A Classification of Memory-Centric Computing

HOANG ANH DU NGUYEN, Delft University of Technology
JINTAO YU, Delft University of Technology
MUATH ABU LEBDEH, Delft University of Technology
MOTTAQIALLAH TAOUIL, Delft University of Technology
SAID HAMDIOUI, Delft University of Technology
FRANCKY CATTHOOR, Inter-university Micro-Electronics Center (IMEC)

Technological and architectural improvements have been constantly required to sustain the demand of faster and cheaper computers. However, CMOS down-scaling is suffering from three technology walls: leakage wall, reliability wall and cost wall. On top of that, performance increase due to architectural improvements is also gradually saturating due to three well-known architecture walls: memory wall, power wall and instruction level parallelism (ILP) wall. Hence, a lot of research is focusing on proposing and developing new technologies and architectures. In this paper, we present a comprehensive classification of memory-centric computing architectures; it is based on three metrics: computation location, level of parallelism and used memory technology. The classification does not only provide an overview of existing architectures with their pros and cons, but also unify the terminology that uniquely identifies these architecture, and highlight the potential future architectures that can be further explored. Hence, it sets up a direction for future research in the field.

CCS Concepts: • **Computer systems organization** → *Special purpose systems*; • **Hardware** → *Spintronics and magnetic technologies.*

Additional Key Words and Phrases: Computation-in-Memory, Computer Architectures, Resistive Computing, Memory-centric Computing

ACM Reference Format:
Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. 2018. A Classification of Memory-Centric Computing. 1, 1 (October 2018), 25 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

For several decades, technology scaling has provided a 43% performance gain for each successive node and cheaper computers as a result of a higher operating frequency and lower cost per transistor, respectively [15, 53]. On top of that, smart architectural improvements such as pipelining and cache hierarchies increased the computer performance up to 50% every two years [48]. However, CMOS scaling suffers from three main walls: leakage wall, reliability wall and cost wall [44], while computer architectures also face three walls: memory wall, power wall and instruction level parallelism

Authors' addresses: Hoang Anh Du Nguyen, Delft University of Technology, Melkeweg 4, H.A.DuNguyen@tudelft.nl; Jintao Yu, Delft University of Technology, Melkeweg 4, J.Yu-1@tudelft.nl; Muath Abu Lebdeh, Delft University of Technology, Melkeweg 4, M.F.M.AbuLebdeh@tudelft.nl; Mottaqiallah Taouil, Delft University of Technology, Melkeweg 4, M.Taouil@tudelft.nl; Said Hamdioui, Delft University of Technology, Melkeweg 4, S.Hamdioui@tudelft.nl; Francky Catthoor, Inter-university Micro-Electronics Center (IMEC), , Francky.Catthoor@imec.be.

(ILP) wall [99]. In order to address these walls, novel technologies and architecture are under research to improve the performance [53]. As a result, an enormous amount of computer architectures has been proposed recently. Therefore, a complete classification of these architectures is needed; not only to have a useful way of describing and comparing them, but also to have a clear view about what is explored and what not yet.

Limited work has addressed this problem. Most of the well-known classifications separate the processors from the memory. Therefore, these classifications often are processor-centric based architectures, such as Flynn's [35], Skillicorn's [117] and Shami-Hemani's classification [112]. Although these classifications work well for processor-centric architectures proposed in the past decades, they are not applicable to the emerging memory-centric architectures. Other small-scale surveys mostly target a specific type of computer architectures such as vector processors, automata processors or processing-in-memory architectures [23, 57, 68, 108, 113, 122, 125, 126]. These surveys only discuss a limited part of the computer architecture classification, and in addition, do not contain the complete space of both conventional processor-centric architectures and memory-centric architectures. Therefore, these surveys often make no distinction between processing inside and near the memory. This leads to a confusion in terminology (e.g., processing-in-memory, logic-in-memory, in-memory computing, near-memory computing, etc.). For example, Hybrid Memory Cube is considered to be near-memory-computing [100], however, it is also referred to as processor-in-memory [3]. Some recent classifications and reviews did mention those architectures in the context of technology development [93, 136]. However, these papers mostly targeted the technological feasibility instead of the characteristics and variants of such computer architectures. In addition to the above, there are some architecture-related papers that briefly discussed the features of emerging architectures [88, 90, 105]. However, they are incomplete, focus mostly on relatively narrow aspects and only classify the architectures based on applications [88] and logic design methods [90, 105]. In short, there is still a lack of systematic and complete classification that focuses on memory-centric computing or computer architectures in general. This is exactly the target of this paper.

This paper presents a comprehensive classification of memory-centric computing, and discusses both conventional and emerging computing architectures. The classification is based on three metrics: computation location, memory technology and computation parallelism. The computation location indicates where the computations are performed (e.g., near or far from the memory) and provides an insight regarding the severeness of the memory wall. The memory technology, which provides characteristics of the memory, can enable new computer architectures (e.g., resistive computing). The computation parallelisms specifies the type of parallelism that can be exploited in an architecture (e.g. task level parallelism). With these distinct metrics, the classification covers *all* computing architectures in general and memory-centric computing in specific. Note however that it does not make previous proposed classifications obsolete, as they typically target specific sub-classes. In short, the contributions of this paper are the following:

- Unify the terminology for computer architectures such that it is applicable to all computing paradigms including conventional, in-memory and near-memory computing.
- Propose a complete classification that includes both existing and emerging architectures.
- Explain one representative architecture of each sub-class in detail.
- Discuss and evaluate the main advantages and disadvantages of the different classes and selected architectures.
- Highlight the whole space of memory-centric computing, including the non-explored architectures.

The rest of this paper is structured as follows. Section 2 shows the metrics used in the classification, briefly introduces the four classes, and provides a quantitative comparison among them. Sections 3, 4, and 5 present the characteristics of the three memory-centric computing classes; the fourth class contains the traditional von Neuman architectures and is

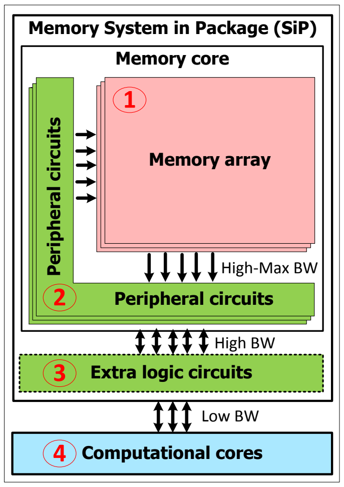


Fig. 1. Computer Architecture

out-of-the-scope of this paper. Section 6 discusses the pros and cons of this classification and compares its with existing ones. Finally, section 7 concludes this paper.

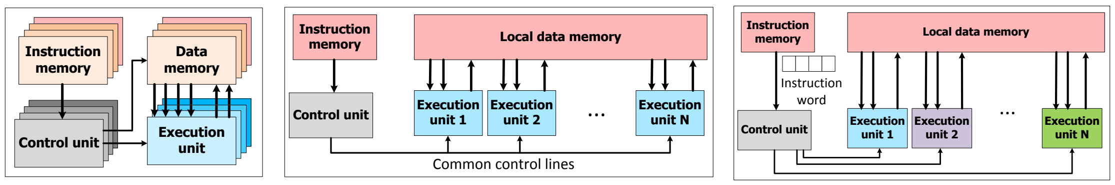
2 CRITERIA AND CLASSIFICATION

In this section, we first present the set of metrics to classify computer architectures. Thereafter, we map the existing architectures on our classification. Finally, we compare the classes qualitatively based on their most important metrics.

2.1 Classification Metrics

We propose several metrics to classify computer architectures based on the computing resources and memory. A computer architecture or system consists of (one or more) memories and (one or more) computational units as shown in Fig. 1. The memories can reside in a core (i.e., memory core) or System-in-Packages (SiP). A memory core consists of one or more cell arrays (used for storage) and peripheral circuits (used to access the memory cells). Note that register files and caches are not considered as storage here, as they are optimized for speed with relatively small capacity and temporary storage [48]. Hence, the long term storage of data takes place in the higher layers such as main memory and solid-state disks. Traditionally, the computing takes place in the computational cores. However, recently architectures with computing power in the memory have been proposed [45, 98, 100]. In case the memory contains additional logic circuits such as in Hybrid Memory Cubes (HMC) [100], we speak of a System in Package (SiP). With an increasing distance from the main memory array, the available bandwidth (specified by BW in Fig. 1) reduces; note that the bandwidth here is related to the memory bottleneck and will be discussed further in Section 2.3. Based on these definitions, the following metrics are used to classify computer architectures: computation location, memory technology and computation parallelism; they are discussed next.

Computation location: it indicates ***where*** *the result of the computation is produced.* A computation is defined here as a primitive logic function (e.g., logical operations) or arithmetic operation (e.g., addition, multiplication). Fig. 1 indicates the four possibilities where a computation result can be produced; they can be identified by four circled numbers. If the result is produced *within* the memory core, (i.e., the computing takes places within one of the memories), the computer



(a) Task Parallelism (b) Data Parallelism (c) Instruction Parallelism

Fig. 2. Three Types of Computation Parallelism

architecture is referred to as *Computation-Inside-Memory (CIM)*. If the result is produced outside the memory core, the architecture is referred to as *Computation-Outside-Memory (COM)*. Both CIM and COM can be further sub-classified.

