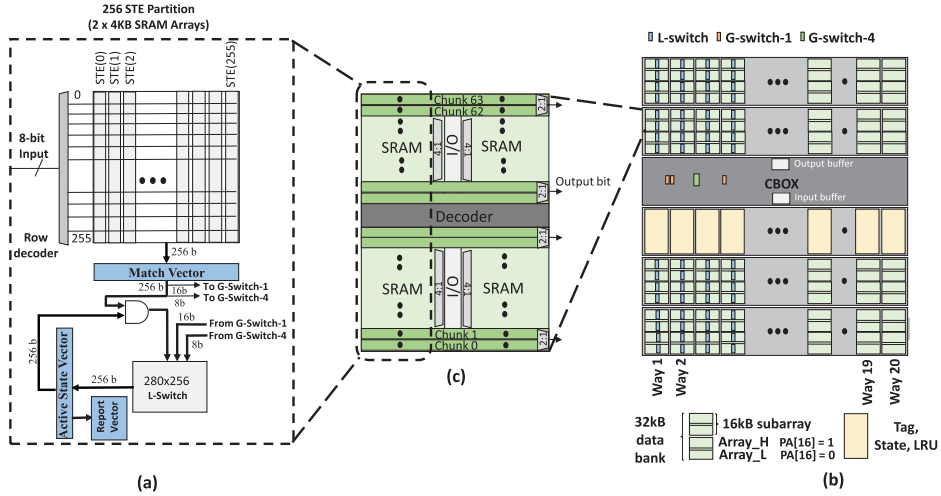



Fig. 9. SRAM Automata Processor(S-AP) [181].

In addition to higher throughput, Ambit has the following advantages on top of the generic CIM-P architecture advantages:

- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses DRAM technology, which has several benefits, such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology, and tools.
- Since the pitch of the SRAM bit-cell is more, it can easily accommodate the modified version of the sense amplifier in a column.

However, it also has the following limitations:

- Computations are currently limited to logical operations. More research is required to map complex functions on the architecture.
- The architecture uses DRAM technology, which suffers from low performance, high energy consumption, large footprint, and is difficult to scale down.

The architecture is simulated by Rambus simulator and evaluated against the implementations on multicore Intel Skylake CPU [88], NVIDIA GeForce GTX 745 GPU [145], and HMC 2.0 [127] using logical vector operations and bitmap index application [34, 177]. The simulation results demonstrated that Ambit can deliver higher throughput than Skylake, GTX, and HMC 2.0 architectures.

### 4.2 S-AP: Cache Automaton

S-AP shown in Figure 9 was proposed in 2017 by A. Subramaniyan et al. from University of Michigan [181]. The architecture targets an automata processor that exploits data-level parallelism by performing computations using state machines. An automata processor contains two main components: the **State Transition Elements (STEs)** and the routing matrix; the STE stores the accepting states, while the routing matrix stores the state transitions as shown in Figure 10. The automata processor accepts one input symbol at a time, generates next active states, and decides whether a complete input string is accepted or not.

The architecture consists of STEs and a routing matrix that are implemented using SRAM technology. Each SRAM column corresponds to an STE that stores the accepting states in SRAM cells.

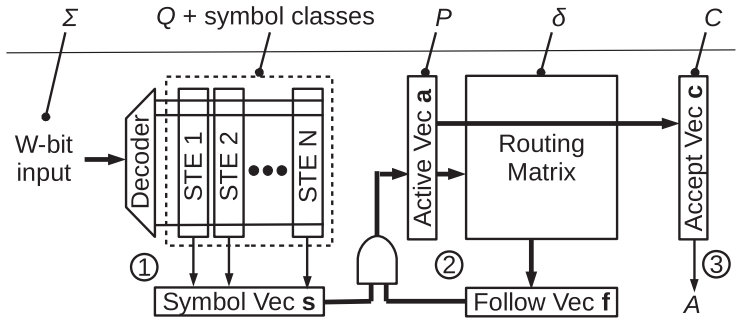



Fig. 10. General architecture for automata processor [206].

The input symbol is fed to all the STEs simultaneously. The sense amplifiers collect a dot-product results of a vector-matrix multiplication. The output of the STE together with the routing matrix are used to determine the next active states; this process is carried on until all input symbols are processed. In case the one or more final active states are part of the acceptance states, it means that the input string has been matched with the corresponding pattern of the acceptance state. Note that data transfer inside the automata processor is carried out using the routing matrix. In addition to the general advantages of CIM-P architectures, S-AP has the following advantages:

- Computations may include logical and arithmetic operations using automata processing.
- Data can be transferred using both direct and indirect schemes.
- The architecture uses SRAM technology, which has several benefits, such as maturity, high endurance, no sneak path currents, and may benefit for the the existing optimizing techniques and tools.
- Since the pitch of the SRAM bit-cell is more, it can easily accommodate the modified version of the sense amplifier in a column.
- The automata processing techniques and tooling are quite mature, hence it is feasible to explore many applications using automata processing.

However, it also has the following limitations:

- The architecture uses SRAM technology, which suffers from high energy consumption, low scalability, and large footprint.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using automata processing.

On one hand, S-AP has potential in reducing non-memory components required to implement the automata processor. The D-AP requires most of its resources for routing matrix and other logic, while the S-AP can be implemented on processor, which has advantages in realizing logic functions. On the other hand, S-AP suffers from low frequency, density, and latency due to SRAM intrinsic properties. The S-AP is simulated using VASim [191] and evaluated against DRAM-AP and x86 CPU using ANMLZoo [191] and the Regex [13] benchmark suites.

### 4.3 ReAP: Resistive Associative Processor

ReAP was proposed in 2014 by L. Yavits et al. from Technion-Israel Institute of Technology [205]. ReAP is a RRAM-based architecture that exploits data parallelism using LUTs implemented with **Contend Addressable Memories (CAMs)** to perform computations. The architecture consists of a crossbar of resistive CAM cells and peripheral circuits including sense amplifiers and registers

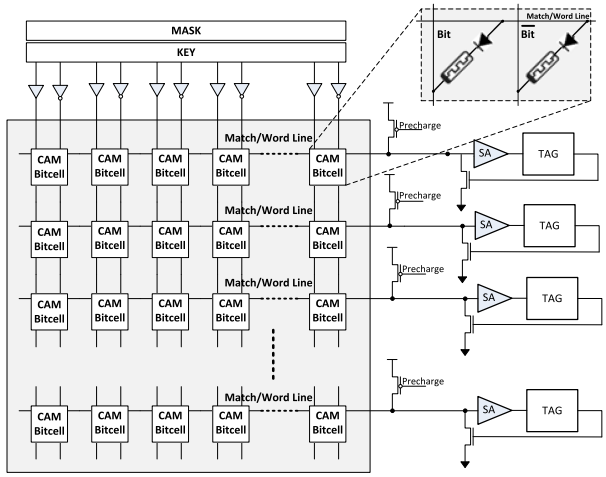



Fig. 11. Resistive Associative Processor (ReAP) [205].

(as shown in Figure 11). In this architecture, multiple CAM cells are employed to create a **look-up-table (LUTs)**; together, they implement a specific logic function. A single CAM cell comprises two resistive RAM cells that store the true and the complementary value of a single bit. When it is activated, it performs a compare operation with the inputs stored in register KEY to the LUT words and produce in case a match occurs the corresponding outputs in TAG registers.

The *compare* operation is performed in a similar manner as in conventional CAMs. First, the Match/Word line is pre-charged. Thereafter, the values in *KEY* are applied to the bit-lines, depending on the *MASK* value; if a bit is masked, then it is kept floating. If the *KEY* matches the content on a particular wordline, then the TAG will generate the value "1" at the output, otherwise "0." For example, in case a key bit is 1, both the true (i.e., low resistance) and complement value (i.e., high resistance) will keep the floating word line high in case a 1 is stored. In case the cell holds a 0, i.e., the true memristor has a high resistance and complement a low resistance value, the complement path will discharge the Match/Word line. Similar conclusions can be drawn in case the key bit is 0. To execute a more complex function, LUTs can be reconfigured. In such cases, the output of the LUT is fed back to input of the same LUT but with a different configuration. Another option is to implement the function using multiple LUTs. ReAP architecture has additional advantages on top of the generic CIM-P architecture advantages. The incremental advantages of ReAP are summarized as follows:

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [35, 193]. In contrast, some CIM-P architectures are used as main memory and they require a much higher endurance.
- Computing based on LUTs is quite mature (e.g., in FPGAs) and can benefit from existing techniques and tools.
- The architecture uses non-volatile memory, hence consumes a low amount of energy and has a small footprint.

However, it should also be noted that ReAP has several limitations that can severely affect its applicability. The limitations of ReAP include:

- Computations using LUTs can be inefficient if the number of inputs per LUT is large. If multiple smaller LUTs are used, then the latency becomes higher.

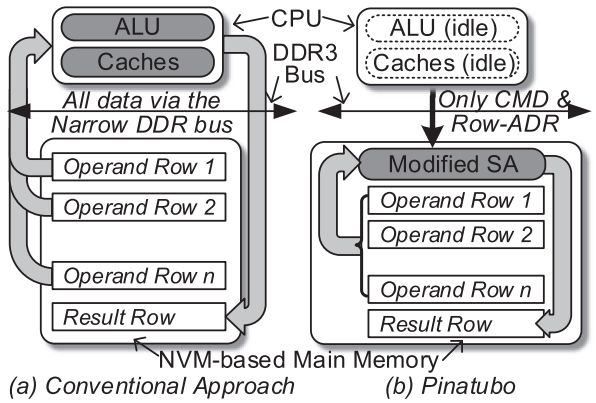



Fig. 12. The Processing-in-Memory architecture for bulk bitwise operations (Pinatubo) [117].

- The data transfer consists of an indirect read-out scheme.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- The write operations of this architecture may suffer from high energy consumption, as two memristors are written per CAM cell.
- The architecture might not exploit the full memory bandwidth, as it is challenging to fit all sense amplifiers into the memory core.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to CAM-based LUTs.

The architecture was evaluated analytically using several benchmarks [40] such as N-pairs Black-Scholes option pricing, N-point Fast Fourier Transform, and Dense Matrix Multiplications. They compared the results of these benchmarks on ReAP with two other platforms: a CMOS equivalent of ReAP denoted by CMOS-AP and GTX480 GPU. The results show that ReAP outperforms GTX480 in terms of performance (8 GFLOPs/W for ReAP versus 5 GLOPs/W for GTX480), while it is being outperformed by CMOS AP (18 GFLOPs/W).

### 4.4 Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations

Pinatubo was proposed in 2016 by S. Li et al. from University of California [117]. Pinatubo is a non-volatile memory-based architecture that exploits data-level parallelism by performing bulk bitwise operations using modified sense amplifiers. The architecture consists of a processor with caches, a non-volatile main memory, and modified sense amplifiers (as shown in Figure 12). The processor sends in-memory instructions to the main memory and also handles the operations that cannot be performed on the main memory. After an instruction is sent to the main memory, single or multiple rows of the memory are activated simultaneously, depending on the type of instructions (i.e., normal read or in-memory instructions). The modified sense amplifiers thereafter perform a read-out operation to produce the results, which can be a normal read or a bitwise vector operation. In case needed, the results are transferred back to the processor for further processing.

The main memory architecture is shown in Figure 13; it consists of multiple banks that are further divided into mats. Note that the modified sense amplifiers can only perform bitwise vector operations on data residing in the same mat. For operations where the data resides in different mats whether on the same bank or not, extra logic gates (e.g., AND, OR) are used to perform the operations. Communication can be performed by enabling two memory rows for direct copy

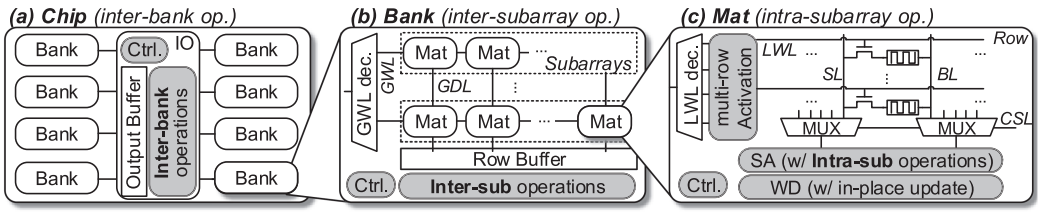



Fig. 13. Main memory of Pinatubo [117].

operations or using the buffers and read-out operations for indirect data transfer. In addition to the general advantages of CIM-P architectures, Pinatubo has the following advantages:

- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, Pinatubo has its own limitations, which are listed as follows:

- The architecture uses non-volatile memory as main memory, which may impact the lifetime due to limited endurance [35, 193].
- Computations currently include only logical operations. More research is required to map complex functions on the architecture.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.
- Efforts are required to use this architecture with a host processor. For example, the instruction set of the host processor has to be adapted and additional software support is needed to maximally exploit the performance.

The architecture is simulated using an in-house cycle accurate simulator modified from Sniper [32] and evaluated using three applications: vector OR operations, bitmap-based BFS for graph processing [12], and bitmap-based database using Fastbit [199].

### 4.5 ISAAC: A Convolutional Neural Network Accelerator with In Situ Analog Arithmetic

ISAAC was proposed in 2016 by Ali Shafiee et al. from University of Utah [164]. ISAAC is a memristor-based architecture that performs dot-product computations using the memristor crossbar and CMOS peripheral circuitry to exploit instruction-level parallelism. The architecture consists of multiple tiles connected through an on-chip concentrated mesh and an I/O interface, as shown in the left part of Figure 14. The architecture is only used during the inference phase of machine learning applications, i.e., the phase after training; the inference phase consists of dot product operations to compute convolutions, shift and add operations, and sigmoid operations. ISAAC processes inputs from the I/O interface in multiple tiles. After processing, the outputs are communicated through the I/O interface to the outside world or a different ISAAC chip. Each tile of ISAAC contains multiple **In Situ Multiply Accumulate (IMA)** units that are connected through a bus, an eDRAM buffer, **output register (OR)**, and computation units (max-pool, sigmoid, and **Shift-and-Add (S+A)**). Each IMA contains multiple memristor arrays with their DAC and **Sample-and-Hold (S+H)** units, an **Input and Output Register (IR, OR)**, S+A, and multiple ADC units. Inputs from the I/O interface are delivered to the memristor arrays and are used to perform a dot product computation with the weights that are already stored in the memristor array.

