A Survey of Memory-Centric Energy Efficient Computer Architecture

Changwu Zhang, Hao Sun, Shuman Li, Yaohua Wang, Haiyan Chen, and Hengzhu Liu

(Survey)

***Abstract*—Energy efficient architecture is essential to improve both the performance and power consumption of a computer system. However, modern computers suffer from the severe "memory wall" problem due to the significant performance gap between the processor technology and the memory technology. Thus, the computer architecture community is evolving from compute-centric to memory-centric designs to reduce the data movement overhead. This paper presents a comprehensive survey of the main challenges and recent advances in memory-centric energy efficient computer architecture. We summarize two research directions: improving the memory technology and processing closer to memory. The former focuses on optimizing the conventional memory technology and exploiting emerging non-volatile memory (NVM) technology. The latter talks about currently popular processing in memory (PIM) technology, including near-memory processing (NMP) and in-memory processing (IMP). Moreover, some other topics like hardware for machine learning (ML), ML for hardware, security, privacy, and reliability are gaining increasing attention and should be considered seriously in the design phase of a computer system. The community is facing various challenges and opportunities simultaneously, requiring researchers to have a more comprehensive understanding of this field which is also the goal of this paper.**

***Index Terms*—Energy efficiency, computer architecture, non-volatile memory (NVM), processing in memory (PIM), memory wall.**

I. Introduction

ENERGY efficiency is attracting increasing attention in the computer architecture community. As the feature size scaling of transistors gradually reaches the physical limit, Moore's Law has slowed down, and Dennard scaling has failed from the severe leakage power dissipation. Moreover, modern computers face the severe "memory wall" challenge due to the performance gap between processing and memory units. Considering that the memory technology develops more slowly than the processor technology, although the processing unit has a high performance, it still has to wait for the memory operations. The community calls for new technologies and novel architectures to improve the system performance and maintain the power dissipation level.

As the "memory wall" problem gradually becomes the bottleneck of the von Neumann architecture, the community has turned from compute-centric to memory-centric designs with an emphasis on dealing with memory-side issues. Roughly speaking, recent progress in memory-centric energy efficient computer architecture falls into two directions: optimizing the memory technology and designing a novel architecture. The former is committed to optimizing the conventional memory technology and exploiting non-volatile memory (NVM) technology in the memory hierarchy. The latter focuses on placing the processing closer to memory, and the in/near-memory processing (IMP/NMP) technology belongs to this domain. Both two directions contribute to reducing the overhead of data movement.

Resistive emerging NVM (ENVM) technologies, such as Phase Change Memory (PCM) [1], Spin Transfer Torque RAM (STT-RAM) [2], and Resistive RAM (RRAM or ReRAM) [3], have zero standby power because of non-volatility, showing inherent energy efficiency advantages. ENVMs leverage resistance states to represent bit information, making them potentially re-architect the memory hierarchy. This paper distinguishes ENVMs from Flash-based NVM. Resistive ENVMs share several common advantages like lower access latency ($\sim$1000x) and energy ($\sim$1000x) and higher write endurance ($>$1000x) compared to Flash [4]. Moreover, ENVMs have DRAM-comparable performance (e.g., nanosecond latency) [4]. Recently, a commercial ENVM product has been available with the release of Intel Optane DC Persistent Memory (PM) DIMM [5], based on the 3D XPoint [6] technology developed jointly by Intel and Micron companies. Yang et al. [7] find that the read latency for Optane DC (169$\sim$305 ns) seen by software is 2x$\sim$3x higher than DRAM (81$\sim$101 ns) due to Optane's longer media latency, while they share similar write latency (60 $\sim$ 90 ns).

As shown in Fig. 1, according to the distance between processing and memory units, this paper classifies modern computer architecture into three categories: off-memory processing (OMP), near-memory processing (NMP), and in-memory processing (IMP). The OMP architecture separates processing units (PUs) and memory units (MUs) into different chips with the data to be transferred through the off-chip interface. The NMP architecture integrates PUs and MUs in the same chip

Manuscript received 15 May 2022; revised 5 June 2023; accepted 18 July 2023. Date of publication 21 July 2023; date of current version 4 August 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62172430 and Grant 62272477, and in part by the Key Laboratory Fund, National University of Defense Technology under Grant WDZC20215250109. Recommended for acceptance by M. Becchi. *(Corresponding author: Yaohua Wang.)*

The authors are with the College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China (e-mail: zhangchangwu@nudt.edu.cn; sunhao1996@nudt.edu.cn; lishuman13@nudt.edu.cn; nudtyh@foxmail.com; hychen608@163.com; hengzhuliu@nudt.edu.cn).

Digital Object Identifier 10.1109/TPDS.2023.3297595

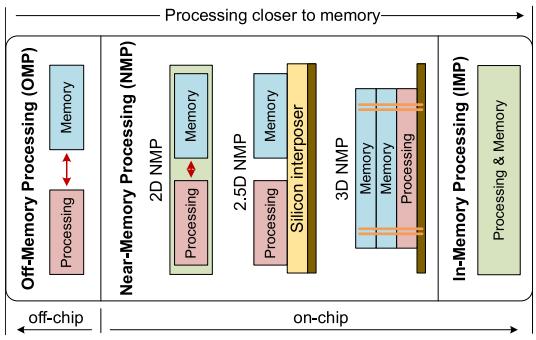


Fig. 1. A taxonomy of modern computer architecture according to the distance between processing and memory units.