It is worth stressing that **CIM** architectures perform computations *within* the memory core. As already mentioned, the memory consists of a *memory array* and the *peripheral circuits*. Specifically, depending on *where* the result of the computation is produced, CIM architectures can be divided into two basic sub-classes. These sub-classes can be combined into many hybrid combinations. We will describe this large space by focusing first on its two extreme sides:

- *CIM-Array (CIM-A):* In CIM-A, the computing result is produced within the array. 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 [65, 71]. CIM-A architectures require always a redesign of cells 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. A memory cell is namely heavily optimized in terms of processing stack and layout; hence, any changes in the array access require a completely new cell design and characterization process as the material stack of a memory array is specifically optimized for specific control voltages, current, etc. In addition, modifications in the periphery are sometimes needed to support the changes in the cell changes. Therefore, CIM-A architectures can be sub-divided into two groups: (1) basic CIM-A where only changes inside the memory array are required, and (2) hybrid CIM-A where in addition to major changes in the memory array also minimal to medium changes are required in the peripheral circuit. An example of basic CIM-A is an architecture that performs computations using implication logic [75]. In this logic style, only one memory row is activated at a time, and a number of columns (bits) are read out through sense amplifiers. Hence, due to the same usage as in normal memory, the peripheral circuits do not require any modifications. An example of hybrid CIM-A is an architecture that performs computations using MAGIC [71]; 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.
- *CIM-Periphery (CIM-P):* In CIM-P, the computing result is produced within the peripheral circuitry. Typical examples of CIM-P architectures contains logical operations and vector-matrix multiplications [21, 80, 134]. CIM-P architectures typically contain dedicated peripheral circuits such as DACs and/or ADCs [37, 111], and customized sense amplifiers [80, 134]. 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, similarly as to the CIM-A sub-classes, the CIM-P architectures are also further

divided into two groups: (1) basic CIM-P where only changes inside the peripheral is required, which means the current levels should not be affected, and (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 basic CIM-P is Pinatubo logic [80]. Pinatubo activates two or more (but not many) rows of a memory array simultaneously during read operations for computations; in addition to a customized sense amplifier to perform the logic operation, this architecture also requires modifications in the address decoder to activate several rows. Note however, that modifications in the cell/array are not required as the total read current is still small. An example of hybrid CIM-P is ISAAC [111]. ISAAC activates all rows of a memory array at the same time during read operations to perform a matrix vector product using an ADC read out circuit. This architecture accumulates currents in the bitline that impose higher electrical loading in the memory array; hence, not only the periphery circuit is heavily modified but also the cell requires changes due to the high bit-line current.

The difference between CIM-A and CIM-P classes is the location of producing results. The results of CIM-A architectures are produced inside the memory array, which may sometimes require read-out operations to obtain the results for further calculations; instead, in CIM-P the results are obtained directly after the operations and may sometimes need an additional step to write the results back to memory. In order to perform computations, both sub-classes impact the design of the memory core. However, in many/most cases both the cell and the peripheral circuitry require changes, i.e., they are hybrids. In case these changes affect mostly the cell, we speak of hybrid CIM-A, otherwise hybrid CIM-P.

In **COM classes**, computations take either place in the extra logic circuits inside the memory SiP (3) or in the traditional computational cores (4) such as CPU, FPGA, etc. In case of the former, the computations take place near the memory core and the architecture is referred to as Computation-Outside-Memory Near (COM-N). In case of the latter, the architecture is referred to as Computation-Outside-Memory Far (COM-F). Note that the bandwidth is still high for COM-N as compared to COM-F, but lower than CIM-A and CIM-P.

Note that architectures where the computation takes place in difference places (e.g., array and peripheral) are called composite architectures. Hence, they are compositions of the leaf nodes in our classification tree. In addition, an architecture could have multiple primitive functions, each with a different computation location. Also these architectures are considered to be composite.

In addition to the computation location, which specifies where the results are produced, it is possible to further divide the classes using the computation method by specifying how the computation is performed. For example, CIM-A often uses memristor-based computations such as IMPLY [14, 115], Snider [118], MAGIC [71]. CIM-P often uses current summations such as Scouting logic [134], Ambit [110] and Pinatubo [80]. However, this metric is not included to the classification for two reasons: (i) it is strongly coupled to the computational location, and (ii) it makes the classification too complex and hence loses its simplicity. Nevertheless, including such a metric can be complementary to our work. A further sub-classification based on this metric can be based on existing classifications as shown in [26, 105].

Memory technology: it indicates which technology is used to implement the memory array. The technologies are either conventional charge-based memories such as DRAM/SRAM [85, 86, 92] or emerging non charge-based memories [107]. The non charge-based memories can be further divided into different types based on their physical mechanism: resistive [73, 107], "magnetic" memories [10, 19, 107], molecular memories [41, 77, 78, 102] or mechanical memories [16, 42]. Resistive memories store the data as a resistance value; it includes Resistive RAM (RRAM) [129], phase change memory (PCM) [73], etc. The resistance in RRAM is determined by the presence or absence of a conductive filament between its two electrodes [107], while the resistance in PCM relies on a change between amorphous and crystalline phases [104, 130]. Magnetic memories, such as Magnetic RAM (MRAM), store the data using the magnetization direction of the free layer with respect to the hard or reference layer; it includes, for example, conventional magnetic RAM [140] and STT-MRAM [36, 50]. The resistive and magnetic memories are organized in crossbars with cells placed at each junction. The other types of memories, (i.e., molecular memories, mechanical memories) have not been shown to be useful for computing yet; hence, they are not discussed further in this classification. It is worth mentioning that each of these memory technologies has its own characteristics (read/write latency, endurance, capacity, etc.) and therefore are deployed at different levels in the memory hierarchy [107]. Therefore, the memory technology does not only dictate which CIM operations are technology-wise feasible, but also where in the memory hierarchy they take place.

Computation parallelism: it indicates the level of parallelism that can be exploited in a computer system; i.e., task, data, and/or instruction level parallelism, as shown in Fig. 2. An architecture supports task level parallelism when it contains multiple independent control units and multiple data memories (see Fig. 2a). The independent control units can be used to execute multiple threads or instruction sequences from the same application concurrently; examples are multithreading [30, 124] and multicore systems [40]. In data parallelism, a single control unit is used to apply the *same* instruction concurrently on a collection of data elements (see Fig. 2b); note that all execution units share the same control signals. The data elements can be processed using constant sizes (e.g., vector and array processor [28, 33]) or varying sub-word sizes (e.g., SWAR (SIMD Within A Register) processor [34, 101]). In instruction level parallelism, a single control unit is used to execute *various* instructions concurrently (see Fig. 2c); hence, the execution units have different control signals. A further distinction can be made based on intra-instruction (e.g., pipe-lined processor [123]), inter-instruction (e.g., VLIW processor [131]) parallelism, or a combination of them (e.g., speculative processor [87]).

The three above-mentioned metrics (i.e., computation location, memory technology and computation parallelism) are dependent on each other. The computation location has a big impact on the feasibility of the other two metrics. For example, realizing ILP in CIM-A is quite difficult or realizing COM-N with SRAM is not economically feasible. Also the parallelism is not fully independent from the computation location and memory technology. For example, data parallelism is often applied straightforward in CIM-A and CIM-P [27, 37]; however it is difficult to realize ILP in CIM-A, while it is much easier in COM-N and COM-F due to the intrinsic pipeline stages in conventional processors. The computation parallelism is also affected by the technology as the technology poses restrictions on the endurance. For example, exploiting ILP in CIM-A architectures demands a high endurance as more writes are required to store immediate stages and hence, are not attractive for emerging memories like RRAM and PCM with endurance limitations.

2.2 Classification

We classify the existing architectures based on the above discussed metrics; the result is shown in Fig. 3. The references of the abbreviated architectures are listed in Table 1.

The classification contains 48 categories. Some categories, the ones located in *red* planes, show that a *lot* of work has been done for that particular class. For the categories in the *pink* planes, a *moderate number* of work has been done. To our best knowledge, no architectures exists in the *blue* planes; these fields are currently unexplored as they received no attention yet from the research community or non-existing due to current restrictions of the technology; these blue planes are not further discussed in this paper due to two main reasons: (1) scope of the paper; the technical and economical feasibility of these planes requires an intensive discussion and is by itself a new contribution and (2) space limitations. Later on, we will discuss several architectures from the red and pink planes.

The developments in memory-centric computing are shown in the timelime of Fig. 4; this shows the trend of computing moving from COM-F to COM-N, CIM-A and CIM-P. In the figure, a larger circle indicates that more

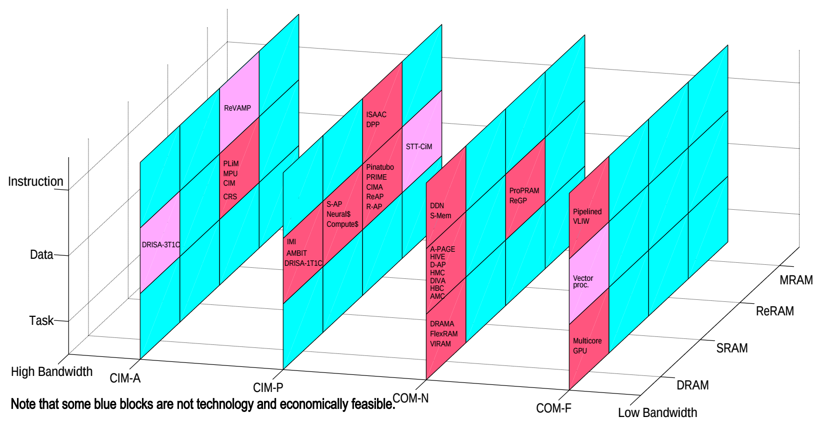


Fig. 3. Memory-centric Computing Classification

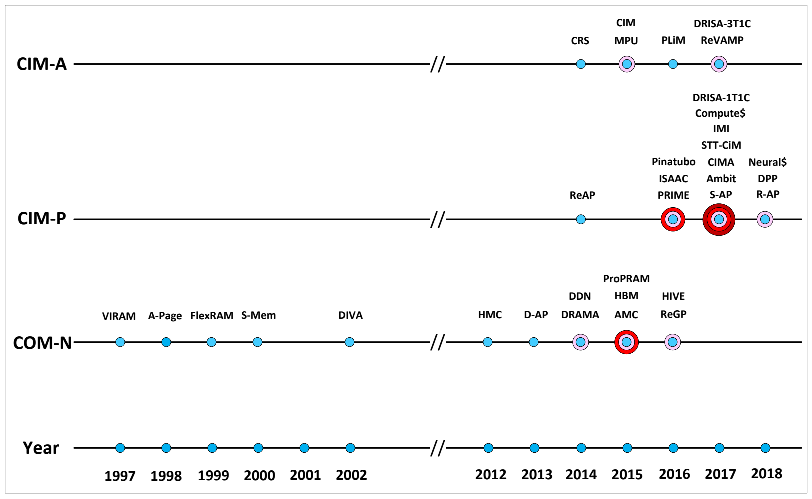


Fig. 4. Memory-centric Computing Timeline

Table 1. Abbreviation List

Abbreviation	Reference
DRISA-3T1C	[79]
ReVAMP	[9]
PLiM	[38]
MPU	[51]
CIM	[27, 45]
CRS	[115]
ISAAC	[111]
DPP	[37]
IMI	[32]
AMBIT	[110]
DRISA-1T1C	[79]
S-AP	[121]
Neural$	[29]
Compute$	[1]
Pinatubo	[80]
PRIME	[21]
CIMA	[26]
ReAP	[137]
R-AP	[138]
STT-CiM	[55]
DDN	[20]
S-Mem	[84]
A-PAGE	[97]
HIVE	[3]
D-AP	[95]
DIVA	[24]
HMC	[49]
AMC	[94]
HBM	[83]
DRAMA	[31]
FlexRAM	[62]
VIRAM	[69]
ProPRAM	[128]
ReGP	[91]
Pipelined	[48, 123]
VLIW	[82, 131]
Vector Proc.	[22, 33, 101]
Multicore	[52]
GPU	[66, 96]