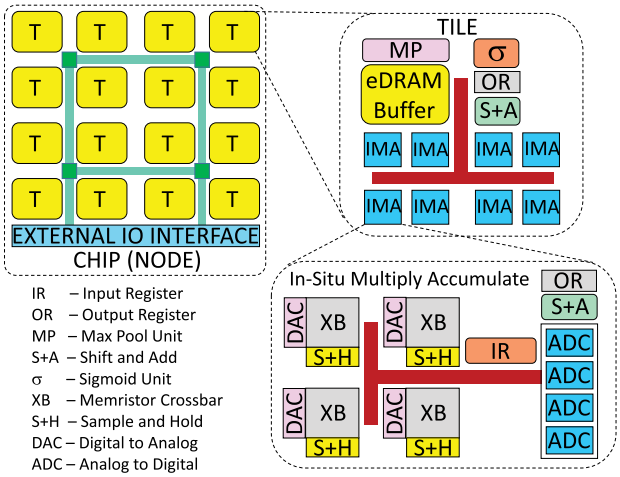



Fig. 14. A convolutional neural network accelerator with In Situ Analog Arithmetic (ISAAC) [164].

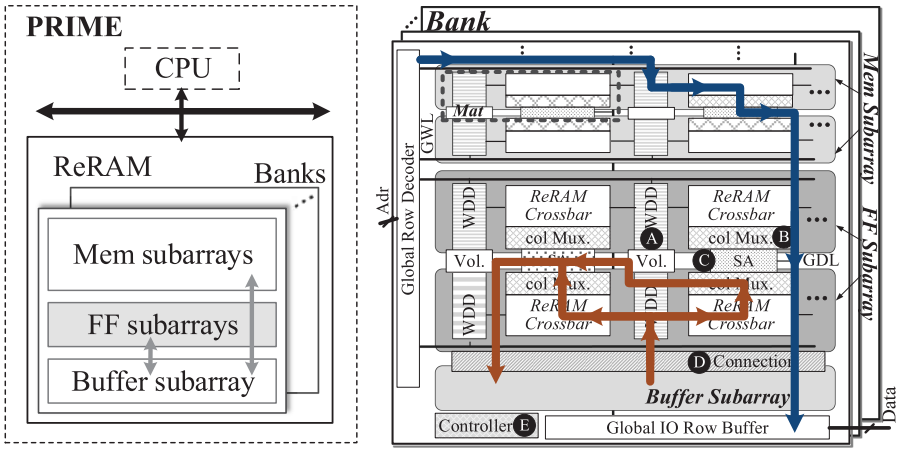
The results thereafter go through the S+H units (to temporarily store data before feeding them to ADCs) and S+A units (to accumulate data) if applicable. Finally, if multiple inputs are fetched, then a pipeline is created using IR and OR of the IMAs and tiles. To transfer data within a single memory array, a controller can be used to apply appropriate voltages to move data directly inside the memory crossbar or use read-out and write-back schemes. The additional merits of ISAAC over the generic CIM-P architecture benefits can be summarized as follows:

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [35, 193]. In contrast, some CIM-P architectures are used as main memory and therefore require a much higher endurance.
- The computations for neural networks are quite mature and can benefit from existing neural network techniques and tools.
- The computations for neural networks do not require a high precision; hence, they are more resilient against device variation.
- Data can be transferred in the crossbar using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, ISAAC also has different limitations that need to be addressed properly. Some of the limitations are:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- The architecture might suffer from a high overhead due to the need of ADC and DAC converters.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.
- In case general purpose computing is desired, the architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using neural network computations.

The architecture is evaluated analytically and compared against DaDianNao architecture (which is an ASIC design with embedded DRAM) using a suite of CNN [78, 168] and DNN workloads




(a) PRIME banks (b) PRIME's Bank architecture

Fig. 15. A Processing-in-Memory architecture for neural network computation (PRIME) [39] (a) PRIME banks organization, (b) architecture of PRIME bank.

[86, 183]. Their analytical result demonstrated that it potentially outperforms DaDianNao in terms of throughput, energy, and computational density. NEWTON introduced techniques to improve the energy efficiency by adapting ADC precision to improve the energy efficiency by adapting sub-block based ADC precision as well as an algorithm to reduce computations [136].

### 4.6 PRIME: A Processing-in-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory

PRIME was proposed in 2016 by C. Pinga et al. from University of California [39]. PRIME is a resistive RAM-based architecture that exploits data-level parallelism to perform computations for neural networks (i.e., weighted vector-matrix multiplication) using high-precision multi-level sense amplifiers and some extra logic circuits. The architecture consists of a CPU and multiple RRAM banks; each RRAM bank contains multiple memory crossbars (mem subarrays), **full function (FF),** and buffer subarrays, as shown in Figure 15(a). The CPU sends instructions to the resistive RAM banks; an instruction is either a memory operation (read/write) or a neural network computation. The memory bank performs the request without blocking the CPU, i.e., the CPU continues executing (different) instructions simultaneously. The results are returned to the CPU for further processing.

In the resistive RAM banks, the memory crossbars store data in multiple mats, while the FF and buffer subarrays serve for computation. Special subarray structures are used to enable both neural network computations and memory operations (blue blocks in Figure 15(b)) feasible. The neural network computations are mainly performed in the FF subarray, while the buffer subarray stores temporary data that needs to be processed; this enables a parallel execution between CPU and FF subarrays. Neural network computations are performed using a vector matrix multiplication between a weighted matrix stored in the FF subarray and a vector stored in the buffer subarray. Additional logic gates such as subtraction and sigmoid units are used to compute negative weights and sigmoid activation functions before the results are sensed by the multi-level sense amplifiers. To communicate between the memories, a controller can be used to apply appropriate voltages to the crossbar to move data directly inside it or use read-out and write-back schemes. On top of the generic advantages of CIM-P architectures, PRIME has the following set of unique advantages:

- The computations for neural networks are quite mature and can benefit from existing neural network techniques and tools.
- The computations for neural networks do not require a high precision; hence, they are more resilient against device variations.
- Data can be transferred in the crossbar using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes a low energy and has a small footprint.

However, the above-mentioned merits of PRIME architecture also come with their own challenges and limitations. The limitations of PRIME include:

- The architecture uses non-volatile memory as main memory, which may impact the lifetime due to limited endurance [35, 193].
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.
- In case general purpose computing is desired, the architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using neural network computations.

The architecture is synthesized using TSMC CMOS library 60 nm and modeled using NVSIM, CACTI-3D and CACTI-IO. It is evaluated using MlBench benchmarks [113] and compared against a CPU-only solution. Their comparison shows that the architecture achieves significant improvements in terms of performance and energy consumption over CPU-only-based solution.

### 4.7 ReGP: Resistive GP-SIMD

ReGP was proposed in 2016 by A. Morad et al. from Technion-Israel Institute of Technology [131]. ReGP is a RRAM memory-based architecture that exploits data parallelism by attaching a SIMD-like processing unit to the resistive memory. The architecture consists of a sequential or conventional processor, its L1 and LLC cache, shared memory array, and SIMD processor. The sequential processor executes traditional code and controls the SIMD processor in a master-slave mode. The SIMD processor executes parallel instructions on the data stored in the shared memory array.

The SIMD processor contains multiple **processing units (PUs)**, a sequencer, and a **Network on Chip (NoC)** with reduction tree. Each PU contains registers, a single bit full-adder, and a function generator to perform arithmetic and logical operations. The sequencer receives instructions from the sequential processor and assigns them to PUs. The PUs load data from the shared memory array and perform the requested operations. If required, the NoC and reduction trees are used to perform more complex functions. In addition to the general advantages of CIM-P architectures, ReGP comes with the following advantages:

- The parallelism is high due to multiple parallel processing units.
- The architecture uses non-volatile memory, hence consumes low amount of energy and has a small footprint.
- The architecture can reuse compilers, programming languages, and tools from SIMD architectures.

However, the operations within the processing units are simple; complex functions such as floating point operations can cause a high overhead. Thus, scalability is the main limitation of ReGP architecture.

The architecture is evaluated analytically against CMOS GP-SIMD [130] using the dense matrix multiplication benchmark [129].

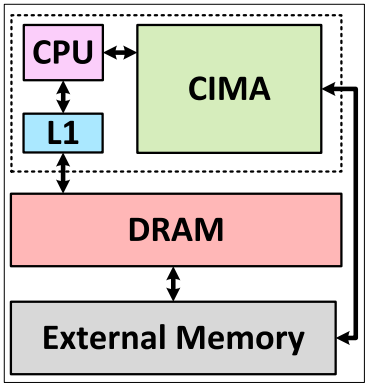



Fig. 16. Computation-in-Memory Accelerator (CIMA) [50].

### 4.8 CIMA: Computation-in-Memory Accelerator

CIMA was proposed in 2017 by H. A. Du Nguyen et al. from Delft University of Technology [50]. CIMA is a resistive-based accelerator that exploits data-level parallelism by performing computations with custom sense amplifiers. The architecture consists of a conventional processor, caches, CIM accelerator, main memory DRAM, and external memory (as shown in Figure 16). The processor fetches, decodes, and executes non-intensive memory parts of an application and off-loads the memory intensive parts to the CIM accelerator. Similarly as in Pinatubo, the CIM accelerator performs operations by activating one or multiple wordlines. The difference with Pinatubo, however, is that CIMA is used as an accelerator and has more efficient sense amplifiers. In case needed, the results are transferred back to the processor for further processing.

The CIM accelerator consists of a CIM controller, peripheral circuits (including decoder, voltage driver, and sense amplifiers), and a memristor crossbar. The CIM controller receives instructions from the processor and performs operations by sensing the current of two or more activated rows of the crossbar. Based on the type of operation, the customized sense amplifiers compute the output based on this current. CIMA uses scouting logic [203]; currently, it can only perform bitwise logical operations. However, more operations such as addition, vector-matrix multiplication, and matrix-matrix multiplication have demonstrated to be feasible [128]. The data transfer in the memory can be performed by enabling two memory rows using direct copy operations or using indirect read-out and write-back operations. In addition to the general advantages of CIM-P architectures, CIMA has the following advantages:

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [35, 193].
- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- Computations currently include only logical operations. More research is required to map complex functions on the architecture.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.

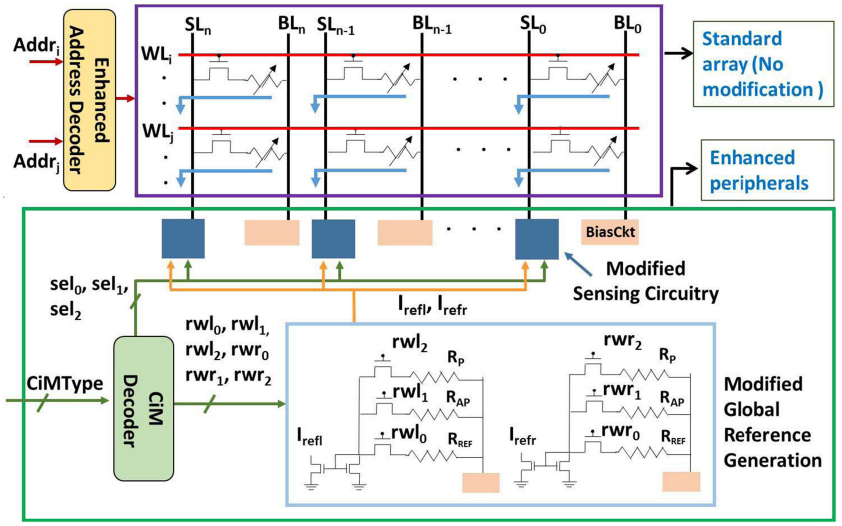



Fig. 17. Computing in Memory Spin-Transfer Torque Magnetic RAM (STT-CiM) [92].

- Additional software support (i.e., profiling and extracting memory intensive kernels) is required to maximally exploit the accelerator performance.

The architecture is evaluated against a theoretical model of a conventional multicore architecture.

### 4.9 STT-CiM: Computing in Memory Spin-Transfer Torque Magnetic RAM

STT-CiM was proposed in 2017 by S. Jain et al. from Purdue University [92]. STT-CiM is a Spin-Transfer Torque Magnetic RAM-based architecture that exploits data-level parallelism by performing computations using both modified sense amplifiers and some additional CMOS gates. The architecture consists of a conventional architecture with an STT-MRAM used as a scratch-pad memory. This scratch-pad memory is equipped with the capability to perform in-memory instructions. These instructions are sent from the main processor.

The STT-CiM contains a CiM decoder, an array of memory cells, enhanced address decoder, and modified sensing circuitry to perform computations, as shown in Figure 17. Based on the in-memory instruction, the enhanced address decoder activates one (for normal read) or multiple rows (for computations) of the memory array. The CiM decoder determines simultaneously the reference currents of the sense amplifiers. For example, in case an addition is executed, the set of logic gates for addition is enabled. The results are captured by the modified sense amplifiers. Data transfer can be performed by enabling two memory rows for direct copy operations or using the buffers and read-out operations for indirect copy operations. In addition to the general advantages of CIM-P architectures, STT-CiM has the following advantages:

- The architecture is used as an accelerator (i.e., scratch-pad memory), which has a positive impact on the endurance due to infrequent use [35, 193].
- Computations currently include both logical operations and addition.
- The data transfer may include both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

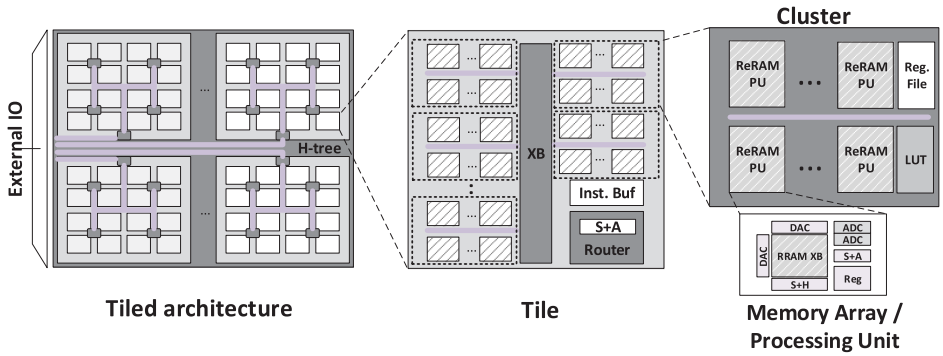



Fig. 18. Data Parallel Processor (DPP) [63].

However, it also has the following limitations:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.
- Additional software support (i.e., profiling and extracting memory intensive kernels) is required to maximally exploit the accelerator performance.