work has been proposed in that year. Note that the conventional architectures in COM-F are not memory-centric and hence, are left out. The concept of merging computation and memory was introduced back in 1970 [120]. This concept became popular around 1997 in COM-N architectures and was further developed until 2002. These COM-N architectures, such as VIRAM [69] (initially named IRAM), DIVA [24] or FlexRAM [62], never commercialized due to the limitations of embedded DRAM technology (i.e., costly fabrication process, and inefficient speed and memory capacity trade-off [54, 63, 64]). After that, a long silent period in academia community was observed from 2002 to 2010. Meanwhile, industrial efforts have been invested to deploy large eDRAM in commercial COM-N systems such as POWER7 processor [60], PlayStation2 [7] and Intel's top-class CPUs [70]. From 2012 to 2016, new commercial COM-N architectures based on novel 3D stacking technology have been proposed such as Hybrid Memory Cube (HMC) [49] and High Bandwidth Memory (HBM) [83]. In the last several years, with the emerging of resistive technology, CIM-A and CIM-P architectures started to become popular.

Note that many of the architectures are hybrid and/or composite which means that they can map into multiple classes. In order to simplify Figure 3, the architectures are classified based on their dominant features. For example, DPP exploits both ILP and DLP; however, DPP focuses more on performing various parallel operations using multiple functional units, while it also processing a whole row/column inside the memory; hence, the dominant feature of DPP parallelism is selected as ILP.

2.3 Qualitative Evaluation

In this subsection, we briefly compare the different computing types qualitatively using the most important classification metric, i.e., the computation location. This metric dictates the type of data movements, computation requirements, available bandwidth, memory design efforts, scalability, endurance requirement and maturity. With respect to the computation requirements, we discuss whether the architectures require a specific data alignment and whether they have the capability to realize complex functions. With respect to available bandwidth, we discuss the capability for data communication between logic and storage units. With respect to the memory design efforts, we discuss the modifications that are required for the cells, array, peripheral circuit and controller. With respect to the scalability, we discuss the possibility to expand the design to increase the concurrent computing capacity. With respect to the endurance requirement, we indicate the endurance level that the architecture demands in order to execute an application. With respect to maturity, we do not only mean the readiness of the memory technology, but also the available programming and software support, and current status (i.e., simulations, prototype or fabrication) for such architectures. With respect to the applications, we roughly indicate the range of applications for each architecture class. The results are shown in Table 2 and 3; their content will be discussed next.

Data movement outside the memory core indicates whether the data will remain in the memory core during computing or transferred to outside computational units. It affects the memory bottleneck due to latency and the energy consumption of data transfers. Both CIM-A and CIM-P architecture have a relatively low amount of data movement outside the memory core, as the processing occurs inside the memory core. Therefore, they have potentials to alleviate the memory bottleneck. Instead of moving data from the memory to the computational cores, the instructions are moved and directly applied to the memory; these instructions typically operate on a large data set, hence a high level of parallelism can be obtained. Note however that the current state of the art typically allows limited functions to be implemented in these architectures. Therefore, complex functions would still require data movements to the computational cores outside the memory. For COM-N and COM-F architectures, data is first read from the memory. Thereafter, they are typically stored in registers before being fed to the processing units. The amount parallelism is limited here due to constraints in the bandwidth, number of available registers and processing units.

Computation requirements include data alignment and the ability to implement complex functions efficiently. Data alignment is required for all architectures. However, CIM-A and CIM-P classes perform computations directly on the data residing inside the memory, and hence, the robustness and performance are impacted more by data misalignment. Note that performing a data alignment cannot be handled by the host processors in in-memory computing architectures

	Data Movement outside memory core	**Computation requirements**		**Available bandwith**	**Memory design efforts**			**Scalability**
		Data Alignment	**Complex function**		**Cells & array**	**Periphery**	**Controller**	
CIM-A	No	Yes	High latency	Max	High	Low/medium	High	Low
CIM-P	No	Yes	High cost	High-Max	Low/medium	High	Medium	Medium
COM-N	Yes	NR	Low cost	High	Low	Low	Low	Medium
COM-F	Yes	NR	Low cost	Low	Low	Low	Low	High

NR: Not Required

Table 2. Comparison among Architecture Classes in terms of Data Movement, Computation Requirements, Available Bandwidth and Memory Design Efforts

	Endurance requirement	**Maturity**		**Applications**
		Software support and Technology	**Development**	
CIM-A	High	Emerging	Simulation	Data Intensive - Computational Complexity (matrix multiplication [47], parallel addition [27])
CIM-P	Medium	Emerging	Simulation	Data Intensive - Bitwise operations (database processing [80, 109, 110], graph processing [2], image processing [46, 110], security and biosequencing application [8])
COM-N	Medium	Commercialized	Fabricated	General-purpose and Application-specific (vector processing [25, 58, 94], automata processing [95], neural computation [20])
COM-F	Low	Common Practise	Fabricated	General-purpose

Table 3. Comparison among Architecture Classes in terms of Endurance Requirement, Maturity and Applications

due to a far communication distance, while adding additional logic inside the memory core to handle this is also not trivial. It requires an area overhead to temporary store operands and do the alignment with CMOS logic. For other classes, the impact of data alignment is less severe; nevertheless, data misalignment can cause a performance degradation in other classes as well.

As the primitive operations in CIM-A and CIM-P are limited, architectures in these classes face challenges in computing complex functions such as arithmetic operations with integer or floating point numbers. As a result, a lot of primitive operations are required to realize such complex functions, if even possible. For example, a multi-bit addition in CIM requires multiple single-bit addition as primitive operations and communication operations between these single-bit additions [27]. On top of that, each primitive step that involves a write operation in a memristor based CIM architecture suffers from a high latency due to its high write time. In addition, current CIM-P architectures require a high cost to implement a diverse set of arithmetic operations as their efficiency today is mainly limited to bitwise logical operations and matrix vector multiplications. Moreover, providing complex functionality using peripheral circuits in CIM-P is difficult, due to limited available area on the memory core. Note that despite these drawbacks, the performance can be still high when sufficient parallelism is exploited, e.g., by operating on multiple crossbars in parallel. Furthermore, data doesn't have to be transferred to the main processor and hence, the energy and performance can be improved. In COM-N and COM-F, computations are performed by CMOS circuits which contain mature, optimized and if needed, dedicated functional units. However, the main bottleneck comes from the many additional data transfers through the memory hierarchy.

Available bandwidth specifies how much data can be transferred between the computational and storage units. This metric is important as it affects the amount of parallelism that can be exploited. The available bandwidth is considered as similarly as bandwidth specification of multiple level in the memory hierarchy; hence it includes four ranges: max (TBs), high (10 GBs), medium (GBs) and low (MBs) [13]. Note that these terms are used for nowadays' memory technology as the exact bandwidth values are subject to change with new or different technologies. CIM-A architectures may exploit the maximum bandwidth, as operations happen inside the memory array. CIM-P architectures have a bandwidth range from high to max, depending on the complexity of the peripheral circuitry. Note that the peripheral circuits can be complex, e.g., when large customized sense amplifiers are used. Therefore, the placement of such sense amplifiers may be limited due to area constraints. In such cases, still a relative high bandwidth can be realized. For COM-N, the bandwidth is bounded by on-chip interconnections between the memory core and extra logic circuits; for example, in Hybrid Memory Cube [100] the bandwidth is limited by the number of TSVs and available registers. This bandwidth for TSV is considered high in comparison with COM-F, where the bandwidth is even lower due to off-chip interconnections [132].

Memory design efforts specifies the required efforts needed to modify the memory (as a storage entity) to make it realized also the computing functionality. In some cases, it is very difficult (or may be even practically impossible) to modify the cells, array, periphery and controller. CIM-A architectures require a redesign of the cell in order to make the computing feasible. Re-characterizing the cell requires a huge effort and induces a huge cost. Other classes, except for hybrid CIM-P, do not require this modification due to the usage of standard memory cells. In terms of changes in the periphery, CIM-P architecture require complex read-out circuits as the output value of two or more accessed cells may end up in multiple levels. Moreover, complex peripheral circuits (i.e., ADC, DAC) limit the scalability when they exhibit internal bottlenecks and could also dominate the area of the memory core when the memory sizes are small. Hence, CIM-P is mainly useful for larger sizes. Other classes, except for hybrid CIM-A, can utilize existing optimized read-out circuits, and hence do not require modifications in the periphery. In terms of memory controller, the complexity reduces from high to low for CIM-A, CIM-P, COM-N and COM-F, respectively. CIM-A architectures require a complex controller as it is responsible for both controlling the crossbar (consisting of a large number of states, each controlling different voltage drivers) and handling data transfer (which involves the usage of buffers/registers to store temporary values). CIM-P architectures have relatively simpler controllers as the computations are constructed in a similar manner as for conventional memory (read/write) operations. The difference is that they typically have to deal with more in-memory operations. COM-N and COM-F architectures utilize the memory in a conventional way, and hence, standard memory controllers can be used.

Scalability specifies how easy or hard is it to scale up the architecture in order to maintain parallelism level. CIM-A has a low scalability due to several reasons such as the lack/ complexity of interconnect network within the memory array it needs, and the difficulty in isolating logic units to ensure parallel executing. CIM-P has a medium scalability as the limited amount of resources inside peripheral circuits makes it difficult to fit large and complex logic units; the complexity of the periphery circuits is the main bottleneck . COM-N also has a medium scalability for the same reason; even though the logic layer of memory SiP has more processing resources than peripheral circuits, it cannot accommodate many complex logic units. COM-F has high scalability due to a mature interconnect network and large space for logic devices.

Endurance requirement specifies how many write operations can be performed before the memory of the architecture starts to fail. A memory that needs a higher number of writes will have a lower lifetime when both have technology-wise the same endurance. Three ranges can be specified for the architectures: a high endurance requirement (i.e. much higher than DRAM endurance 10^{15} [139]); a medium endurance requirement approximately equal to DRAM endurance; and a low endurance requirement much less than the DRAM endurance. CIM-A has in general a high endurance requirement due to the need of multiple write steps to perform simple Boolean functions. CIM-P has a lower endurance requirement as operations are performed during read operations [134]. Nevertheless, results have to be still written back to the memory in order to perform complex functions. As CIM-A and CIM-P architectures are typically based on emerging devices such as memristors, their endurance could be a potential issue. Similarly to CIM-P, COM-N architectures operate closely to the memory and have to write back the results to the main memory due to the absence/limited number of registers and caches. In contrast, COM-F architectures have a much lower endurance requirement as computations are performed using CMOS and the results of the operations are rarely written back to the main memory due to the usage of registers and caches.

Maturity does not only refer to how feasible/reliable the memory technology is, but also how much software support exists and the development status of the architectures in the classes. As CIM-A and CIM-P are relatively new concepts and typically resistive based, lots of work has still to be done to realize these architectures both from a hardware and software point of view. Resistive memories and non-volatile memories in general are typically under research development. For example, the limited endurance puts a constraint on the amount of computations that can be performed in resistive CIM-A architectures. Programming languages, compilers, simulators still need to be developed in general for these architectures. In COM-N class, several architectures have been prototyped in the industry and therefore, they are more mature than CIM-A and CIM-P. Architectures in COM-N have also more software support as they are equipped with tool chains that allow product development on these architectures; for example, Micron's automata processor is already commercialized and is programmed in Automata Network Markup Language (ANML) [95]. COM-F architectures are today's conventional von Neumann architectures. They have the highest maturity from both technological point and software support. With respect to the development status, CIM-A and CIM-P architectures mostly are verified using simulations, either cycle-accurate simulations [4, 12] or circuit verification simulation (i.e., HSpice). COM-N and COM-F architectures are further developed; they have been demonstrated in prototypes and commercial products [100, 103]. In general, COM architectures are more mature than CIM architectures.

Applications that run effectively on the architectures are also described in Table 3. In general, CIM architectures can be more efficient than COM architectures for certain data intensive applications as they are less affected by the memory bottleneck. For CIM-A architectures and several CIM-P architectures (e.g., Pinatubo, CIMA, STT-CiM), there are currently limited types of operations can be efficiently performed on these architectures; hence, limited range of applications can be mapped on these architectures. For example, CIM-A architectures focus more on arithmetic operations such as matrix multiplication [47], parallel addition [27]. CIM-P architectures focus on bulk bitwise applications such as database processing, graph processing, image processing, security and biosequencing application [2, 8, 46, 80, 109, 110]. COM-N architectures are used for both general-purpose and application-specific. A limited number of COM-N architectures are considered as general purpose computers such as FlexRAM [61] and SM [84]. Other COM-N architectures targets specific applications such as vector processing (e.g., VIRAM [58], DIVA [25], AMC [94], etc.), automata processing (D-AP [95]), and neural computation (DDN [20]). COM-F architectures are mostly designed for general-purpose applications.

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

The CIM-A class contains mostly resistive computing architectures that use memristive-based logic circuits [26] to perform computations and resistive RAM (RRAM) as memory technology. The resistive logic circuits may implement different design styles such as stateful logic [75], IMPLY [72], MAGIC [71], CRS-based logic [115], etc. These design

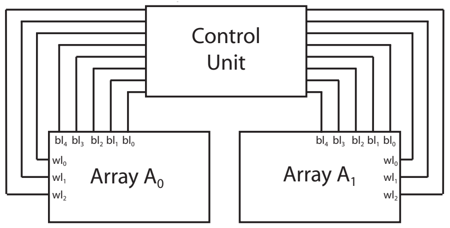


Fig. 5. Complementary Resistive Switch-logic Crossbar Array (CRS) [115]

styles can be further classified, as presented in [105]. In addition to resistive computing, computations can be performed using DRAM cells as demonstrated in [79] which will be explained later.

Few architectures have been proposed in this class; they are Complementary Resistive Switch (CRS) [115], Computation-in-Memory (CIM) [27, 43, 45], Memristive Memory Processing Unit (MPU) [51], Programmable Logic-in-Memory Computer (PLiM) [38], ReRAM based VLIW architecture (ReVAMP) [9], A DRAM-based Reconfigurable In-Situ Accelerator with a 3T1C cell design (DRISA-3T1C) [79]. Most of the architectures, except for REVAMP, have similar organizations. They contain a memristor crossbar (except for DRISA-3T1C) that is used for both storage and computation and a controller that applies the voltages to the memory array. Each architecture uses a different logic style and controller; for example, CRS, MPU, and PLiM use CRS-based logic, Memristive-Aid loGIC (MAGIC), and majority logic, respectively, while CIM can use any logic styles. ReVAMP uses a different architecture and integrates the resistive memory in a pipelined processor in which the memory replaces both the cache and register file. It optimizes traditional pipelined processors by combining the execution, memory and write-back in a single stage. DRISA-3T1C contains a DRAM memory array and performs NOR instructions by reading two rows simultaneously and writing the results back via the sense amplifier to another row. During the read, the capacitances of the accessed cells will discharge the bitline via a transistor when one or both cell values are high; only when both capacitance values are zero the bitline remains high. As examples, we only describe the CRS and ReVAMP architectures next in more detail; they are the latest proposed architectures that represent a basic CIM-A and hybrid CIM-A architectures, respectively. Due to page limitations, only one representative figure is used to describe each architecture.

3.1 Basic CIM-A architecture

CRS architecture was proposed in 2014 by A. Siemon, et al., from RWTH Aachen University [115]. It is a memristor based architecture that exploits data level parallelism using implication logic. The architecture consists of multiple crossbars and a control unit (as shown in Fig. 5 [115]). 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 once 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 [81, 115]. The data transfer between CRS cells is carried out

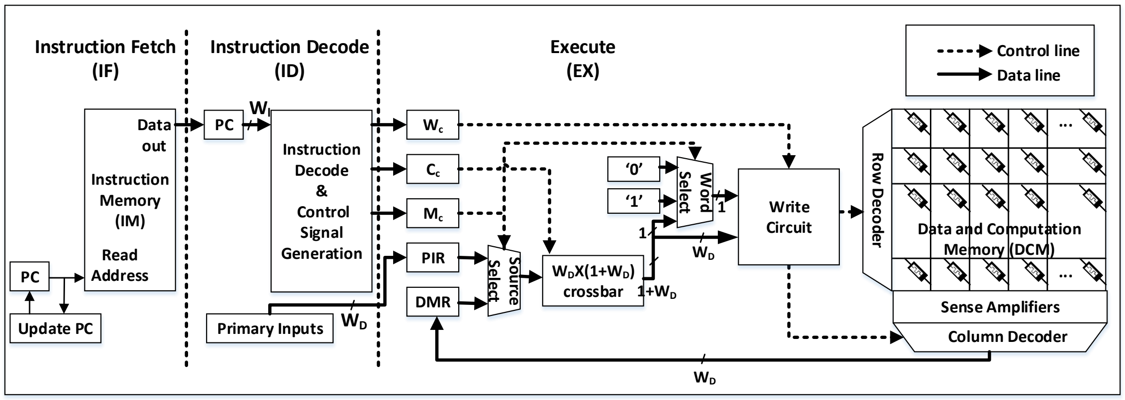


Fig. 6. ReRAM based VLIW architecture (ReVAMP) [9]

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 this value into the destination cell.

In addition to the general characteristic of CIM-A described in Table 2 and 3, CRS has the following advantages: (i) 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, (ii) CRS logic requires fewer cells to perform computations than Fast Boolean Logic (FBL) [133]. However, it also has the following limitations: (i) the latency of the primitive functions varies and requires read-out instructions to determine the voltages that have to be applied, (ii) the RO operation is destructive, hence, a WB operation is required after each RO operation, which increases the latency and energy of computations, (iii) the data tranfer 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, (iv) 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), (v) 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.

3.2 Hybrid CIM-A architecture

ReVAMP was proposed in 2017 by D. Bhattacharjee, et al., from Nanyang Technological University [9]. 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 (EX) stage (as shown in Fig. 6 [9]). The IF block fetches instructions from the Instruction Memory using the program counter (PC) as address, and puts the resulting instruction in the Instruction Register (IR). The ID block decodes the instruction and generates control signals which 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 architecture. 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 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).