The architecture is evaluated using the STT-CiM device-to-architecture evaluation framework [92] and a set of benchmarks including string matching, text processing, low-level graphics, data compression, bio-informatic, image processing, and cryptography.

### 4.10 DPP: Data Parallel Processor

DPP was proposed in 2018 by D. Fujiki et al. from University of Michigan [63]. DPP is a RRAM-based architecture that exploits instruction and data-level parallelism by performing computations using a combination of RRAM-based dot-product operations and LUTs. The architecture consists of multiple RRAM tiles connected as an H-tree; each tile has multiple clusters and some logic units (as shown in Figure 18). Tiles and clusters form a SIMD-like processor that performs the parallel operations. The architecture is considered as a general purposed architecture, as it can perform all primitive functions such as logical, arithmetic, shift, and copy operations.

In addition to clusters, each tile has several units to support the computations, including instruction buffer, **Shift and Add (S+A)**, and router. Each cluster additionally has one or more computational units; they are S+A, **Sample and Hold (S+H)**, DAC and ADC, a LUT and register file (as shown in the right part of Figure 18). While reading from the high latency RRAM, other units are simultaneously used for processing. Therefore, the S+H is used to read data (in the form of a current) from the RRAM array and temporarily store it. Once that data is needed, it is fed to an ADC to convert the analog value to a digital value. The S+A is used to perform carry propagation in a multiple-bit addition. DAC is used to apply a digital value to the RRAM array with an appropriate control voltage. Some complex functions that cannot be realized with these units are performed using LUTs and register file in each cluster. Data transfer can be performed by enabling two memory rows for direct copy operations or using the buffers and read-out operations for indirect copy operations. In addition to the general advantages of CIM-P architectures, DPP has the following advantages:

- Computations include both logical operations and simple arithmetic operations (e.g., addition, multiplication).

- Data can be transferred using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- This architecture is claimed to be general purpose, hence it can exploit the existing instruction set, compiling techniques, and tools, as well as applications.

However, it also has the following limitations:

- The architecture uses non-volatile memory as main memory, which may impact the lifetime due to limited endurance [35, 193].
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.

The architecture potential was simulated and evaluated against CPU Intel Xeon E5-2697 using a subset of PARSEC benchmarks [19] and against GPU NVIDIA Titan XP using Rodinia benchmarks [36].

### 4.11 R-AP: Resistive RAM Automata Processor

R-AP was proposed in 2018 by J. Yu et al. from Delft University of Technology [206]. R-AP is an automata processor that exploits data-level parallelism by performing computations similarly as mentioned in Section 4.2. The working principle of R-AP is similar to the S-AP. In contrast to S-AP, R-AP uses RRAM-based STEs and routing matrice, as shown in Figure 19. In addition to the general advantages of CIM-P architectures, R-AP has the following advantages:

- The architecture is used as a read-favored accelerator, which has a positive impact on the endurance due to infrequent use [35, 193].
- Automata processing can be used to perform both logical and arithmetic operations in general.
- Data can be transferred using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- The automata processing techniques and tooling are quite mature, hence it is feasible to explore many applications using automata processing.

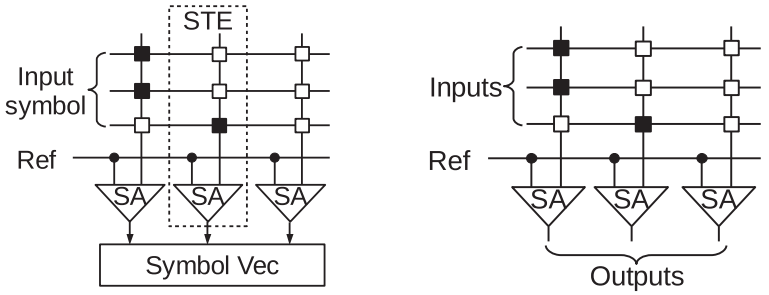
However, it also has the following limitations:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.3.
- As the sense amplifiers are complex, a tradeoff between area and bandwidth has to be made.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using automata processing.

The architecture has been validated using circuit-level simulations and evaluated against S-AP.

## 5 COMPUTATION-OUT-MEMORY (COM)

The COM class consists of architectures that perform computation using additional logic units outside the memory core but inside the memory SiP. These architectures were proposed in the past and evolved through different memory technologies ranging from embedded DRAM to emerging memory technologies such as RRAM. A large number of architectures have been proposed in this category. Table 4 shows a brief comparison among the architectures that will be explained in each subsection. On one hand, these architectures have several common advantages:




(a) Used as STEs (b) Used as routers

Fig. 19. Resistive RAM Automata Processor (R-AP) [206].

- Reduced memory bottleneck compared to conventional Von-Neumann architecture, as the computations take place close to the memory core and therefore can benefit from the on-chip memory bandwidth.
- A wide variety of high-performance functions can be implemented, as the computing takes place with mature CMOS technology.
- These architectures do not suffer from endurance requirements, especially as traditional mature memories can be used.

On the other hand, they all share the following limitations:

- The amount of parallelism is either limited by the bandwidth or the available resources, which puts much more bandwidth restrictions than what can be achieved in the CIM-A/-P class. An area tradeoff has to be made between registers and the type and amount of computing resources.
- There is an additional write overhead when the results have to be stored back into the memory. Note that the outputs are produced outside the memory core, and therefore, extra write operations would be necessary in such cases.
- Relative complex memory controllers are needed, as these architectures have to support COM instructions within the memory SiP and maintain the memory coherency with the rest of the memory hierarchy.
- Efforts are still required to modify instruction sets, compilers, and tools to support in-memory instructions.

The following subsections discuss the details of COM architectures. The discussion in this section is organized as follows: eDRAM and DRAM-based architectures are discussed in chronological order of publication date first, followed by a chronologically ordered discussion of the architectures that utilize 3D-DRAM memory. Finally, emerging technology-based architectures are discussed towards the end of the section. This ordering technique is also reflected in Table 4.

### 5.1 N3XT: Approach to Energy-efficient Abundant-data Computing

N3XT [9] was proposed in 2018 by Mohamed Sabry Aly et al. from Stanford. N3XT creates new architecture by tightly integrating computation and memory components with fine-grained and dense connectivity. The N3XT component technologies features (1) efficient logic devices fabricated at low temperature; (2) dense nonvolatile memory, and (3) fine-grained interconnect between logic and memory units. N3XT architecture exploits these technology features to enable concurrent access to a nonvolatile on chip memory as well as fast and efficient logic unit and memory

Table 4. Comparison among Architectures of COM Classes

| | Hierarchy level | Computations | | Memory Technology | Overheads | | Sneak path current | Destructive read | Required read-out* | Copy scheme | Evaluation | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Logic style | Processor type | | Periphery | Controller | | | | | Simulator | App. |
| **N3XT** | Main memory | DNN | Flexible | Hybrid | Conv. | Complex | No | No | No | Indirect | Custom-made | HTAP |
| **CoNDA** | Main memory | DNN | CPU | DRAM | Conv. | Medium | No | No | No | Indirect | Gem5 | DNN/PARSEC |
| **DIVA** | Main memory | Bool. | Vector | eDRAM | Conv. | Medium | No | No | No | Indirect | Prototype | (3) |
| **D-AP** | Accelerator | Bool. | Logical,+,x | DRAM | modif. | Simple | No | No | No | Indirect | Rambus | Automata |
| **DaDianNao** | Accelerator | NN. | NFU | eDRAM | Conv. | Simple | No | No | No | Indirect | Booksim2.0 | CNN, DNN |
| **DRAMA** | Main memory | Bool. | CGRA | 3D-DRAM | Conv. | Complex | No | No | No | Indirect | Gem5 | San Diego Vision, Parboil |
| **HBM** | Main memory | Bool. | - | 3D-DRAM | Conv. | Complex | No | No | No | Indirect | Product | No |
| **AMC** | Main memory | Bool. | Vector | 3D-DRAM | Conv. | Complex | No | No | No | Indirect | Mambo | DGEMM, DAXPY |
| **HIVE** | Main memory | Bool. | Vector | 3D-DRAM | Conv. | Complex | No | No | No | Indirect | SiNUCA | (4) |
| **ProPRAM** | Main memory | Bool. | Logical,+,x | NVM (PCM) | Modif. | Medium | No | No | No | Indirect | Multi2Sim, NVSIM | PUMA |

+: n-bit addition, Conv.: Conventional, (*): Required read-out during computations.
x: n-bit multiplication, Modif.: Modified, App.: Applications and benchmarks.
NVM: Non-Volatile Memory, Bool.: Boolean, MM.: Matrix Multiplication benchmarks.
NN.: Neural Network, NFU: Neural Functional Unit.
(1): data mining, protein pattern matching, TCP-D, MPEG-2 motion estimation.
(2): 1,024-point FFT, a 13-tap FIR filter, a $7 \times 7$ convolution, and an $8 \times 8$ DCT.
(3): scientific computing, databases, and image processing.
(4): vector search, memory reset/set operations, vector sum, matrix stencil, matrix multiplication kernels.

access circuitry. The N3XT is a vertically integrated architecture with multiple tiers as shown in Figure 20, where each tier is consisting of circuit layer or memory cell and metal layers for interconnect. The individual tiers of N3XT:

(1) Compute tier: Supports CPU cores, GPUs, domain-specific accelerators, and so on. N3XT architecture assumes the cores in the compute tier to be digital logic with local SRAM blocks, but it is able to integrate analog circuit blocks.
(2) Memory tier: A memory subsystem consisting of either memory controller, memory access circuits, or memory cells tiers.
(3) Interconnect tier: Support for **uniform memory access (UMA)** so each compute unit has the same access latency to each memory location.
(4) Cooling tier: Controls the temperature by providing different cooling solutions, such as phase-change materials and 2-D materials that can spread the heat to the edge of the chip to alleviate thermal hot spots and decrease the overall temperature.

In addition to the general advantages of COM architectures, VIRAM comes with the following advantages:

- High parallelism due to its vertically integrated tiers and UMA.
- The computations using optimized conventional processor designs, which have been optimized critically.
- The architecture uses embedded SRAM, which is a matured technology and has some advantages such as high performance [91, 101].

However, it also has the following limitations:

- The SRAM has some limitations such as leakage power, low density, and tradeoff between performance and capacity [91, 100, 101].
- The architecture faces the limitation of 3D integration such as interconnect delay.

The architecture was evaluated with domain-specific accelerators for different configurations and workload applications. For instance, the architecture achieved inference using **Deep Neural**

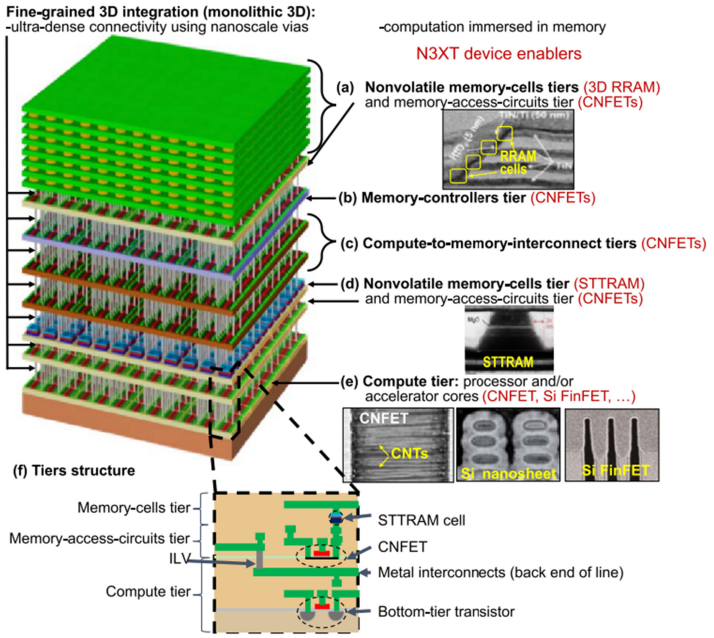



Fig. 20. N3XT architecture with 3-D integration of circuits on different tiers [9].

**Networks (DNNs)**, the architecture was able to achieve a 1,000× improvement in energy-efficiency for a reduced-precision 8-bit DNN inference acceleration.

### 5.2 CoNDA: Efficient Cache Coherence Support for Near-data Accelerators

CoNDA [30] was proposed by Amirali Boroumand et al. in 2019. CoNDA architecture (shown in Figure 21) is a mechanism that lets the **Near Data Accelerators (NDA)** optimistically start execution assuming that it has coherence permissions, without issuing any coherence messages off-chip. CoNDA executes NDA kernels optimistically while recording the memory accesses inside the NDA to get insight into the portion of shared data that is accessed by the kernel. When it finishes optimistic execution for the NDA kernel portion, it exploits the recorded information to check which coherence operations are necessary to avoid off-chip data movement for unnecessary coherence operations. If the NDA kernel portion does not need coherence operations for any of its data updates, then CoNDA commits the updates. If the NDA kernel portion actually requires any coherence operations, then CoNDA invalidates the uncommitted data updates to perform the needed coherence operations and re-executes the NDA kernel portion.

In addition to the general advantages of COM architectures, CoNDA comes with the following advantages:

- Optimistic execution enables it to avoid unnecessary coherence requests.
- It provides an interface for programmers to port application to CoNDA.

However, it also has the following limitations:

- The parallelism is limited, as it executes NDA kernels in portions.
- It needs hardware additions to support optimistic NDA execution. The needed hardware additions impose area and energy overheads.
- CoNDA needs modification of the ISA to incorporate the CoNDA specific macros.

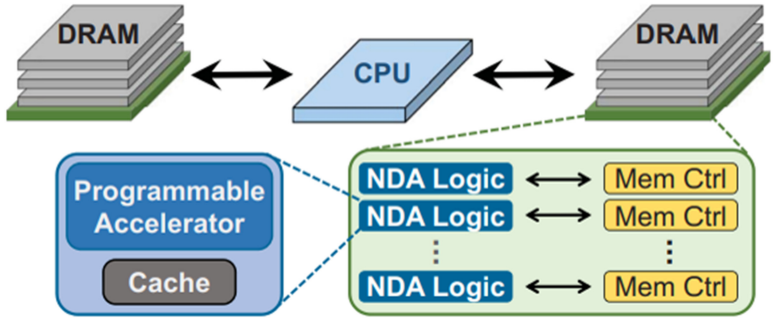



Fig. 21. Organization of CoNDA architecture [30].