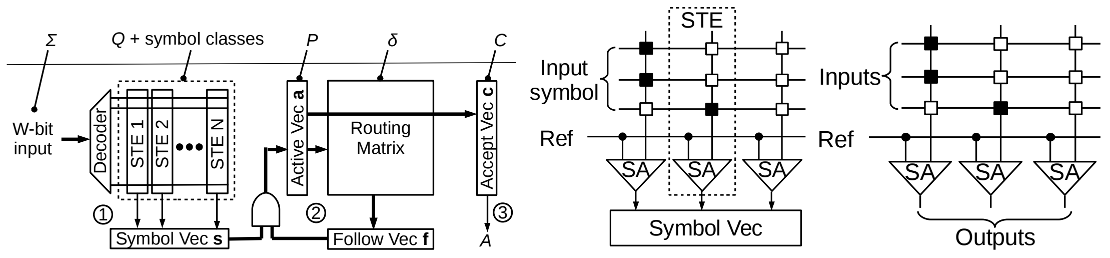
In addition to the general characteristic of CIM-A described in Table 2 and 3, ReVAMP has the following advantages: (i) the data transfer may include direct (within the crossbar based on copying resistance values) and indirect (based on read-out/write-back) schemes, (ii) the crossbar is based on only one device per cell, resulting in a more compact architecture as compared with other architectures which make use of two devices per cell (i.e., Complementary Resistive Switch CRS [115]), (iii) the architecture does not suffer destructive reads as is the case for CRS architecture [115], hence the write energy might be less due to the absence of a write back operation. However, it also has the following limitations: (i) 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 in order to determine the appropriate control voltages, (ii) the architecture has to deal with sneak path currents. Possible solutions as mentioned above, (iii) the EX stage is complex as it integrates both the 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, (iv) 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 [5] and compared against PLiM [38], which is based on a resistive memory with the same logic style.

4 COMPUTATION-IN-MEMORY - PERIPHERY (CIM-P)

The CIM-P class consists of architectures which perform computations during read-out operations (i.e., two or more word lines are activated simultaneously) using special peripheral circuitry. Such operations are typically analog in nature. 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.

A medium number of architectures have been proposed in this class: Resistive Associative Processor (ReAP) [137], A Processing-in-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory (PRIME) [21], A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic (ISAAC) [111], In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology (Ambit) [110], A Processing-in-Memory Architecture for Bulk Bitwise Operations (Pinatubo) [80], In-Memory Intelligence (IMI) [32], Compute Caches (Compute$) [1], A DRAM-based Reconfigurable In-Situ Accelerator with 1T1C design (DRISA-1T1C) [79], Computation-in-Memory Accelerator (CIMA) [26], Computing in Memory Spin-Transfer Torque Magnetic RAM (STT-CiM) [55], Cache Automaton (S-AP) [121], Neural Cache (Neural$) [29], RRAM Automata Processor (R-AP) [138], Data Parallel Processor (DPP) [37].

These architectures fundamentally perform computations in the same way by activating multiple rows simultaneously in the memory and using generally specialized sense amplifiers and/or ADC converters to get the results. ReAP performs computations by implementing Content-addressable-Memory (CAM) operations using look-up-tables (LUTs). PRIME and ISAAC perform vector-matrix multiplications for neural applications. Ambit, IMI, Compute$, DRISA-1T1C, Pinatubo, CIMA, STT-CIM, Neural$ and DPP perform computations using customized sense-amplifiers only; Ambit, IMI and



(a) General Architecture for Automata Processor [138] (b) Used as STEs (c) Used as routers

Fig. 7. Resistive RAM Automata Processor (R-AP) [138]

DRISA-1T1C use DRAM, Compute$ and Neural$ uses SRAM, while the rest is based on non-volatile memory. These architectures can only perform logical operations except for IMI, DRISA-1T1C, Neural$ and DPP which also perform more complex functions by having additional logic inside the peripheral circuits. S-AP and R-AP implement inner product operations in automata processors. S-AP is implemented using SRAM technology while R-AP uses non-volatile memory. As examples, we only describe the R-AP and DPP architectures next in more details; they are the latest proposed architectures that represent basic CIM-P and hybrid CIM-P architectures, respectively.

4.1 Basic CIM-P architecture

R-AP was proposed in 2018 by J. Yu, et al., from Delft University of Technology [138]. The architecture targets an automata processor which 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 Fig. 7a [138]. 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 which are implemented using RRAM technology. Each RRAM column corresponds to an STE which stores the accepting states in RRAM cells, as shown in Fig. 7b [138]. 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, as shown in Fig. 7c [138]; 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 characteristic of CIM-P described in Table 2 and 3, R-AP has the following advantages: (i) the architecture is used as a read-favoured accelerator, which has a positive impact on the endurance due to infrequent use [17, 127]. Only when the automata changes, the STEs and routing matrix have to be updated, (ii) automata processing can be used to perform both logical and arithmetic operations in general, (iii) data can be transferred using both direct and indirect schemes, (iv) the architecture uses non-volatile memory, hence consumes low energy and has a small footprint, (v) 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: (i) the modified peripheral circuitry (row drivers) might pose high overhead in the memory system, (ii) the architecture requires additional

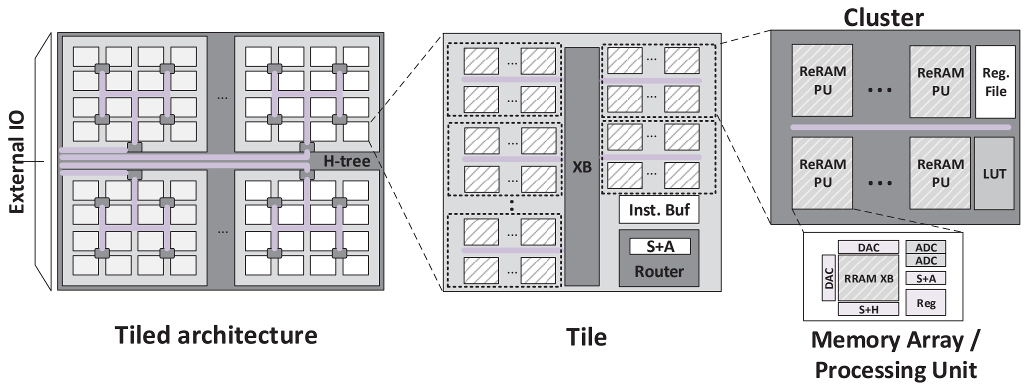


Fig. 8. Data Parallel Processor (DPP) [37]

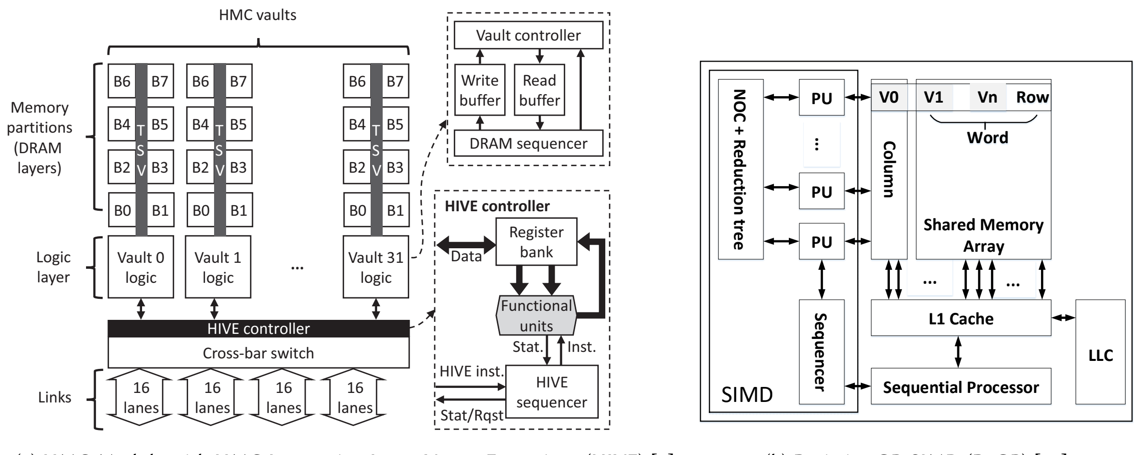
compiling techniques and tools to perform conventional Boolean logic functions using automata processing. The architecture has been validated using circuit level simulations and evaluated against S-AP [121].

4.2 Hybrid CIM-P architecture

DPP was proposed in 2018 by D. Fujiki, et al. from University of Michigan [37]. 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 Fig. 8 [37]). 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 Shift and Add (S+A), Sample and Hold (S+H), DAC and ADC, a LUT and register file (as shown in the right part of Fig. 8). 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 the general characteristic of CIM-P described in Table 2 and 3, DPP has the following advantages: (i) computations include both logical operations and simple arithmetic operations (i.e., addition, multiplication), (ii) Data can be transferred using both direct and indirect schemes, (iii) the architecture uses non-volatile memory, hence consumes low energy and has a small footprint, (iv) this architecture is claimed to be general purpose, hence it can exploits existing instruction set, compiling techniques and tools, as well as applications. However, it also has the following limitations: (i) the architecture uses non-volatile memory as main memory, which may impact the life time due to limited endurance [17, 127], (ii) as the sense amplifies are complex, a trade-off between area and bandwidth has to be made.



(a) HMC Module with HMC Instruction Large Vector Extensions (HIVE) [3] (b) Resistive GP-SIMD (ReGP) [91]

Fig. 9. Examples of COM-N Architectures

The architecture potential was simulated and evaluated against CPU Intel Xeon E5-2697 using a subset of PARSEC benchmarks [11] and against GPU NVIDIA Titan XP using Rodinia benchmarks [18].

5 COMPUTATION-OUT-MEMORY - NEAR (COM-N)

The COM-N 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 conventional DRAM, embedded DRAM to emerging memory technologies such as RRAM.

Many architectures have been proposed in this class: Vector Intelligent RAM (VIRAM) [58, 67, 69, 98], Active Page(A-Page) [97], Advance Intelligent Memory System (FlexRAM) [62], Modular Reconfigurable Smart Memories (S-Mem) [84], Data-intensive Architecture (DIVA) [24, 25], Hybrid Memory Cube (HMC) [56, 100], Active Memory Cube (AMC) [94], Micron Automata Processor (D-AP) [95], A machine-learning supercomputer (DDN) [20], An Architecture for Accelerated Processing Near Memory (DRAMA) [31], High-Bandwidth Memory (HBM) [59, 74, 83, 119], A Near Data Computing Architecture using Non-Volatile Memory (ProPRAM) [128], Resistive GP-SIMD (ReGP) [91], HMC Instruction Large Vector Extensions (HIVE) [3].

These architectures mainly differ as a consequence of using different technologies. VIRAM, FlexRAM, SM, DIVA, and DRAMA are based on embedded DRAM technology and try to integrate processing units near the main memory; FlexRAM integrates multiple single-core processors with caches, SM a reconfigurable processor, VIRAM and DIVA a vector processor, and DRAMA a Coarse-grain reconfigurable accelerators. A-Page is based on reconfigurable DRAM architecture which integrates conventional DRAM into an FPGA; it implements the reconfigurable logic near the main DRAM memory. HMC, AMC, HBM, and HIVE are based on 3D-stacked DRAM; HMC and HBM support general computing, while AMC and HIVE are optimized for VLIW and vector processing, respectively. DaDianNao, D-AP, ReGP, and ProPRAM utilize logic units that are located near the memory. DaDianNao and D-AP implements a neural network and an automata processor, respectively with very simple logic units inside the conventional DRAM. ReGP integrates a simplified SIMD processor near non-volatile memory. ProPRAM utilizes existing logic units near the non-volatile memory to perform simple computations such as addition and logical operations. As an example, we only describe the HIVE and ReGP architectures next in more details; They are the most recent architectures proposed in COM-N class.

HIVE was proposed in 2016 by M. A. Z. Alves, et al., from Federal University of Rio Grande do Sul [3]. HIVE is a Hybrid Memory Cube (HMC) [56, 100] based architecture that performs large vector operations inside the logic die of a HMC. The architecture consists of a host processor and a HMC module that is extended with a HIVE, as shown in Fig. 9a [3]. 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, 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 Fig. 9a). 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 Fig. 9a). The logic vaults contains a vault controller, write and read buffer, and a DRAM sequencer (as shown in the top right of Fig. 9a). Once the HIVE sequencer receives an instruction, it locks the involved memory address space; if the memory has already been locked, the requested instruction returns a fail status to processor; otherwise, a memory synchronization occurs by flushing related 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 8KB 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 operation 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 characteristic of COM-N described in Table 2 and 3, HIVE comes with the following advantages: (i) the parallelism is high due to vector processing on 8KB of data, (ii) the architecture uses HMC which is mature, commercialized and has some advantages such as high performance, high bandwidth, low power, high density [56, 100]. However, it also has the following limitation that the architecture has a complex HMC module which has a control, communication and programming overhead. The architecture is simulated 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 have also additional processing capacities; for the baseline platforms they are as follows: 1) HMC instructions using HMC 2.0 memory [49] (HMC+HMC), 2) 128-bit SSE instructions with DDR-3 1333 modules (SSE+DDR) and 3) 128-bit SSE instructions with HMC 2.0 (SSE+HMC).

ReGP was proposed in 2016 by A. Morad, et al., from Technion-Israel Institute of Technology [91]. ReGP is a RRAM memory based architecture that exploits data parallelism by attaching a SIMD-like processing unit to the resistive memory, as shown in Figure 9b [91]. The architecture consists of a sequential processor (which is a 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 characteristic of COM-N described in Table 2 and 3, ReGP comes with the following advantages: (i) the parallelism is high due to multiple parallel processing units, (ii) the architecture uses non-volatile memory, hence consumes a low amount of energy and has a small footprint, (iii) the architecture can reuse compilers, programming languages and tools from SIMD architectures. However, it also has the following limitation that the operations within the processing units are simple; complex functions such as floating point operations can cause a high overhead. The architecture is simulated and evaluated against CMOS GP-SIMD [90] using a benchmark for dense matrix multiplications [89].

6 DISCUSSION

This section aims to first evaluate the completeness of the proposed classification. Thereafter, we compare it with existing work in the field. Finally, we discuss the limitations of this work and propose directions for future work.

6.1 Completeness

The proposed classification presented in Fig. 3 is complete and comprehensive. These points can as follows be proven: (i) theoretically, due to the exploration of all the possible classes derived from the classification metrics, and (ii) practically, by mapping all existing memory-centric architectures on the classification.

Theoretically, the classification contains four main classes derived from the "computation location" (first metric); both inside and outside, approximately close or distant from the memory core. Moreover, the second metric consists of both charge-based and non-charge-based memories. Finally, the parallelism metric ranges from instruction, to data and task levels. Each metric is in it self complete, and therefore, the entire classification is complete. The classification does not only contain the existing solutions, but also highlights the potential future solutions that can be further explored (e.g., classes in blue spaces in Fig. 3). Note that hybrid architectures are also covered in this classification. For example, a conventional architecture (COM-F) with accelerator in CIM-P class (e.g., ReAP, ISAAC, CIMA) is considered a hybrid architecture, i.e., a COM-F/CIM-P hybrid.

Practically, it contains an overview of the most existing computer architectures and places them in perspective. In addition, the classification can be used to illustrate the past and future trends (see Fig. 4). Moreover, it clearly depicts a shift from conventional processor-centric architectures towards memory-centric architectures based on emerging technologies (3D stacking, RRAM, etc.).

6.2 Related work

Comparison with traditional/processor-centric architecture classifications: conventional classifications like Flynn's [35], Skillicorn's [117] and Shami-Hemani's [112] classification are quite comprehensive and were considered complete at the time they were published. However, these classifications focus on processor-centric architectures and, hence they can only be used to classify conventional architectures (i.e., architectures in COM-F class). Aside from the above mentioned classifications, some publications on COM-N class have presented intensive architectural reviews [23, 113, 116]. However, they have a restricted focus on near-memory-processing architectures based on 2D, 2.5D, and 3D-stacked DRAM. Signh's classification [116] is the most recent work that provides a review of near-memory computing architectures, i.e. COM-N architectures. It classifies architectures mainly based on the memory hierarchy and processing type (e.g., programmable unit, fixed functional unit and reconfigurable unit). Moreover, it evaluates the architectures based on multiple characteristics of memory, processing, evaluation tools, interoperability, and application domains. However, the classification is not easy to use as the metrics are not systematic. Furthermore, it is not clear if the classification is complete and if it covers all ranges of near-computing architectures. Last but not least, in comparison with the aforementioned classifications, our proposed classification goes one step further to cover both conventional and emerging architectures by having the additional classes CIM-A and CIM-P. Moreover, the proposed classification is so broad that several of its classes are not explored yet. New architectures in these unexplored areas can be easily added to the classification. In addition, our proposed classification uses three selective metrics which create distinctive and easy-to-use terminologies, classes and sub-classes.

Comparison with recent/emerging architecture classifications: recent surveys and classifications for emerging architectures have been proposed by Mittal [88] and Reuben [105]. Mittal's classification only tries to link architectures with their applications. Specifically, the classification discusses three unconventional architectures: processing-in-memory, machine learning and neural network based architectures using RRAM. They mostly focus on applications containing dot-product operations in the RRAM crossbar. This classification is not complete, as RRAM in particular and emerging memory technology in general can also be used to implement other functions such as bitwise logic operations [80, 134], arithmetic operations using implication logic [71, 114], Boolean logic[118, 133], etc. Reuben's classification classifies existing resistive logic design methods into three classes: in-memory, near-memory and out-of-memory computing. The near-memory class has three sub-classes without identifiers (e.g., they are based on how data moves out of the memory array; this includes data movements (1) between consecutive logic levels, (2) for computing each Sum-of-Products, (3) for computing each logic gate). This classification, however, tries to redefine the terminologies without defining clear generic metrics for each class. Instead, each class uses different criteria to distinguish between their sub-classes. Therefore, it is not a systematic and comprehensive classification, which makes it difficult to use in identifying and exploring architectures. Moreover, it is difficult to judge if the classification is complete. Furthermore, the classification focuses only on resistive memories, while other emerging memory technologies are also promising. Overall, both Mittal and Reuben classifications are not complete and comprehensive enough to classify all architectures.

6.3 Future directions and challenges

Memory-enteric computing is seen as one of the promising solution to alleviate (even if partially) the memory bottleneck. Not only the communication between the processing core and the main memory will be significantly reduced, but also the energy consumption; the data communication on its own is extremely energy consuming. Implementing CIM based on DRAM or emerging memristive devices seems to be more realistic than using on chip SRAMs. Although SRAM technology is more CMOS compatible when it comes to manufacturing, the cost per bit for such technology is much higher than that of other memory technologies. Hence, the overall cost of large capacity SRAMs (which is needed for CIM) is by far much higher; for the same capacity, SRAM consumes much more area/power compared to DRAM and non-volatile memories. In addition, the two main directions that are currently explored are CIM-A and CIM-P, in which CIM-P is more feasible than CIM-A due to the complex underlining memory technology. CIM-P requires less effort and modification in the memory core (mainly in the periphery). Moreover, CIM architectures do not make conventional architectures obsolete; in fact, multicore architectures with caches are relevant for applications with high data locality, while CIM architectures can be only used efficiently for certain specific applications [106]. Furthermore, building appropriate simulators and tools for CIM architectures based on technology calibrated models will enable a real estimation of the potential of such architectures [6, 39, 76, 135].

It is worth mentioning that the focus of this paper is to propose a unified terminology and classification instead of presenting a survey. In our future work we will present a survey that intensively discusses all architectures.

7 CONCLUSION

In this paper, we have proposed a classification using three metrics: computational location, memory technology and level of parallelism. We have used the most important metric, i.e. computational location, to describe and evaluate the four main classes (and the selected architectures therein). The work shows that architectures do not only require to be memory bottleneck free, but also energy and area efficient. In order to accomplish that, the architectures must be implemented with the right technologies. The relationship and dependency between the architecture and technologies becomes stronger for memory-centric computing architectures. This work also showed that new architectures typically emerge after new technology developments (e.g., introduction of 3D stacking and RRAM). Our classification unifies the prior work and aims to provide a comprehensive and unique terminology for memory-centric computing architectures. Finally, the classification does not only present an overview of existing architectures, but also predicts the potential of future architecture variants, including hybrid architectures that may combine different strengths of the different classes.

ACKNOWLEDGMENTS

The results presented in this paper have been obtained in the framework 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.