CoNDA is implemented in the Gem5 simulator and it is evaluated with full-system simulation using x86 ISA. It includes a local coherence directory for the NDAs and sets the CPU coherence directory as the main point of coherence for the system. Evaluation results show that CoNDA reduces the overall data movement when compared to prior NDA coherence mechanisms with average reduction of 30.9%. CoNDA also achieved an average of 1.84× speedup. These results are achieved at the cost of 0.003% DRAM capacity overhead and 1.6% increase in NDA L1 data cache size.

### 5.3 DIVA: Data-intensive Architecture

DIVA was proposed in 2002 by J. Draper et al. from USC Information Sciences Institute [47, 48] The architecture is based on an intelligent RAM [151], computational RAM [55], or embedded DRAM [101] and exploits data-level parallelism by performing computations using vector processors. The architecture consists of a host processor, host memory interface, and multiple in-memory computing blocks as co-processors (as shown in Figure 22); the set of in-memory computing blocks (denoted as PIMs) can be accessed as a conventional memory or smart-memory co-processor. Through the host memory interface, the host processor is responsible for distributing workloads to PIMs, managing memory, and switching context between different user programs; a PIM-to-PIM interconnect provides high bandwidth links among PIMs.

A PIM architecture contains a host interface, a **PIM Routing Component (PiRC)**, and several PIM nodes. Through the host interface, memory accesses and computation-packed parcels are transferred to PIMs. The PiRC routes parcels among PIM nodes. A PIM consists of processing logic units, several megabytes of memory, PBUF, and a memory port. The processing unit in a PIM includes a scalar processor and a special unit called **At-the-Sense-Amps Processor (ASAP)**. The scalar processor is a single-issue, in-order execution, 32-bit processor with a floating-point unit. ASAP, also referred to as Wide World Unit, is used for 256-bit wide operations on data objects stored in a local memory row. PBUF in each PIM is served as local memory.

In addition to the general advantages of COM architectures, DIVA comes with the following advantages:

- The parallelism is high due to vector processing of multiple 256-bit operations concurrently.
- The architecture uses embedded DRAM, which is mature and has some advantages such as high performance, high bandwidth, low power [91, 101].

However, it also has the following limitations:

- The architecture has a complex processor design that requires overhead in controlling, communication, and programming.

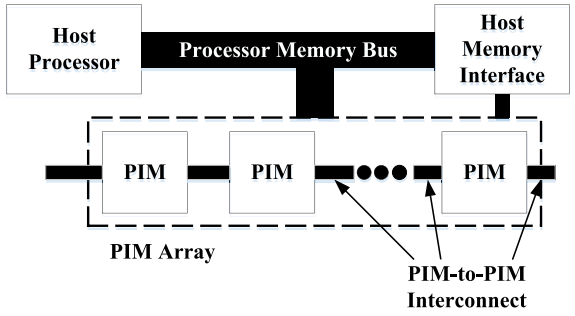



Fig. 22. Data-intensive Architecture (DIVA) [47].

- The architecture uses embedded DRAM, which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [91, 100, 101].

A DIVA chip prototype was fabricated in TSMC 0.18 um technology. Its performance was measured using a set of benchmark applications that includes scientific computing, databases, and image processing.

### 5.4 D-AP: Micron Automata Processor

D-AP was proposed in 2013 by P. Dlugosch et al. from Micron Technology [143]. D-AP is an automata processor that exploits instruction-level parallelism based on the same concept as mentioned in Section 4.2. D-AP consists of multiple STEs and a routing matrix (as shown in Figure 23). The STEs are implemented using DRAM technology. The routing matrix contains multiple programmable switches, buffers, routing lines, and cross-point connections.

Each DRAM column corresponds to an STE that stores for each state the input symbols it accepts. One input symbol is fed each cycle to all the DRAM columns simultaneously and as a result possible next states based on the current input symbol are returned. Simultaneously, the routing matrix is used to calculate the possible next states from the current active states. By AND-ing the results of both operations, the actual next states can be determined. The program terminates when all input symbols are processed. In addition to the general advantages of COM architectures, D-AP has the following advantages:

- The architecture uses DRAM technology, which has several benefits such as a high maturity, high endurance, no sneak path currents, and may benefit from existing optimizing techniques and tools.
- The automata processing techniques are quite mature and several tools exist already [6, 11, 157], hence it is feasible to explore many applications using automata processing.

However, it also has the following limitations:

- The architecture uses DRAM technology, which suffers from low performance, high energy consumption, low scalability, and a large footprint.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions, as they will have to be mapped to automata's.

The architecture is fabricated and evaluated and compared against hardware implementations of automata processing [4, 14, 70, 187, 208] in terms of configurability, throughput, and scalability. The D-AP shows improvements in comparison to other existing hardware implementations published prior to D-AP. However, S-AP (Section 4.2) and R-AP (Section 4.11) outperform D-AP.

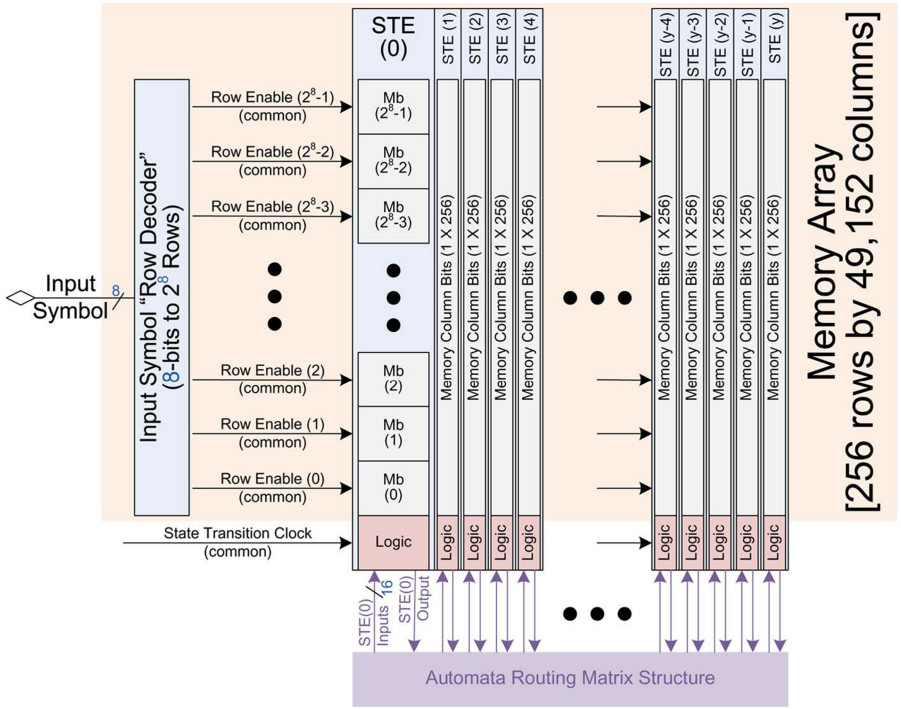



Fig. 23. DRAM Automata Processor (D-AP) [143].

### 5.5 DaDianNao: A Machine-learning Supercomputer

DaDianNao was proposed in 2014 by Y. Chen et al. from Chinese Academy of Sciences [38]. DaDianNao is eDRAM-based architecture that exploits instruction-level parallelism by performing neural computations using a **Neural Functional Unit (NFU)**. The architecture consists of multiple nodes; each node has its own interconnection to other nodes, multiple tiles, and eDRAM router that connects these tiles (as shown in the left part of Figure 24). DaDianNao is an improved version of DianNao; it exploits the interconnect network to reduce data movements.

A DaDianNao tile contains four eDRAM banks and an NFU that can perform multiplications, additions, and transfer functions of linear interpolation, depending on the type of the neural network. The eDRAM banks store input neurons that are broadcasted through a fat tree interconnect network and output neurons that are computed using NFU. After computation, the values of the output neurons are collected by the center eDRAM bank of DaDianNao. In addition to the general advantages of COM architectures, DaDianNao has the following advantages:

- Computations for neural networks are quite mature and they do not require a high precision (therefore, they are resilient against device variation), and can benefit from existing neural network techniques and tools.
- The architecture uses embedded DRAM, which is mature and has some advantages such as a high performance, high bandwidth, low power [91, 101].

However, it also has the following limitation:

- The architecture uses embedded DRAM, which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [91, 100, 101].

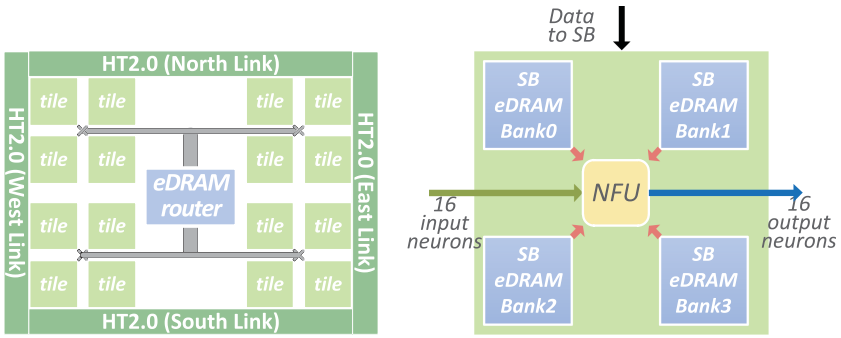



Fig. 24. A machine-learning supercomputer (DaDianNao) [38].

- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions, as they have to be mapped to neural networks.

The architecture is simulated using Booksim2.0 [42, 98] and evaluated by implementing state-of-the-art machine-learning algorithms (CNNs and DNNs) and is compared against GPU NVIDIA K20 [144].

### 5.6 DRAMA: DRAM-Accelerator

DRAMA was proposed in 2014 by A. Farmahini-Farahania et al. from University of Wisconsin-Madison [56]. DRAMA is based on 3D-stacked DRAM technology that exploits task-level parallelism by computating with **coarse-grain reconfigurable accelerators (CGRAs)**. The architecture includes a host processor, an **accelerator-enabled DRAM rank (AEDR)**, and a DRAM DIMM interface (as shown in Figure 25). The host processor sends the configuration data, address generation parameters for the data to be processed, and other kernel parameters to AEDR before triggering the kernel execution on AEDR. The host processor communicates with the AEDR through DRAM DIMM.

An AEDR includes CGRAs stacked on top of DRAM devices. The CGRAs operate on data stored in the DRAM, communicate with the DRAM through TSVs and execute kernels. A CGRA contains a grid of 32-bit **functional units (FUs)**, small distributed storages, configurable routing switches, and a memory controller. The FUs perform computational operations. The distributed storages are used to store immediate results. The configurable routing switches set the connections up between FUs and storages. They are configured at runtime by the configuration from the host processor. The memory controller is used to issue memory requests to a DRAM device. In addition to the general advantages of COM architectures, DRAMA comes with the following advantages:

- The parallelism is high due to multiple parallel cores.
- The architecture uses 3D-stacked DRAM, which has some advantages, such as a high performance, high bandwidth, and high scalability [119, 197].

However, it also has the following limitations:

- The architecture has a complex accelerator design, which has control, communication, and programming overhead.
- The architecture uses 3D-stacked DRAM, which has a high fabrication cost and high power consumption [207].

The architecture is simulated with gem5 simulator using San Diego Vision suite and Parboil benchmarks suites [179, 188] to verify its functional timing and execution. The power is estimated by

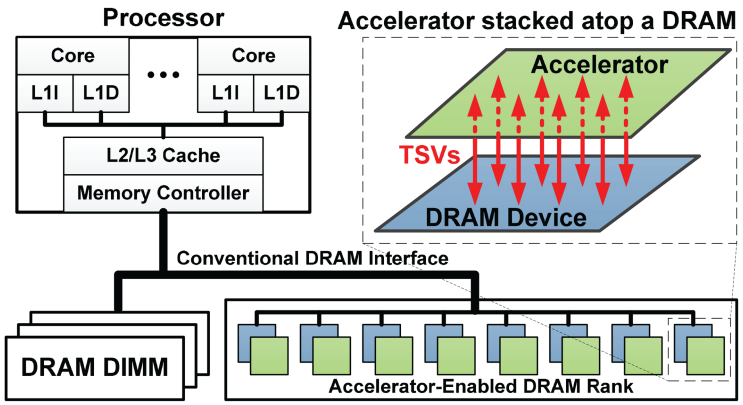



Fig. 25. An architecture for accelerated processing near memory (DRAMA) [56].

McPAT model. Components in the architecture such as 3D stacking DRAM with TSVs and CGRAs are synthesized with 32 and 40 nm to estimate their area.

### 5.7 HBM: High-Bandwidth Memory

HBM was proposed in 2015 by H. Jun et al. from SK Hynix [97, 111, 120, 176]. HBM is a heterogeneous 3D stacked DRAM platform that exploits data-level parallelism by performing near-memory computations using additional logic circuits in the memory SiP. The architecture consists of one logic die and multiple DRAM dies that are connected through TSVs and microbumps (as shown in Figure 26). The logic die performs computations on the data that is stored in the DRAM dies and tries to avoid data movements to the host processors. In addition to the general advantages of COM architectures, HBM comes with the following advantages:

- The parallelism is high due to multiple parallel cores.
- The architecture uses 3D-stacked DRAM, which has some advantages such as a high performance, high bandwidth, high scalability, and low power consumption [119, 120, 197].

However, it also has the following limitations:

- The architecture has a complex 3D stacked DRAM module, which has a control, communication, and programming overhead.
- The architecture uses 3D-stacked DRAM, which has a high fabrication cost and high power consumption [207].

The first generation of HBM increases the memory bandwidth dramatically up to 100 GB/s per stack and reduces the power consumption up to 60% [120]. Later generations of HBM such as HBM2 and HBM3 realize even bandwidth improvements up to 400 GB/s using new error correction techniques [97].

### 5.8 HMC: Hybrid Memory Cube

HMC was introduced in 2012 by J. Jeddeloh et al. from Micron [94, 153]. HMC is a 3D DRAM-based architecture that exploits data-level parallelism by performing near-memory computations inside the additional logic layer of the memory SiP. The architecture consists of a host CPU, a DRAM chip that contains a logic chip and multiple stacked DRAM dies based on TSVs, and a high-speed link between CPU and DRAM (as shown in Figure 27). The logic layer controls the DRAM as its slaves, communicates with the host CPU, and is able to perform near-memory computations. There are two architectures that are based on HMC. They are explained next.

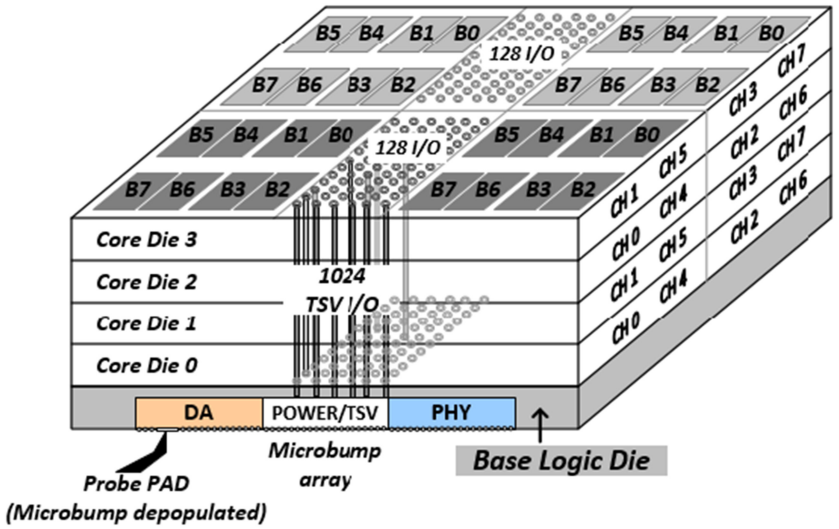



Fig. 26. High-Bandwidth Memory (HBM) [97].

*5.8.1 AMC: Active Memory Cube.* AMC was introduced in 2015 by R. Nair et al. from IBM [137]. AMC is an HMC-based architecture that performs vector instructions in the logic die of the HMC. The architecture consists of a host processor, system network, and multiple AMCs that contain a base logic layer and multiple DRAM layers (as shown in Figure 28). The host processor communicates to AMCs using the system network that is capable of transferring data with a bandwidth of 256 GB/s for read and write operations. This bandwidth is split in eight lanes (i.e., 32 GB/s) where each lane is connected to an AMC. Each AMC has an 8 GB capacity with an internal bandwidth of 320 GB/s. The architecture comes with a compiler that analyzes applications, prepares sets of code that are distributed to the AMCs, and also generates code for the interaction and communication between AMCs. The host processor handles the interaction code, while each AMC executes the code containing useful computations.

Each AMC contains multiple AMC lanes, vault controllers, and an interconnect network. Each AMC lane includes an instruction buffer, registers, control units, a load-store unit to perform conventional memory operations, and arithmetic units to perform either vector operations or long-instruction-word operations. The vault controller stores data that is loaded from DRAM layers and performs atomic operations on the data before moving it forward to the host processor or AMC lanes. Furthermore, the vault controller is also responsible for maintaining the coherence between AMC lanes. In addition to the general advantages of COM architectures, AMC comes with the following advantages:

- The parallelism is high due to multiple and concurrent processing lanes.
- The architecture uses HMC, which is mature, already commercialized, and in addition has some advantages such as a high performance, high bandwidth, low power, and high density [94, 153].

However, it also has the following limitation:

- The architecture has a complex processor design, which has a control, communication, and programming overhead.

The architecture was simulated using Mambo simulator [25] and evaluated using two computation kernels for supercomputers: DGEMM [46] and DAXPY [41].

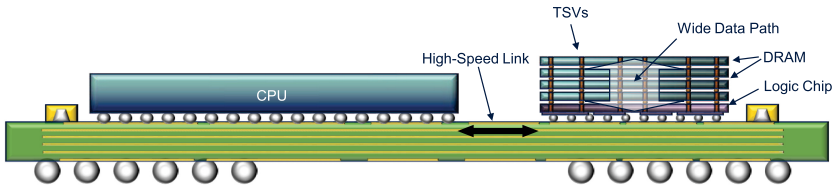



Fig. 27. Hybrid Memory Cube (HMC) [153].

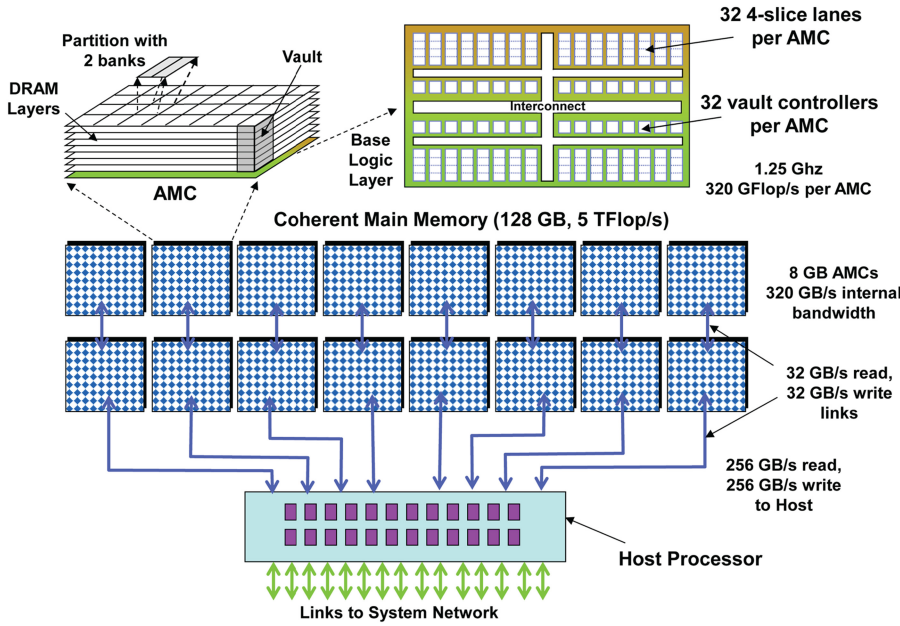



Fig. 28. Active Memory Cube (AMC) [137].

*5.8.2 HIVE: HMC Instruction Large Vector Extensions.* HIVE was proposed in 2016 by M. Alves et al. from Federal University of Rio Grande do Sul [7]. HIVE is a **Hybrid Memory Cube (HMC)**-based [94, 153] architecture that performs large vector operations inside the logic die of an HMC. The architecture consists of a host processor and an HMC module that is extended with a HIVE, as shown in Figure 29. The host processor, not shown in the figure, is a pipelined-like architecture with six stages; it fetches, decodes, renames, dispatches, executes, and commits a sequence of instruction. If an instruction fragment has to be executed using in-memory instructions, then the processor diverts the instruction fragment to the HMC module. HMC module executes the fragment and returns the result back to the processor.

HMC module consists of multiple DRAM layers, logic vaults, HIVE controller, a crossbar switch, and multiple-lane links to host processor (as shown in the left side of Figure 29). The data is stored in multiple DRAM layers and retrieved by the HIVE. The HIVE controller contains a register bank, functional units, and a HIVE sequencer (as shown in the bottom right of Figure 29). The logic vaults contains a vault controller, write and read buffer, and a DRAM sequencer (as shown in the top right of Figure 29). Once the HIVE sequencer receives an instruction, it locks the involved memory address space; if the memory has already been locked, then the requested instruction returns a fail status to processor; otherwise, a memory synchronization occurs by flushing related

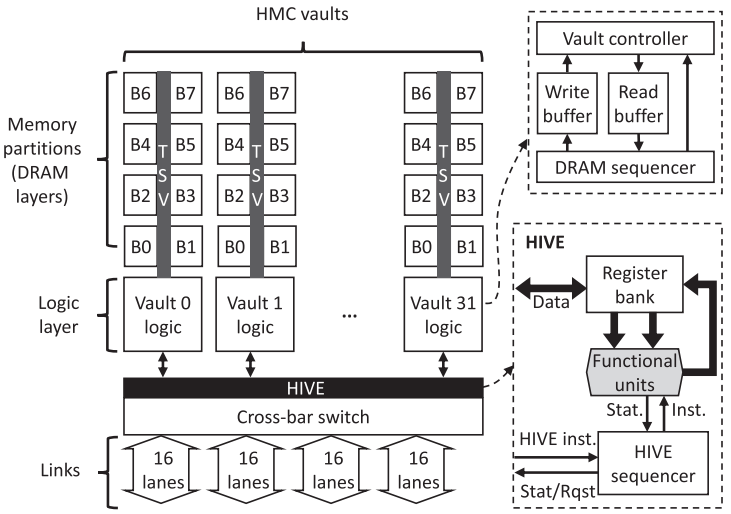



Fig. 29. HMC Instruction Large Vector Extensions (HIVE) [7].

cache data into DRAM. The logic vaults and HIVE subsequently execute the instructions by reading data to read buffers and register bank, performing operations using functional units, and (optional) storing into memory using write buffers. The operations in HIVE are based on vector operations that operate on 8 KB of data at a time executed by the 32 logic vaults and HIVE functional units. As the amount of data is large, a DRAM sequencer and HIVE sequencer schedule these operations accordingly. The results can be collected in register banks and sent back to the host processor through the crossbar switch and links. In addition to the general advantages of COM architectures, HIVE comes with the following advantages:

- The parallelism is high due to vector processing on 8 KB of data.
- The architecture uses HMC, which is mature, commercialized, and has some advantages such as high performance, high bandwidth, low power, high density [94, 153].

However, HIVE architecture has a complex HMC module, which has a control, communication, and programming overhead, which is its main limitation.

The architecture is simulated using SiNUCA [8] and evaluated using some integer (vector search and memory reset/set operations) and floating-point (vector sum, matrix stencil, and matrix multiplication) kernels against three baseline platforms; both HIVE and baseline platforms are based on the Intel Atom processor. Like HIVE, the three baseline platforms also have additional processing capacities; for the baseline platforms they are as follows: (1) HMC instructions using HMC 2.0 memory [127] (HMC+HMC), (2) 128-bit SSE instructions with DDR-3 1,333 modules (SSE+DDR), and (3) 128-bit SSE instructions with HMC 2.0 (SSE+HMC).

### 5.9 ProPRAM: A Near Data Computing Architecture Based on Non-volatile Memory

ProPRAM was proposed in 2015 by Y. Wang et al. from Chinese Academy of Sciences [195]. ProPRAM is a non-volatile memory (i.e., PCM)-based architecture that exploits data-level parallelism by performing computations using computational resources in the peripheral circuitry. The architecture consists of a host processor with a modified instruction set and a non-volatile memory module with its peripheral circuits transparent to the host processor. The host processor sends instructions to the non-volatile memory module in a similar manner as normal read/write

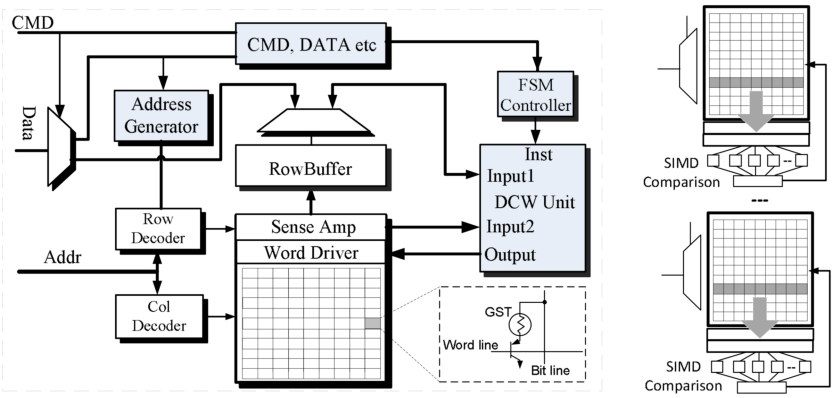



Fig. 30. A near data computing architecture based on non-volatile memory (ProPRAM) [195].

instruction but with a modified instruction set. The non-volatile memory receives these instructions and reuses the computational resources in the memory module to perform the computations on the data stored in the non-volatile memory.

The non-volatile memory module contains a matrix of memory cells, a conventional peripheral circuit consisting of sense amplifiers, word drivers, row buffers, col decoders, and a specific peripheral circuit consisting of components related to non-volatile memory instructions such as **Data-Comparison Write (DCW)**, **DCW Inversion (DCWI)**, and Flip-n-Write (as shown in Figure 30). The non-volatile memory-related peripheral circuit is capable of performing comparisons, additions, and logic operations. Therefore, an appropriate addressable scheme can be used to direct operands (data from non-volatile memory) to these computational units. For example, the DCW unit in Figure 30 is able to compute different operations using the first input from the row buffer and the second input from the sense amplifiers; the output is stored back into the memory. In addition to the general advantages of COM architectures, ProPRAM comes with the following advantages:

- The parallelism is high due to multiple parallel processing units.
- The architecture exploits existing computational resources, which reduces area and power consumption.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- The parallelism is low due to limited available computing resources in the memory peripheral circuitry.
- The architecture has programming and compiling overhead to make the computation units transparent to the host processor.

The architecture is simulated using Multi2Sim [186] and NVSIM [45]; it is evaluated using PUMA benchmark suite [57, 81, 110, 148, 182] and has been compared against a multicore computing system.

## 6 PROSPECT AND CHALLENGES

Memories occupy most of the silicon area, and their demand is growing continuously, especially for data-centric applications. In traditional Von-Neumann architectures wherein memory and

processor units are separated, various memory technologies are organized in a certain hierarchical order, depending on their response time and capacity characteristics. However, in such architectures, a large amount of data movement happens during the execution of computational tasks that incurs significant costs in terms of energy and latency. **Computing-In-Memory (CIM)** architectures, where computational tasks are performed within the memory itself, eliminate this unnecessary data movement. As a consequence, these architectures address the memory bottleneck issue as well as provide high data parallelism.

Computing with charge-based devices such as SRAM, DRAM, and Flash have an advantage of technology maturity, nevertheless they are facing leakage-related challenges. However, non-charge-based memristive devices such as RRAM, PCRAM, and STT-MRAM have low leakage due to their non-volatile nature of storage [147, 158, 161]. Moreover, the nature of their state dynamics and small size make them more suitable for multiply-accumulate operations, which is desirable for the CIM architecture. But these technologies are facing severe problems due to various device-level non-idealities such as resistance variations, interconnect parameters, endurance/lifetime, sneak paths, imperfect write behavior and so on, which lead to errors and/or accuracy-loss in the computation [33, 132, 133, 139, 160]. Moreover, there are circuit-level non-idealities and (potentially) severe area overhead in the periphery, which is due to the need for digital-to-analog converters, analog-to-digital converters, or sense amplifiers in CIM architecture.