REFERENCES

[1] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das. 2017. Compute caches. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on.* IEEE, 481–492.

[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3 (2016), 105–117.

[3] M. A. Alves, M. Diener, P. C. Santos, and L. Carro. 2016. Large vector extensions inside the HMC. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016.* IEEE, 1249–1254.

[4] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux. 2015. SiNUCA: A Validated Micro-Architecture Simulator.. In *HPCC/CSS/ICESS.* 605–610.

[5] L. Amarú, P.-E. Gaillardon, and G. De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS).*

[6] A. BanaGozar, K. Vadivel, S. Stuijk, H. Corporaal, S. Wong, M. A. Lebdeh, J. Yu, and S. Hamdioui. 2019. CIM-SIM: computation in Memory SIMuIator. In *International Workshop on Software and Compilers for Embedded Systems.* ACM, 1–4.

[7] J. Barth, D. Plass, E. Nelson, C. Hwang, G. Fredeman, M. Sperling, A. Mathews, T. Kirihata, W. R. Reohr, K. Nair, et al. 2010. A 45 nm SOI embedded DRAM macro for the POWERâĎć processor 32 MByte on-chip L3 cache. *IEEE Journal of Solid-State Circuits* 46, 1 (2010), 64–75.

[8] G. Benson, Y. Hernandez, and J. Loving. 2013. A bit-parallel, general integer-scoring sequence alignment algorithm. In *Annual Symposium on Combinatorial Pattern Matching.* Springer, 50–61.

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

[10] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. Piramanayagam. 2017. Spintronics based random access memory: a review. *Materials Today* (2017).

[11] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques.* ACM, 72–81.

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[13] E. Bolotin, D. Nellans, O. Villa, M. O'Connor, A. Ramirez, and S. W. Keckler. 2015. Designing efficient heterogeneous memory architectures. *IEEE Micro* 35, 4 (2015), 60–68.

[14] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams. 2010. Memristive switches enable stateful logic operations via material implication. *Nature* 464, 7290 (2010), 873–876.

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

[16] R. Cabrera, E. Merced, and N. Sepúlveda. 2013. A micro-electro-mechanical memory based on the structural phase transition of VO2. *physica status solidi (a)* 210, 9 (2013), 1704–1711.

[17] M.-F. Chang, C.-H. Chuang, M.-P. Chen, L.-F. Chen, H. Yamauchi, P.-F. Chiu, and S.-S. Sheu. 2012. Endurance-aware circuit designs of nonvolatile logic and nonvolatile SRAM using resistive memory (memristor) device. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 329–334.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 44–54.

[19] 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 Transactions on Magnetics* 46, 6 (2010), 1873–1878.

[20] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[21] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. 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.

[22] G. Conte, S. Tommesani, and F. Zanichelli. 2000. The long and winding road to high-performance image processing with MMX/SSE. In *Computer Architectures for Machine Perception, 2000. Proceedings. Fifth IEEE International Workshop on*. IEEE, 302–310.

[23] J. P. C. de Lima, P. C. Santos, M. A. Z. Alves, A. C. S. Beck, and L. Carro. 2018. Design Space Exploration for PIM architectures in 3D-stacked memories. In *Computer Frontier*. ACM, 295–308.

[24] J. Draper, J. T. Barrett, J. Sondeen, S. Mediratta, C. W. Kang, I. Kim, and G. Daglikoca. 2005. A prototype processing-in-memory (PIM) chip for the data-intensive architecture (DIVA) system. *Journal of VLSI signal processing systems for signal, image and video technology* 40, 1 (2005), 73–84.

[25] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, et al. 2002. The architecture of the DIVA processing-in-memory chip. In *Proceedings of the 16th international conference on Supercomputing*. ACM, 14–25.

[26] H. Du Nguyen, J. Yu, L. Xie, M. Taouil, S. Hamdioui, and D. Fey. 2017. Memristive devices for computing: Beyond CMOS and beyond von Neumann. In *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 1–10.

[27] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels. 2017. On the implementation of computation-in-memory parallel adder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 8 (2017), 2206–2219.

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

[29] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *arXiv preprint arXiv:1805.03718* (2018).

[30] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen. 1997. Simultaneous multithreading: A platform for next-generation processors. *IEEE micro* 17, 5 (1997), 12–19.

[31] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. 2015. DRAMA: An architecture for accelerated processing near memory. *IEEE Computer Architecture Letters* 14, 1 (2015), 26–29.

[32] T. Finkbeiner, G. Hush, T. Larsen, P. Lea, J. Leidel, and T. Manning. 2017. In-Memory Intelligence. *IEEE Micro* 37, 4 (2017), 30–38.

[33] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo. 2008. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel white paper* 19 (2008), 20.

[34] R. J. Fisher. 2003. General-purpose SIMD within a register: Parallel processing on consumer microprocessors. (2003).

[35] M. Flynn. 1966. Very high-speed computing systems. *Proc. IEEE* 54, 12 (Dec 1966), 1901–1909. https://doi.org/10.1109/PROC.1966.5273

[36] G. Fuchs, N. Emley, I. Krivorotov, P. Braganca, E. Ryan, S. Kiselev, J. Sankey, D. Ralph, R. Buhrman, and J. Katine. 2004. Spin-transfer effects in nanoscale magnetic tunnel junctions. *Applied Physics Letters* 85, 7 (2004), 1205–1207.

[37] D. Fujiki, S. Mahlke, and R. Das. 2018. In-Memory Data Parallel Processor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1–14.

[38] P.-E. Gaillardon, L. Amar, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli. 2016. The programmable logic-in-memory (PLiM) computer. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 427–432.

[39] M. Gao, G. Ayers, and C. Kozyrakis. 2015. Practical near-data processing for in-memory analytics frameworks. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 113–124.

[40] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem. 2006. Introduction to Intel Core Duo Processor Architecture. *Intel Technology Journal* 10, 2 (2006).

[41] J. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu, Y. S. Shin, et al. 2007. A 160-kilobit molecular electronic memory patterned at 10 11 bits per square centimetre. *Nature* 445, 7126 (2007), 414.

[42] B. Halg. 1990. On a micro-electro-mechanical nonvolatile memory cell. *IEEE Transactions on Electron Devices* 37, 10 (1990), 2230–2236.

[43] S. Hamdioui, K. L. M. Bertels, and M. Taouil. 2017. Computing device for âĂIJbig dataâĂİ applications using memristors. US Patent 9,824,753.

[44] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels. 2017. Memristor for computing: Myth or reality?. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 722–731.

[45] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1718–1725.

[46] J. Han, C.-S. Park, D.-H. Ryu, and E.-S. Kim. 1999. Optical image encryption based on XOR operations. *Optical Engineering* 38, 1 (1999), 47–55.

[47] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels. 2016. Parallel matrix multiplication on memristor-based computation-in-memory architecture. In *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 759–766.

[48] J. L. Hennessy and D. A. Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

[49] HMC. 2018. Hybrid Memory Cube Specification 2.1. http://hybridmemorycube.org/

[50] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, et al. 2005. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 459–462.

[51] R. B. Hur and S. Kvatinsky. 2016. Memristive memory processing unit (MPU) controller for in-memory processing. In *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 1–5.

[52] IBM. 2014. Power 4 - The First Multi-Core, 1GHz Processor.

[53] ITRS. 2010. ITRS ERD report. http://www.itrs.net

[54] S. S. Iyer and H. L. Kalter. 1999. Embedded DRAM technology: opportunities and challenges. *IEEE spectrum* 36, 4 (1999), 56–64.

[55] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan. 2017. Computing in memory with spin-transfer torque magnetic RAM. *arXiv preprint arXiv:1703.02118* (2017).

[56] J. Jeddeloh and B. Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 87–88.

[57] Z. Jianwu, Z. Danying, et al. 2008. Survey on microprocessor architecture and development trends. In *Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on*. IEEE, 297–300.

[58] D. Judd, K. Yelick, C. Kozyrakis, D. Martin, and D. Patterson. 2001. Exploiting on-chip memory bandwidth in the VIRAM compiler. In *Intelligent Memory Systems*. Springer, 122–134.

[59] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *Memory Workshop (IMW), 2017 IEEE International*. IEEE, 1–4.

[60] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. 2010. Power7: IBM's next-generation server processor. *IEEE micro* 30, 2 (2010), 7–15.

[61] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. [n. d.]. FlexRAM: Toward an advanced Intelligent Memory system. In *2012 IEEE 30th International Conference on Computer Design (ICCD)* (2012). 5–14. https://doi.org/10.1109/ICCD.2012.6378608

[62] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. 2012. FlexRAM: Toward an advanced intelligent memory system. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 5–14.

[63] D. Keitel-Schulz and N. Wehn. 1998. Issues in embedded DRAM development and applications. In *Proceedings of the 11th international symposium on System synthesis*. IEEE Computer Society, 23–31.

[64] D. Keitel-Schulz and N. Wehn. 2001. Embedded DRAM development: Technology, physical design, and application issues. *IEEE Design & Test of Computers* 18, 3 (2001), 7–15.

[65] K. Kim, S. Shin, and S.-M. Kang. 2011. Stateful logic pipeline architecture. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2497–2500.

[66] D. Kirk et al. 2007. NVIDIA CUDA software and GPU parallel computing architecture. In *ISMM*, Vol. 7. 103–104.

[67] C. Kozyrakis. 2002. *Scalable vector media-processors for embedded systems*. Technical Report. CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV.

[68] C. Kozyrakis and D. Patterson. 2002. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society Press, 283–293.

[69] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, et al. 1997. Scalable processors in the billion-transistor era: IRAM. *Computer* 30, 9 (1997), 75–78.

[70] N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, P. Mosalikanti, M. Neidengard, A. Deval, A. Khanna, et al. 2014. Haswell: A family of IA 22 nm processors. *IEEE Journal of Solid-State Circuits* 50, 1 (2014), 49–58.

[71] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. 2014. MAGIC–Memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs* 61, 11 (2014), 895–899.

[72] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. 2014. Memristor-based material implication (IMPLY) logic: design principles and methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 10 (2014), 2054–2066.