It is important to discuss that memory-centric computing paradigm is not only limited to academia and research institutes, but industries (from startup to well-established industries) are also paying significant attention to memory-centric architectures. For instance, UPMEM, a fabless startup semiconductor company, has developed a DRAM-based processing-in-memory accelerator for data-intensive operations [142]. Similarly, several SRAM-based computing platforms have been developed by various industries such as Cerebras [93], CEA-LETI [142]. These computing engines are useful for data-intensive application segments such as DNNs [93]. It is also important to mention that the systolic architecture adopted by TPU-V3 [107] from Google [29], 3D-stacked memory [2, 82], and CoNDA [30] can be classified as near-memory computing architectures [173, 174], as it minimizes data movement by storing the parameters in a grid during computation. GRIM-Filter [103] is another near-memory architecture that integrates computation within a logic layer stacked under memory layers to perform **processing-in-memory (PIM)** for seed location filtering in DNA read mapping. All these developments are a big step into the realization and commercialization of memory-centric computing engines for data-intensive application segments.

The growth of data-intensive applications such as **deep neural networks (DNN)**, data-centric server applications, image processing, and database query applications, has created a demand for high-performance and efficient hardware architectures, especially for the edge devices. For that purpose, CIM-based computation architectures using emerging non-charged-based resistive devices have the potential to improve performance as well as energy/area efficiency compared to the traditional computing systems, which can play an important role in defining the future of computing. In such architecture, the current-mode of compute function, i.e., multiply and accumulate operation, is actually the approximation of the given applications whose accuracy can be improved with the maturity of the manufacturing process and material improvements.

## 7 CONCLUSION

This article presented a comprehensive survey of memory-centric architectures classified into different classes. The article is based on a classification framework that we have published in earlier work, and, from there, we have focused on a survey of two classification metrics (i.e., computation location and memory technologies) and three main related classes, namely, CIM-A, CIM-P, and COM. Then, nearly 30 selected architectures were presented, evaluated, and compared

qualitatively. The work shows that a potential architecture does not only require to be memory-bottleneck-free, but also energy- and area-efficient. This can only be achieved through the joint effort of both architectural improvement and technology development. This work also demonstrated that architectures are not changing dramatically, but gradually with small changes and technology developments. Indeed, emerging memory technologies play a vital role in overcoming the energy, performance, and memory bandwidth challenges of Von-Neumann architecture. However, one needs to revisit the architectural challenges of memory-centric architectures to harness the full potential of emerging memory technologies.

## ACKNOWLEDGMENTS

The authors would like to thank Xie Lei and Razvan Nane for a lot of intensive and useful discussions.

## REFERENCES

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2017. Compute caches. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*. IEEE, 481–492.

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *42nd Annual International Symposium on Computer Architecture*. 105–117.

[3] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *International Symposium on Computer Architecture (ISCA'15)*. 336–348.

[4] Alfred V. Aho and Margaret J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Commun. ACM* 18, 6 (1975), 333–340.

[5] Mustafa F. Ali, Akhilesh Jaiswal, and Kaushik Roy. 2019. In-memory low-cost bit-serial addition using commodity DRAM technology. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 67, 1 (2019), 155–165.

[6] Nourah A. Almubarak, Anwar Alshammeri, and Imtiaz Ahmad. 2016. Automata processor architecture and applications: A survey. *Int. J. Grid Distrib. Comput.* 9, 4 (2016), 53–66.

[7] Marco A. Z. Alves, Matthias Diener, Paulo C. Santos, and Luigi Carro. 2016. Large vector extensions inside the HMC. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1249–1254.

[8] Marco Antonio Zanata Alves, Carlos Villavieja, Matthias Diener, Francis Birck Moreira, and Philippe Olivier Alexandre Navaux. 2015. SiNUCA: A validated micro-architecture simulator. In *IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conf on Embedded Software and Systems*. 605–610.

[9] Mohamed M. Sabry Aly, Tony F. Wu, Andrew Bartolo, Yash H. Malviya, William Hwang, Gage Hills, Igor Markov, Mary Wootters, Max M. Shulaker, H.-S. Philip Wong, et al. 2018. The N3XT approach to energy-efficient abundant-data computing. *Proc. IEEE* 107, 1 (2018), 19–48.

[10] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *24th International Workshop on Logic & Synthesis (IWLS'15)*.

[11] Kevin Angstadt, Jack Wadden, Vinh Dang, Ted Xie, Dan Kramp, Westley Weimer, Mircea Stan, and Kevin Skadron. 2018. MNCaRT: An open-source, multi-architecture automata-processing research and execution ecosystem. *IEEE Comput. Archit. Lett.* 17, 1 (2018), 84–87.

[12] Scott Beamer, Krste Asanović, and David Patterson. 2013. Direction-optimizing breadth-first search. *Sci. Program.* 21, 3-4 (2013), 137–148.

[13] Michela Becchi, Mark Franklin, and Patrick Crowley. 2008. A workload for evaluating deep packet inspection architectures. In *IEEE International Symposium on Workload Characterization*. IEEE, 79–89.

[14] Michela Becchi, Charlie Wiseman, and Patrick Crowley. 2009. Evaluating regular expression matching engines on network and general purpose processors. In *5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 30–39.

[15] Rotem Ben-Hur, Ronny Ronen, Ameer Haj-Ali, Debjyoti Bhattacharjee, Adi Eliahu, Natan Peled, and Shahar Kvatinsky. 2019. Simpler magic: Synthesis and mapping of in-memory logic executed in a single row to improve throughput. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 39, 10 (2019), 2434–2447.

[16] Christopher Bengel, Johannes Mohr, Stefan Wiefels, Abhairaj Singh, Anteneh Gebregiorgis, Rajendra Bishnoi, Said Hamdioui, Rainer Waser, Dirk Wouters, and Stephan Menzel. 2022. Reliability aspects of binary vector-matrix-multiplications using ReRAM devices. *Neuromorph. Comput. Eng.* 2, 3 (2022), 4001–4020.

[17] Debjyoti Bhattacharjee, Rajeswari Devadoss, and Anupam Chattopadhyay. 2017. ReVAMP: ReRAM based VLIW architecture for in-memory computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 782–787.

[18] Sabpreet Bhatti, Rachid Sbiaa, Atsufumi Hirohata, Hideo Ohno, Shunsuke Fukami, and S. N. Piramanayagam. 2017. Spintronics based random access memory: A review. *Mater. Today* 20, 9 (2017), 530–548.

[19] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *17th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 72–81.

[20] Rajendra Bishnoi, Mojtaba Ebrahimi, Fabian Oboril, and Mehdi B. Tahoori. 2014. Asynchronous asymmetrical write termination (AAWT) for a low power STT-MRAM. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*. IEEE, 1–6.

[21] Rajendra Bishnoi, Mojtaba Ebrahimi, Fabian Oboril, and Mehdi B. Tahoori. 2016. Improving write performance for STT-MRAM. *IEEE Trans. Magnet.* 52, 8 (2016), 1–11.

[22] Rajendra Bishnoi, Fabian Oboril, Mojtaba Ebrahimi, and Mehdi B. Tahoori. 2015. Self-timed read and write operations in STT-MRAM. *IEEE Trans. Very Large Scale Integ. Syst.* 24, 5 (2015), 1783–1793.

[23] Rajendra Bishnoi, Lizhou Wu, Moritz Fieback, Christopher Münch, Sarath Mohanachandran Nair, Mehdi Tahoori, Ying Wang, Huawei Li, and Said Hamdioui. 2020. Special session-Emerging memristor based memory and CIM architecture: Test, repair and yield analysis. In *VLSI Test Symposium (VTS'20)*. IEEE, 1–10.

[24] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. 2007. PRESENT: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 450–466.

[25] Patrick Bohrer, James Peterson, Mootaz Elnozahy, Ram Rajamony, Ahmed Gheith, Ron Rockhold, Charles Lefurgy, Hazim Shafi, Tarun Nakra, Rick Simpson, et al. 2004. Mambo: A full system simulator for the PowerPC architecture. *ACM SIGMETRICS Perform. Eval. Rev.* 31, 4 (2004), 8–12.

[26] Giancarlo Bongiovanni and Fabrizio Luccio. 1980. Maintaining sorted files in a magnetic bubble memory. *IEEE Trans. Comput.* 10 (1980), 855–863.

[27] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and R. Stanley Williams. 2010. Memristive switches enable stateful logic operations via material implication. *Nature* 464, 7290 (2010), 873–876.

[28] S. Borkar. 1999. Design challenges of technology scaling. *IEEE Micro* 19, 4 (July 1999), 23–29. DOI: https://doi.org/10.1109/40.782564

[29] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, et al. 2018. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. 316–331.

[30] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, et al. 2019. CoNDA: Efficient cache coherence support for near-data accelerators. In *46th International Symposium on Computer Architecture*. 629–642.

[31] Rafmag Cabrera, Emmanuelle Merced, and Nelson Sepúlveda. 2013. A micro-electro-mechanical memory based on the structural phase transition of VO2. *Phys. Stat. Solid. (a)* 210, 9 (2013), 1704–1711.

[32] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 52.

[33] Indranil Chakraborty, Deboleena Roy, and Kaushik Roy. 2018. Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars. *IEEE Trans. Emerg. Topics Computat. Intell.* 2, 5 (2018), 335–344.

[34] Chee-Yong Chan and Yannis E. Ioannidis. 1998. Bitmap index design and evaluation. In *ACM SIGMOD Record*, Vol. 27. ACM, 355–366.

[35] Meng-Fan Chang, Ching-Hao Chuang, Min-Ping Chen, Lai-Fu Chen, Hiroyuki Yamauchi, Pi-Feng Chiu, and Shyh-Shyuan Sheu. 2012. Endurance-aware circuit designs of nonvolatile logic and nonvolatile SRAM using resistive memory (memristor) device. In *17th Asia and South Pacific Design Automation Conference (ASP-DAC'12)*. IEEE, 329–334.

[36] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization*. IEEE, 44–54.

[37] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, et al. 2010. Advances and future prospects of spin-transfer torque random access memory. *IEEE Trans. Magnet.* 46, 6 (2010), 1873–1878.

[38] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. DaDianNao: A machine-learning supercomputer. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[39] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[40] Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. 2010. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? In *43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 225–236.

[41] LAPACK Contributors. 2013. DAXPY subroutine, Linear Algebra Package, LAPACK 3.5.0. (2013). Retrieved from http://www.netlib.org/lapack#_lapack_version_3_5_0.

[42] William James Dally and Brian Patrick Towles. 2004. *Principles and Practices of Interconnection Networks*. Elsevier.

[43] M. Di Ventra and Y. V. Pershin. 2013. Memcomputing: A computing paradigm to store and process information on the same physical platform. *Nat. Phys.* 9, 4 (Apr. 2013), 200–202. DOI: https://doi.org/10.1038/nphys2566

[44] Sumit Diware, Anteneh Gebregiorgis, Rajiv V. Joshi, Said Hamdioui, and Rajendra Bishnoi. 2021. Unbalanced bit-slicing scheme for accurate memristor-based neural network architecture. In *IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS'21)*. 1–4.

[45] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. 2012. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 31, 7 (2012), 994–1007.

[46] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. 2003. The LINPACK benchmark: Past, present and future. *Concurr. Computat.: Pract. Exper.* 15, 9 (2003), 803–820.

[47] Jaffrey Draper, J. Tim Barrett, Jeff Sondeen, Sumit Mediratta, Chang Woo Kang, Ihn Kim, and Gokhan Daglikoca. 2005. A prototype processing-in-memory (PIM) chip for the data-intensive architecture (DIVA) system. *J. VLSI Sig. Process. Syst. Sig. Image Vid. Technol.* 40, 1 (2005), 73–84.

[48] Jeff Draper, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, Chun Chen, Chang Woo Kang, et al. 2002. The architecture of the DIVA processing-in-memory chip. In *16th International Conference on Supercomputing*. ACM, 14–25.

[49] H. A. Du Nguyen, Lei Xie, Jintao Yu, Mottaqiallah Taouil, and Said Hamdioui. 2017. Interconnect networks for resistive computing architectures. In *12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'17)*. IEEE, 1–6.

[50] H. A. Du Nguyen, Jintao Yu, Lei Xie, Mottaqiallah Taouil, Said Hamdioui, and Dietmar Fey. 2017. Memristive devices for computing: Beyond CMOS and beyond Von Neumann. In *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC'17)*. IEEE, 1–10.

[51] Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, Razvan Nane, Said Hamdioui, and Koen Bertels. 2017. On the implementation of computation-in-memory parallel adder. *IEEE Trans. Very Large Scale Integ. Syst.* 25, 8 (2017), 2206–2219.

[52] P. Dudek and S. J. Carey. 2006. General-purpose 128/spl times/128 SIMD processor array with integrated image sensor. *Electron. Lett.* 42, 12 (2006), 678–679.

[53] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, and Reetuparna Das. 2018. Neural cache: Bit-serial in-cache acceleration of deep neural networks. *arXiv preprint arXiv:1805.03718* (2018).

[54] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. 1997. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro* 17, 5 (1997), 12–19.

[55] Duncan G. Elliott, W. Martin Snelgrove, and Michael Stumm. 1992. Computational RAM: A memory-SIMD hybrid and its application to DSP. In *IEEE Custom Integrated Circuits Conference*. IEEE, 30–6.

[56] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. DRAMA: An architecture for accelerated processing near memory. *IEEE Comput. Architect. Lett.* 14, 1 (2015), 26–29.

[57] P. Ferragina and G. Navarro. 2005. Pizza and Chili repository. Retrieved from http://pizzachili.dcc.uchile.cl/texts.html.

[58] Moritz Fieback et al. 2020. Testing scouting logic-based computation-in-memory architectures. In *IEEE European Test Symposium (ETS'20)*.

[59] Moritz Fieback, Guilherme Cardoso Medeiros, Lizhou Wu, Hassen Aziza, Rajendra Bishnoi, Mottaqiallah Taouil, and Said Hamdioui. 2022. Defects, fault modeling, and test development framework for RRAMs. *ACM J. Emerg. Technol. Comput. Syst.* 18, 3 (2022), 1–26.

[60] Tim Finkbeiner, Glen Hush, Troy Larsen, Perry Lea, John Leidel, and Troy Manning. 2017. In-memory intelligence. *IEEE Micro* 37, 4 (2017), 30–38.

[61] Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. 2008. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel White Pap.* 19 (2008), 20.

[62] Adi Fuchs and David Wentzlaff. 2019. The accelerator wall: Limits of chip specialization. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE, 1–14.

[63] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. 2018. In-memory data parallel processor. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1–14.

[64] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. 2019. Duality cache for data parallel acceleration. In *46th International Symposium on Computer Architecture*. 397–410.
[65] Pierre-Emmanuel Gaillardon, Luca Amar, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. 2016. The programmable logic-in-memory (PLiM) computer. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'16)*. IEEE, 427–432.
[66] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. 2019. ComputeDRAM: In-memory compute using off-the-shelf DRAMs. In *52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 100–113.
[67] Anteneh Gebregiorgis, Rajendra Bishnoi, and Mehdi B. Tahoori. 2018. A comprehensive reliability analysis framework for NTC caches: A system to device approach. *Trans. Comput.-aid. Des. Integ. Circ. Syst.* 38, 3 (2018), 439–452.
[68] Anteneh Gebregiorgis, Rajendra Bishnoi, and Mehdi B. Tahoori. 2018. Spintronic normally-off heterogeneous system-on-chip design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. 113–118.
[69] Simcha Gochman, Avi Mendelson, Alon Naveh, and Efraim Rotem. 2006. Introduction to Intel Core Duo processor architecture. *Intel Technol. J.* 10, 2 (2006).
[70] Mario Góngora-Blandón and Miguel Vargas-Lombardo. 2012. State of the art for string analysis and pattern search using CPU and GPU based programming. *J. Inf. Secur.* 3, 04 (2012), 314.
[71] Jonathan E. Green, Jang Wook Choi, Akram Boukai, Yuri Bunimovich, Ezekiel Johnston-Halperin, Erica DeIonno, Yi Luo, Bonnie A. Sheriff, Ke Xu, Young Shik Shin, et al. 2007. A 160-kilobit molecular electronic memory patterned at 10 11 bits per square centimetre. *Nature* 445, 7126 (2007), 414.
[72] Ameer Haj-Ali, Rotem Ben-Hur, Nimrod Wald, and Shahar Kvatinsky. 2018. Efficient algorithms for in-memory fixed point multiplication using MAGIC. In *IEEE International Symposium on Circuits and Systems (ISCAS'18)*. IEEE, 1–5.
[73] Beat Halg. 1990. On a micro-electro-mechanical nonvolatile memory cell. *IEEE Trans. Electron. Dev.* 37, 10 (1990), 2230–2236.
[74] Said Hamdioui, Koenraad Laurent Maria Bertels, and Mottaqiallah Taouil. 2017. Computing device for "big data" applications using memristors. (Nov. 21 2017). US Patent 9,824,753.
[75] Said Hamdioui, Shahar Kvatinsky, Gert Cauwenberghs, Lei Xie, Nimrod Wald, Siddharth Joshi, Hesham Mostafa Elsayed, Henk Corporaal, and Koen Bertels. 2017. Memristor for computing: Myth or reality? In *Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 722–731.
[76] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1718–1725.
[77] Adib Haron, Jintao Yu, Razvan Nane, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2016. Parallel matrix multiplication on memristor-based computation-in-memory architecture. In *International Conference on High Performance Computing & Simulation (HPCS'16)*. IEEE, 759–766.
[78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision*. 1026–1034.
[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[80] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture: A Quantitative Approach*. Elsevier.
[81] R. Nigel Horspool. 1980. Practical fast searching in strings. *Softw. Pract. Exper.* 10, 6 (1980), 501–506.
[82] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler. 2016. Transparent offloading and mapping (TOM) enabling programmer-transparent near-data processing in GPU systems. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 204–216.
[83] Rotem Ben Hur and Shahar Kvatinsky. 2016. Memristive memory processing unit (MPU) controller for in-memory processing. In *IEEE International Conference on the Science of Electrical Engineering (ICSE'16)*. IEEE, 1–5.
[84] Rotem Ben Hur, Nimrod Wald, Nishil Talati, and Shahar Kvatinsky. 2017. SIMPLE MAGIC: Synthesis and in-memory mapping of logic execution for memristor-aided logic. In *IEEE/ACM International Conference on Computer-aided Design (ICCAD'17)*. IEEE, 225–232.
[85] Kai Hwang and Naresh Jotwani. 2016. *Advanced Computer Architecture, 3e*. McGraw-Hill Education.
[86] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F. Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V. Francés-Víllora. 2015. Simplified spiking neural network architecture and STDP learning algorithm applied to image classification. *EURASIP J. Image Vid. Process.* 2015, 1 (2015), 4.
[87] Mohsen Imani, Saransh Gupta, and Tajana Rosing. 2017. Ultra-efficient processing in-memory for data intensive applications. In *54th Annual Design Automation Conference*. ACM, 6.
[88] Intel. 2018. 6th Generation Intel Core Processor Family Datasheet. Retrieved from http://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-datasheet-vol-1.html.
[89] IRDS. 2020. International roadmap for devices and systems. IRDS.
[90] ITRS. 2010. ITRS ERD report. Retrieved from http://www.itrs.net.

[91] Subramanian S. Iyer and Howard L. Kalter. 1999. Embedded DRAM technology: Opportunities and challenges. *IEEE Spectrum* 36, 4 (1999), 56–64.
[92] Shubham Jain, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2017. Computing in memory with spin-transfer torque magnetic RAM. *arXiv preprint arXiv:1703.02118* (2017).
[93] Michael James, Marvin Tom, Patrick Groeneveld, and Vladimir Kibardin. 2020. ISPD 2020 physical mapping of neural networks on a wafer-scale deep learning accelerator. In *International Symposium on Physical Design*. 145–149.
[94] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Symposium on VLSI Technology (VLSIT'12)*. IEEE, 87–88.
[95] Mario Jino and Jane W. S. Liu. 1978. Intelligent magnetic bubble memories. In *5th Annual Symposium on Computer Architecture*. 166–174.
[96] Mike Johnson and Mike Johnson. 1991. *Superscalar Microprocessor Design*. Vol. 77. Prentice Hall, Englewood Cliffs, NJ.
[97] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (high bandwidth memory) DRAM technology and architecture. In *IEEE International Memory Workshop (IMW'17)*. IEEE, 1–4.
[98] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. 2012. Orion 2.0: A power-area simulator for interconnection networks. *IEEE Trans. Very Large Scale Integ. Syst.* 20, 1 (2012), 191–196.
[99] Yi Kang, Wei Huang, Seung-Moon Yoo, Diana Keen, Zhenzhou Ge, Vinh Lam, Pratap Pattnaik, and Josep Torrellas. 2012. FlexRAM: Toward an advanced intelligent memory system. In *IEEE 30th International Conference on Computer Design (ICCD'12)*. IEEE, 5–14.
[100] Doris Keitel-Schulz and Norbert Wehn. 1998. Issues in embedded DRAM development and applications. In *11th International Symposium on System Synthesis*. IEEE Computer Society, 23–31.
[101] Doris Keitel-Schulz and Norbert Wehn. 2001. Embedded DRAM development: Technology, physical design, and application issues. *IEEE Des. Test Comput.* 18, 3 (2001), 7–15.
[102] J. S. Kim, J. Gómez-Luna, and O. Mutlu. Processing-in-memory: A workload-driven perspective. *IBM Journal of Research and Development* 63, 6 (2019), 1–20.
[103] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genom.* 19, 2 (2018), 23–40.
[104] Kyosun Kim, Sangho Shin, and Sung-Mo Kang. 2011. Stateful logic pipeline architecture. In *IEEE International Symposium of Circuits and Systems (ISCAS'11)*. IEEE, 2497–2500.
[105] Christoforos E. Kozyrakis, Stylianos Perissakis, David Patterson, Thomas Anderson, Krste Asanovic, Neal Cardwell, Richard Fromm, Jason Golbus, Benjamin Gribstad, Kimberly Keeton, et al. 1997. Scalable processors in the billion-transistor era: IRAM. *Computer* 30, 9 (1997), 75–78.
[106] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Conference on Advances in Neural Information Processing Systems*. 1097–1105.
[107] Sameer Kumar, Victor Bitorff, Dehao Chen, Chiachen Chou, Blake Hechtman, HyoukJoong Lee, Naveen Kumar, Peter Mattson, Shibo Wang, Tao Wang, et al. 2019. Scale MLPerf-0.6 models on Google TPU-v3 Pods. *arXiv preprint arXiv:1909.09756* (2019).
[108] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. 2014. MAGIC–memristor-aided logic. *IEEE Trans. Circ. Syst. II: Expr. Briefs* 61, 11 (2014), 895–899.
[109] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. 2014. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. *IEEE Trans. Very Large Scale Integ. Syst.* 22, 10 (2014), 2054–2066.
[110] MIT Lincoln Laboratory. DARPA Intrusion Detection Data Sets. (n.d.).
[111] Jong Chern Lee, Jihwan Kim, Kyung Whan Kim, Young Jun Ku, Dae Suk Kim, Chunseok Jeong, Tae Sik Yun, Hongjung Kim, Ho Sung Cho, Yeon Ok Kim, et al. 2016. 18.3 A 1.2 V 64Gb 8-channel 256GB/s HBM DRAM with peripheral-base-die architecture and small-swing technique on heavy load interface. In *IEEE International Solid-State Circuits Conference (ISSCC'16)*. IEEE, 318–319.
[112] Eero Lehtonen, Jussi H. Poikonen, and Mika Laiho. 2014. Memristive stateful logic. In *Memristor Networks*. Springer, 603–623.
[113] Friedrich Leisch and Evgenia Dimitriadou. 2010. Machine learning benchmark problems. *R Package, mlbench* 2, 1 (2010). https://cran.microsoft.com/snapshot/2015-02-06/web/packages/mlbench/mlbench.pdf.
[114] Chao Li, Wendy Fan, Bo Lei, Daihua Zhang, Song Han, Tao Tang, Xiaolei Liu, Zuqin Liu, Sylvia Asano, Meyya Meyyappan, et al. 2004. Multilevel memory based on molecular devices. *Appl. Phys. Lett.* 84, 11 (2004), 1949–1951.
[115] Chao Li, Daihua Zhang, Xiaolei Liu, Song Han, Tao Tang, Chongwu Zhou, Wendy Fan, Jessica Koehne, Jie Han, Meyya Meyyappan, et al. 2003. Fabrication approach for molecular memory arrays. *Appl. Phys. Lett.* 82, 4 (2003), 645–647.

[116] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 288–301.

[117] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6.

[118] E. Linn, R. Rosezin, S. Tappertzhofen, R. Waser, et al. 2012. Beyond Von Neumann–logic operations in passive crossbar arrays alongside memory operations. *Nanotechnology* 23, 30 (2012), 305205.

[119] Gabriel H. Loh. 2008. 3D-stacked memory architectures for multi-core processors. In *ACM SIGARCH Computer Architecture News*, Vol. 36. IEEE Computer Society, 453–464.

[120] Joe Macri. 2015. AMD's next generation GPU and high bandwidth memory architecture: FURY. In *IEEE Hot Chips Symposium (HCS'15)*. IEEE, 1–26.

[121] Ariel Maislos et al. 2011. A new era in embedded flash memory. In *Flash Memory Summit*.

[122] Jack A. Mandelman, Robert H. Dennard, Gary B. Bronner, John K. DeBrosse, Rama Divakaruni, Yujun Li, and Carl J. Radens. 2002. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM J. Res. Devel.* 46, 2.3 (2002), 187–212.

[123] Sasikanth Manipatruni, Dmitri E. Nikonov, and Ian A. Young. 2018. Beyond CMOS computing with spin and polarization. *Nat. Phys.* 14, 4 (2018), 338–343.

[124] Pedro Marcuello, Antonio González, and Jordi Tubella. 1998. Speculative multithreaded processors. In *12th International Conference on Supercomputing*. ACM, 77–84.

[125] Mahta Mayahinia, Abhairaj Singh, Christopher Bengel, Stefan Wiefels, Muath A. Lebdeh, Stephan Menzel, Dirk J. Wouters, Anteneh Gebregiorgis, Rajendra Bishnoi, Rajiv Joshi, et al. 2022. A voltage-controlled, oscillation-based ADC design for computation-in-memory architectures using emerging ReRAMs. *ACM J. Emerg. Technol. Comput. Syst.* 18, 2 (2022), 1–25.

[126] Emmanuelle J. Merced-Grafals, Noraica Dávila, Ning Ge, R. Stanley Williams, and John Paul Strachan. 2016. Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications. *Nanotechnology* 27, 36 (2016), 365202.

[127] Micron. 2018. Hybrid Memory Cube - HMC Gen2.0. Retrieved from http://www.micron.com/-/media/client/global/documents/products/data-sheet/hmc/gen2/hmc_gen2.pdf.

[128] Sparsh Mittal. 2018. A survey of ReRAM-based architectures for processing-in-memory and neural networks. *Mach. Learn. Knowl. Extract.* 1, 1 (2018). DOI: https://doi.org/10.3390/make1010005

[129] Amir Morad, Leonid Yavits, and Ran Ginosar. 2014. Efficient dense and sparse matrix multiplication on GP-SIMD. In *24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'14)*. IEEE, 1–8.

[130] Amir Morad, Leonid Yavits, and Ran Ginosar. 2015. GP-SIMD processing-in-memory. *ACM Trans. Archit. Code Optim.* 11, 4 (2015), 53.

[131] Amir Morad, Leonid Yavits, Shahar Kvatinsky, and Ran Ginosar. 2016. Resistive GP-SIMD processing-in-memory. *ACM Trans. Archit. Code Optim.* 12, 4 (2016), 57.

[132] Christopher Münch, Rajendra Bishnoi, and Mehdi B. Tahoori. 2019. Reliable in-memory neuromorphic computing using spintronics. In *24th Asia and South Pacific Design Automation Conference*. 230–236.

[133] Christopher Münch, Rajendra Bishnoi, and Mehdi B. Tahoori. 2020. Tolerating retention failures in neuromorphic fabric based on emerging resistive memories. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. IEEE, 393–400.

[134] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *5th IEEE International Memory Workshop (IMW'13)*. IEEE, 21–25.

[135] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2019. Processing data where it makes sense: Enabling in-memory computation. *Microprocess. Microsyst.* 67 (2019), 28–41.

[136] Anirban Nag, Rajeev Balasubramonian, Vivek Srikumar, Ross Walker, Ali Shafiee, John Paul Strachan, and Naveen Muralimanohar. 2018. Newton: Gravitating towards the physical limits of crossbar acceleration. *IEEE Micro* 38, 5 (2018), 41–49.