[73] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. 2010. Phase change memory architecture and the quest for scalability. *Commun. ACM* 53, 7 (2010), 99–106.

[74] J. C. Lee, J. Kim, K. W. Kim, Y. J. Ku, D. S. Kim, C. Jeong, T. S. Yun, H. Kim, H. S. Cho, Y. O. 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 *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 318–319.

[75] E. Lehtonen, J. H. Poikonen, and M. Laiho. 2014. Memristive stateful logic. In *Memristor Networks*. Springer, 603–623.

[76] J. D. Leidel and Y. Chen. 2016. Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 621–630.

[77] C. Li, W. Fan, B. Lei, D. Zhang, S. Han, T. Tang, X. Liu, Z. Liu, S. Asano, M. Meyyappan, et al. 2004. Multilevel memory based on molecular devices. *Applied Physics Letters* 84, 11 (2004), 1949–1951.

[78] C. Li, D. Zhang, X. Liu, S. Han, T. Tang, C. Zhou, W. Fan, J. Koehne, J. Han, M. Meyyappan, et al. 2003. Fabrication approach for molecular memory arrays. *Applied physics letters* 82, 4 (2003), 645–647.

[79] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie. 2017. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 288–301.

[80] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*. IEEE.

[81] 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.

[82] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri. 2003. A VLIW processor with reconfigurable instruction set for embedded applications. *IEEE Journal of solid-state circuits* 38, 11 (2003), 1876–1886.

[83] J. Macri. 2015. AMD's next generation GPU and high bandwidth memory architecture: FURY. In *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 1–26.

[84] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. 2000. Smart memories: A modular reconfigurable architecture. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 161–171.

[85] A. Maislos et al. 2011. A new era in embedded flash memory. *Flash memory summit* (2011).

[86] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens. 2002. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM Journal of Research and Development* 46, 2.3 (2002), 187–212.

[87] P. Marcuello, A. González, and J. Tubella. 1998. Speculative multithreaded processors. In *Proceedings of the 12th international conference on Supercomputing*. ACM, 77–84.

[88] S. Mittal. 2018. A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks. *Machine Learning and Knowledge Extraction* 1, 1 (2018). https://doi.org/10.3390/make1010005

[89] A. Morad, L. Yavits, and R. Ginosar. 2014. Efficient dense and sparse Matrix multiplication on GP-SIMD. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*. IEEE, 1–8.

[90] A. Morad, L. Yavits, and R. Ginosar. 2015. GP-SIMD processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4 (2015), 53.

[91] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar. 2016. Resistive GP-SIMD processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4 (2016), 57.

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

[93] R. Nair. 2015. Evolution of memory architecture. *Proc. IEEE* 103, 8 (2015), 1331–1345.

[94] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. Costa, J. Doi, C. Evangelinos, et al. 2015. Active memory cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development* 59, 2/3 (2015), 17–1.

[95] H. Noyes et al. 2014. MicronâĂŹs automata processor architecture: Reconfigurable and massively parallel automata processing. In *Proc. of Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*.

[96] NVIDIA. 2012. Tesla K20X GPU Accelerator Board Specification.

[97] M. Oskin, F. T. Chong, and T. Sherwood. 1998. *Active pages: A computation model for intelligent memory*. Vol. 26. IEEE Computer Society.

[98] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. 1997. A case for intelligent RAM. *IEEE micro* 17, 2 (1997), 34–44.

[99] D. A. Patterson. 2006. Future of computer architecture. In *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*.

[100] J. T. Pawlowski. 2011. Hybrid memory cube (HMC). In *Hot Chips 23 Symposium (HCS), 2011 IEEE*. IEEE, 1–24.

[101] A. Peleg and U. Weiser. 1996. MMX technology extension to the Intel architecture. *IEEE micro* 16, 4 (1996), 42–50.

[102] M. Radosavljević, M. Freitag, K. Thadani, and A. Johnson. 2002. Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors. *Nano Letters* 2, 7 (2002), 761–764.

[103] R. Ramanathan. 2006. Intel® Multi-Core Processors. *Making the Move to Quad-Core and Beyond* (2006).

[104] S. Raoux, F. Xiong, M. Wuttig, and E. Pop. 2014. Phase change materials and phase change memory. *MRS bulletin* 39, 8 (2014), 703–710.

[105] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky. 2017. Memristive logic: A framework for evaluation and comparison. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017 27th International Symposium on*. IEEE, 1–8.

[106] M. T. A. S. M. L. G. S. P. S. S. F. C. S. D. F. G. R. G. K. A. R. L. B. Said Hamdioui, Hoang Anh Du Nguyen. 2019. Applications of computation-in-memory architectures based on memristive devices. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 486–491.

[107] G. S. Sandhu. 2013. Emerging memories technology landscape. In *Non-Volatile Memory Technology Symposium (NVMTS), 2013 13th*. IEEE, 1–5.

[108] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 422–433.

[109] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. 2015. Fast bulk bitwise AND and OR in DRAM. *IEEE Computer Architecture Letters* 14, 2 (2015), 127–131.

[110] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 273–287.

[111] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[112] M. Shami and A. Hemani. 2012. Classification of Massively Parallel Computer Architectures. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International.* 344–351. https://doi.org/10.1109/IPDPSW.2012.42

[113] P. Siegl, R. Buchty, and M. Berekovic. 2016. Data-centric computing frontiers: A survey on processing-in-memory. In *Proceedings of the Second International Symposium on Memory Systems.* ACM, 295–308.

[114] A. Siemon, S. Menzel, A. Chattopadhyay, R. Waser, and E. Linn. 2015. In-memory adder functionality in 1S1R arrays. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 1338–1341.

[115] A. Siemon, S. Menzel, R. Waser, and E. Linn. 2015. A complementary resistive switch-based crossbar array adder. *IEEE journal on emerging and selected topics in circuits and systems* 5, 1 (2015), 64–74.

[116] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra. 2018. A Review of Near-Memory Computing Architectures: Opportunities and Challenges. In *Proceedings of the 21st Euromicro Conference on Digital System Design (DSD).*

[117] D. Skillicorn. 1988. A taxonomy for computer architectures. *Computer* 21, 11 (Nov 1988), 46–57. https://doi.org/10.1109/2.86786

[118] G. Snider. 2005. Computing with hysteretic resistor crossbars. *Applied Physics A: Materials Science & Processing* 80, 6 (2005), 1165–1172.

[119] K. Sohn, W.-J. Yun, R. Oh, C.-S. Oh, S.-Y. Seo, M.-S. Park, D.-H. Shin, W.-C. Jung, S.-H. Shin, J.-M. 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 Journal of Solid-State Circuits* 52, 1 (2017), 250–260.

[120] H. S. Stone. 1970. A logic-in-memory computer. *IEEE Trans. Comput.* 100, 1 (1970), 73–78.

[121] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das. 2017. Cache Automaton. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17).* ACM, New York, NY, USA, 259–272. https://doi.org/10.1145/3123939.3123986

[122] J. Suh, E.-G. Kim, S. P. Crago, L. Srinivasan, and M. C. French. 2003. A performance analysis of PIM, stream processing, and tiled processing on memory-intensive signal processing kernels. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 410–421.

[123] M. R. Thistle and B. J. Smith. 1988. A processor architecture for Horizon. In *Supercomputing'88.[Vol. 1]., Proceedings.* IEEE, 35–41.

[124] D. M. Tullsen, S. J. Eggers, and H. M. Levy. 1995. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, Vol. 23. ACM, 392–403.

[125] M. Vestias and H. Neto. 2014. Trends of CPU, GPU and FPGA for high-performance computing. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on.* IEEE, 1–6.

[126] B. Wang, M. Torres, D. Li, J. Zhao, and F. Rusu. 2016. Performance Implications of Processing-in-Memory Designs on Data-Intensive Applications. In *Parallel Processing Workshops (ICPPW), 2016 45th International Conference on.* IEEE, 115–122.

[127] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi. 2014. Endurance-aware cache line management for non-volatile caches. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 1 (2014), 4.

[128] Y. Wang, Y. Han, L. Zhang, H. Li, and X. Li. 2015. ProPRAM: exploiting the transparent logic resources in non-volatile memory for near data computing. In *Proceedings of the 52nd Annual Design Automation Conference.* ACM, 47.

[129] R. Waser. 2012. Redox-based resistive switching memories. *Journal of nanoscience and nanotechnology* 12, 10 (2012), 7628–7640.

[130] R. Waser and M. Aono. 2007. Nanoionics-based resistive switching memories. *Nature materials* 6, 11 (2007), 833.

[131] S. Wong, T. Van As, and G. Brown. 2008. ρ-VEX: A reconfigurable and extensible softcore VLIW processor. In *ICECE Technology, 2008. FPT 2008. International Conference on.* IEEE, 369–372.

[132] W. A. Wulf and S. A. McKee. 1995. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[133] L. Xie, H. A. D. Nguyen, M. Taouil, and K. Bertels Said Hamdioui. 2015. Fast boolean logic mapped on memristor crossbar. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on.* IEEE, 335–342.

[134] L. Xie, H. A. D. Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui. 2017. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI).* IEEE, 335–340.

[135] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li. 2018. PIMSim: A Flexible and Detailed Processing-in-Memory Simulator. *IEEE Computer Architecture Letters* 18, 1 (2018), 6–9.

[136] J. J. Yang, D. B. Strukov, and D. R. Stewart. 2013. Memristive devices for computing. *Nature nanotechnology* 8, 1 (2013), 13–24.

[137] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar. 2015. Resistive Associative Processor. *CAL* (2015).

[138] J. Yu, L. Xie, M. Taouil, and S. Hamdioui. 2018. Memristive Devices for Computation-In-Memory. In *Design, Automation and Test in Europe DATE.*

[139] S. Yu and P.-Y. Chen. 2016. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine* 8, 2 (2016), 43–56.

[140] J.-G. Zhu. 2008. Magnetoresistive random access memory: The path to competitiveness and scalability. *Proc. IEEE* 96, 11 (2008), 1786–1798.