[137] Ravi Nair, Samuel F. Antao, Carlo Bertolli, Pradip Bose, Jose R. Brunheroto, Tong Chen, C.-Y. Cher, Carlos H. A. Costa, Jun Doi, Constantinos Evangelinos, et al. 2015. Active memory cube: A processing-in-memory architecture for exascale systems. *IBM J. Res. Devel.* 59, 2/3 (2015), 17–1.

[138] Sarath Mohanachandran Nair, Rajendra Bishnoi, Mohammad Saber Golanbari, Fabian Oboril, Fazal Hameed, and Mehdi B. Tahoori. 2017. VAET-STT: Variation aware STT-MRAM analysis and design space exploration tool. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 37, 7 (2017), 1396–1407.

[139] Sarath Mohanachandran Nair, Rajendra Bishnoi, and Mehdi B. Tahoori. 2019. A comprehensive framework for parametric failure modeling and yield analysis of STT-MRAM. *IEEE Trans. Very Large Scale Integ. Syst.* 27, 7 (2019), 1697–1710.

[140] Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. 2020. A classification of memory-centric computing. *ACM J. Emerg. Technol. Comput. Syst.* 16, 2 (2020), 1–26.
[141] Dmitri E. Nikonov and Ian A. Young. 2015. Benchmarking of beyond-CMOS exploratory devices for logic integrated circuits. *IEEE J. Explor. Solid-State Computat. Dev. Circ.* 1 (2015), 3–11.
[142] J.-P. Noel, M. Pezzin, R. Gauchi, J.-F. Christmann, M. Kooli, H.-P. Charles, L. Ciampolini, M. Diallo, F. Lepin, B. Blampey, et al. 2020. A 35.6 TOPS/W/mm$^2$ 3-stage pipelined computational SRAM with adjustable form factor for highly data-centric applications. *IEEE Solid-State Circ. Lett.* 3 (2020), 286–289.
[143] H. Noyes et al. 2014. Micron's automata processor architecture: Reconfigurable and massively parallel automata processing. In *5th International Symposium on Highly-efficient Accelerators and Reconfigurable Technologies*.
[144] NVIDIA. 2012. Tesla K20X GPU Accelerator Board Specification. *Board Specification* (2012). https://www.nvidia.com/content/PDF/kepler/tesla-k20-active-bd-06499-001-v03.pdf.
[145] NVIDIA. 2014. GeForce GTX 745. Retrieved from http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-745-oem/specifications.
[146] NVIDIA. 2016. TITAN X Pascal. Retrieved from https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/.
[147] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B. Tahoori. 2015. Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 34, 3 (2015), 367–380.
[148] OpenCV. 2020. OpenCV library. Retrieved from https://docs.opencv.org/master/.
[149] Mark Oskin, Frederic T. Chong, and Timothy Sherwood. 1998. *Active Pages: A Computation Model for Intelligent Memory*. Vol. 26. IEEE Computer Society.
[150] D. Pala, Giovanni Causapruno, Marco Vacca, Fabrizio Riente, Giovanna Turvani, Mariagrazia Graziano, and Maurizio Zamboni. 2015. Logic-in-memory architecture made real. In *IEEE International Symposium on Circuits and Systems (ISCAS'15)*. IEEE, 1542–1545.
[151] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (1997), 34–44.
[152] David A. Patterson. 2006. Future of computer architecture. In *Berkeley EECS Annual Research Symposium (BEARS'06)*.
[153] J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *IEEE Hot Chips Symposium (HCS'11)*. IEEE, 1–24.
[154] Alex Peleg and Uri Weiser. 1996. MMX technology extension to the Intel architecture. *IEEE Micro* 16, 4 (1996), 42–50.
[155] M. Radosavljević, M. Freitag, K. V. Thadani, and A. T. Johnson. 2002. Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors. *Nano Lett.* 2, 7 (2002), 761–764.
[156] Akshay Krishna Ramanathan, Gurpreet S. Kalsi, Srivatsa Srinivasa, Tarun Makesh Chandran, Kamlesh R. Pillai, Om J. Omer, Vijaykrishnan Narayanan, and Sreenivas Subramoney. 2020. Look-up table based energy efficient processing in cache support for neural network acceleration. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 88–101.
[157] Indranil Roy. 2015. *Algorithmic Techniques for the Micron Automata Processor*. Ph.D. Dissertation. Georgia Institute of Technology.
[158] Sayeef Salahuddin, Kai Ni, and Suman Datta. 2018. The era of hyper-scaling in electronics. *Nat. Electron.* 1, 8 (2018), 442–450.
[159] Gurtej S. Sandhu. 2013. Emerging memories technology landscape. In *13th Non-Volatile Memory Technology Symposium (NVMTS'13)*. IEEE, 1–5.
[160] Nour Sayed, Rajendra Bishnoi, and Mehdi B. Tahoori. 2019. Fast and reliable STT-MRAM using nonuniform and adaptive error detecting and correcting scheme. *IEEE Trans. Very Large Scale Integ. Syst.* 27, 6 (2019), 1329–1342.
[161] Nour Sayed, Longfei Mao, Rajendra Bishnoi, and Mehdi B. Tahoori. 2019. Compiler-assisted and profiling-based analysis for fast and efficient STT-MRAM on-chip cache design. *ACM Trans. Des. Automat. Electron. Syst.* 24, 4 (2019), 1–25.
[162] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, et al. 2013. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In *46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 185–197.
[163] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 273–287.
[164] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 14–26.

[165] Adnan Shaout and Taisir Eldos. 2003. On the classification of computer architecture. *Int. J. Sci. Technol.* 14 (2003).
[166] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. 2016. Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs. In *Conference on Design, Automation & Test in Europe*. EDA Consortium, 948–953.
[167] Anne Siemon, Stephan Menzel, Rainer Waser, and Eike Linn. 2015. A complementary resistive switch-based crossbar array adder. *IEEE J. Emerg. Select. Topics Circ. Syst.* 5, 1 (2015), 64–74.
[168] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[169] Abhairaj Singh, Rajendra Bishnoi, Rajiv V. Joshi, and Said Hamdioui. 2022. Referencing-in-array scheme for RRAM-based CIM architecture. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 1413–1418.
[170] Abhairaj Singh, Sumit Diware, Anteneh Gebregiorgis, Rajendra Bishnoi, Francky Catthoor, Rajiv V. Joshi, and Said Hamdioui. 2021. Low-power memristor-based computing for edge-ai applications. In *IEEE International Symposium on Circuits and Systems (ISCAS'21)*. 1–5.
[171] Abhairaj Singh, Muath Abu Lebdeh, Anteneh Gebregiorgis, Rajendra Bishnoi, Rajiv V. Joshi, and Said Hamdioui. 2021. SRIF: Scalable and reliable integrate and fire circuit ADC for memristor-based CIM architectures. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 68, 5 (2021), 1917–1930.
[172] Abhairaj Singh, Mahdi Zahedi, Taha Shahroodi, Mohit Gupta, Anteneh Gebregiorgis, Manu Komalan, Rajiv Joshi, Francky Catthoor, Rajendra Bishnoi, and Said Hamdioui. 2022. CIM-based robust logic accelerator using 28 nm STT-MRAM characterization chip tape-out. In *International Conference on Artificial Intelligence Circuits and Systems (AICAS'22)*.
[173] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2019. Near-memory computing: Past, present, and future. *Microprocess. Microsyst.* 71 (2019), 102868.
[174] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2018. A review of near-memory computing architectures: Opportunities and challenges. In *21st Euromicro Conference on Digital System Design (DSD'18)*.
[175] Mathias Soeken, Saeideh Shirinzadeh, Pierre-Emmanuel Gaillardon, Luca Gaetano Amarú, Rolf Drechsler, and Giovanni De Micheli. 2016. An MIG-based compiler for programmable logic-in-memory architectures. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6.
[176] Kyomin Sohn, Won-Joo Yun, Reum Oh, Chi-Sung Oh, Seong-Young Seo, Min-Sang Park, Dong-Hak Shin, Won-Chang Jung, Sang-Hoon Shin, Je-Min Ryu, et al. 2017. A 1.2 V 20 nm 307 GB/s HBM DRAM with at-speed wafer-level IO test scheme and adaptive refresh considering temperature distribution. *IEEE J. Solid-state Circ.* 52, 1 (2017), 250–260.
[177] Kurt Stockinger and Kesheng Wu. 2008. Bitmap indices for data warehouses. In *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications*. IGI Global, 1590–1605.
[178] Harold S. Stone. 1970. A logic-in-memory computer. *IEEE Trans. Comput.* C-19, 1 (1970), 73–78. DOI: https://doi.org/10.1109/TC.1970.5008902
[179] John A. Stratton, Christopher Rodrigues, I.-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-Mei W. Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Cent. Reliab. High-Perform. Comput.* 127 (2012).
[180] Stanley Y. W. Su et al. 1980. Magnetic bubble memory architectures for supporting associative searching of relational databases. *IEEE Trans. Comput.* 100, 11 (1980), 957–970.
[181] Arun Subramaniyan, Jingcheng Wang, Ezhil R. M. Balasubramanian, David Blaauw, Dennis Sylvester, and Reetuparna Das. 2017. Cache automaton. In *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. ACM, New York, NY, 259–272. DOI: https://doi.org/10.1145/3123939.3123986
[182] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. 2011. Evaluation of delta compression techniques for efficient live migration of large virtual machines. *ACM SIGPLAN Not.* 46, 7 (2011), 111–120.
[183] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. DeepFace: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1701–1708.
[184] Mark R. Thistle and Burton J. Smith. 1988. A processor architecture for horizon. In *Supercomputing'88. Vol. 1. Proceedings*. IEEE, 35–41.
[185] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. 1995. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, Vol. 23. ACM, 392–403.
[186] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. IEEE, 335–344.
[187] Giorgos Vasiliadis, Michalis Polychronakis, and Sotiris Ioannidis. 2011. Parallelization and characterization of pattern matching using GPUs. In *IEEE International Symposium on Workload Characterization (IISWC'11)*. IEEE, 216–225.

[188] Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor. 2009. SD-VBS: The San Diego Vision Benchmark Suite. In *IEEE International Symposium on Workload Characterization*. IEEE, 55–64.

[189] Oreste Villa, Daniel R. Johnson, Mike Oconnor, Evgeny Bolotin, David Nellans, Justin Luitjens, Nikolai Sakharnykh, Peng Wang, Paulius Micikevicius, Anthony Scudiero, et al. 2014. Scaling the power wall: A path to exascale. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 830–841.

[190] Ioannis Vourkas, Dimitrios Stathis, Georgios Ch Sirakoulis, and Said Hamdioui. 2016. Alternative architectures toward reliable memristive crossbar memories. *IEEE Trans. Very Large Scale Integ. Syst.* 24, 1 (2016), 206–217.

[191] Jack Wadden, Vinh Dang, Nathan Brunelle, Tommy Tracy II, Deyuan Guo, Elaheh Sadredini, Ke Wang, Chunkun Bo, Gabriel Robins, Mircea Stan, et al. 2016. ANMLzoo: A benchmark suite for exploring bottlenecks in automata processing engines and architectures. In *IEEE International Symposium on Workload Characterization (IISWC'16)*. IEEE, 1–12.

[192] David W. Wall. 1991. Limits of instruction-level parallelism. In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*. 176–188.

[193] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman P. Jouppi. 2014. Endurance-aware cache line management for non-volatile caches. *ACM Trans. Archit. Code Optim.* 11, 1 (2014), 4.

[194] Xiaowei Wang, Jiecao Yu, Charles Augustine, Ravi Iyer, and Reetuparna Das. 2019. Bit prudent in-cache acceleration of deep convolutional neural networks. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE, 81–93.

[195] Ying Wang, Yinhe Han, Lei Zhang, Huawei Li, and Xiaowei Li. 2015. ProPRAM: Exploiting the transparent logic resources in non-volatile memory for near data computing. In *52nd Annual Design Automation Conference*. ACM, 47.

[196] Rainer Waser. 2012. Redox-based resistive switching memories. *J. Nanosci. Nanotechnol.* 12, 10 (2012), 7628–7640.

[197] Christian Weis, Norbert Wehn, Loi Igor, and Luca Benini. 2011. Design space exploration for 3D-stacked DRAMs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'11)*. IEEE, 1–6.

[198] Stephan Wong, Thijs Van As, and Geoffrey Brown. 2008. $\rho$-VEX: A reconfigurable and extensible softcore VLIW processor. In *International Conference on ICECE Technology*. IEEE, 369–372.

[199] Kesheng Wu. 2005. FastBit: An efficient indexing technology for accelerating data-intensive science. In *Journal of Physics: Conference Series*, Vol. 16. IOP Publishing, 556.

[200] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2015. Interconnect networks for memristor crossbar. In *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'15)*. IEEE, 124–129.

[201] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2016. Boolean logic gate exploration for memristor crossbar. In *International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS'16)*. IEEE, 1–6.

[202] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, and Koen Bertels Said Hamdioui. 2015. Fast Boolean logic mapped on memristor crossbar. In *33rd IEEE International Conference on Computer Design (ICCD'15)*. IEEE, 335–342.

[203] Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. 2017. Scouting logic: A novel memristor-based logic design for resistive computing. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'17)*. IEEE, 335–340.

[204] Peng Yao, Huaqiang Wu, Bin Gao, Guo Zhang, and He Qian. 2015. The effect of variation on neuromorphic network based on 1T1R memristor array. In *15th Non-Volatile Memory Technology Symposium (NVMTS'15)*. IEEE, 1–3.

[205] Leonid Yavits, Shahar Kvatinsky, Amir Morad, and Ran Ginosar. 2015. Resistive associative processor. *IEEE Computer Architecture Letters* 14, 2 (2014), 148–151.

[206] Jintao Yu, Hoang Anh Du Nguyen, Mottaqiallah Taouil, and Said Hamdioui. 2018. Memristor devices for computation-in-memory. In *Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1718–1725.

[207] Qiuling Zhu, Berkin Akin, H. Ekin Sumbul, Fazle Sadi, James C. Hoe, Larry Pileggi, and Franz Franchetti. 2013. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In *IEEE International 3D Systems Integration Conference (3DIC'13)*. IEEE, 1–7.

[208] Yuan Zu, Ming Yang, Zhonghu Xu, Lin Wang, Xin Tian, Kunyang Peng, and Qunfeng Dong. 2012. GPU-based NFA implementation for memory efficient high speed regular expression matching. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 129–140.

Received 23 July 2020; revised 4 February 2022; accepted 1 June 2022