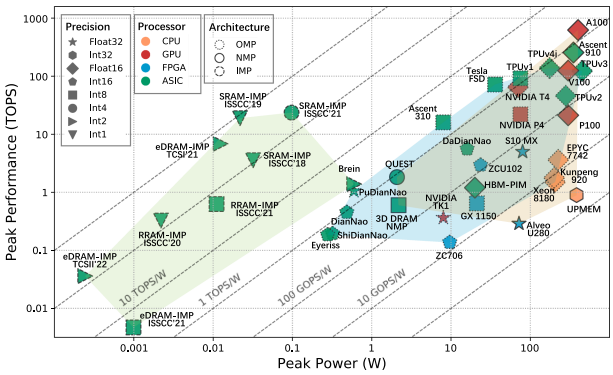


Fig. 2. Performance and power comparison among different OMP, NMP and IMP chips. Raw data and references are available on GitHub [10].

to shorten the distance of data transferring. NMP has three prototypes: 2D, 2.5D, and 3D structures. IMP utilizes the memory to perform in-situ (in-place) processing without data movement, implementing the data processing and storing functions in the same device. Such classification provides a clear boundary for different memory-centric energy efficient techniques. NMP and IMP are also referred to as the processing in memory (PIM) technology in other papers, but we distinguish them in this survey.

From Fig. 2, we can find it is common for IMP accelerators to obtain energy efficiency over 1 TOPS/W, outperforming OMP and NMP architecture. One reason is IMP accelerators usually perform low-precision ($\leq$8-bit) computation, and another reason is the IMP architecture eliminates a large portion of data movement. Meanwhile, NMP architecture usually has higher peak performance than OMP with the benefit of the reduced distance between processing and memory units. To some degree, current IMP designs are more suitable for embedded/mobile markets, whose advantages are lower power and higher energy efficiency. NMP architecture has better programmability and achieves higher performance at a cost of higher power consumption, making it more suitable for server/cloud markets.

There have been some survey papers about how to mitigate the memory wall problem. Mittal et al. [4] propose a survey about software techniques for utilizing NVM in the secondary storage and main memory levels but lack works about NVM in the cache and register levels. Mutlu et al. [8] give a modern primer on the PIM technology but concentrate on the review of DRAM-based NMP and IMP. Xiao et al. [9] review the NVM-based analog IMP architectures but only list related works for neural network acceleration. In short, the existing surveys usually have a restricted focus on only one or several parts of the memory-centric architectural techniques. This paper presents a comprehensive survey of the memory-centric architecture with an emphasis on improving system-level energy efficiency. To strike a balance between breadth and brevity, we focus more on NMP and IMP. Considering the community is developing rapidly with a large number of works proposed every year, we have created a repository on GitHub [10] and will keep updating a detailed research list for each topic.

The remainder of this paper is organized as follows. Section II talks about energy efficient memory techniques of the OMP architecture, including optimizing conventional memory technology (e.g., DRAM and cache) and exploiting emerging NVM technology. Section III introduces recent advances in the NMP architecture where 2D, 2.5D and 3D NMP techniques are discussed separately. In Section IV, we present a holistic survey of the IMP architecture implemented based on SRAM, DRAM, and NVM, respectively. Section V presents a discussion about the programming model of NMP/IMP. In Section VI, we discuss the future research trend of energy efficient computer architecture. Finally, Section VII gives a conclusion about this paper. Table I shows an overview of memory-centric energy efficient techniques in the order of OMP, NMP, and IMP architectures.

II. Off-Memory Processing

Traditional von Neumann architecture-based off-memory processing (OMP) is undergoing the severe "memory wall" problem. There are two directions to improve the energy efficiency of the OMP system: optimizing conventional memory technology and exploiting emerging NVM technology.

A. Optimizing Conventional Memory Technology

1) Energy Efficient DRAM Techniques: The energy efficiency of DRAM is constrained by three key factors: *refresh operations, latency, and bandwidth.* DRAM requires periodical refresh operations to retain the data saved in the capacitor. To **reduce refresh operations**, researchers introduce several refresh-aware techniques, such as retention-aware refresh [11], [12], access-refresh parallelization [13], half-page refresh scheduling [14] and zero value-aware refresh elimination [15]. DRAM access latency is affected by many variations (e.g., manufacturing process, operating voltages and temperatures), requiring worst-case timing parameters to guarantee the correct operation. Mutlu et al. [8] conduct a survey on exploiting various latency variations to **reduce the DRAM latency**.

Although the industry has proposed some high-bandwidth DRAM standards such as GDDR, HBM, and WideIO, the demand for **higher bandwidth** is still urgent. Various architectural techniques have been proposed, including increasing the subchannel count with reduced row activation granularity [16],

TABLE I
AN OVERVIEW OF MEMORY-CENTRIC ENERGY EFFICIENT TECHNIQUES

Topics	Classification	Techniques	Reference
Off-Memory Processing (OMP)			
Optimizing conventional memory technology	Energy efficient DRAM	Refresh / Latency / Bandwidth	[11], [12], [13], [14], [15] / [8] / [16], [17], [18], [19], [20]
	Energy efficient cache	General-purpose cache optimization	[21], [22], [23] / [24], [25], [26] / [27] / [28]
		Mitigating cache contention	[29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39]
Exploiting emerging NVM technology	Storage level	ENVM as storage	[1], [3], [40]
	Main memory level	ENVM as main memory	[41], [42], [43], [44], [45], [46], [47], [48], [49], [50]
	Cache level	ENVM as cache	[2], [51], [52], [53], [54]
	Register level	ENVM as register file	[55], [56], [57], [58], [59], [60], [61], [62]
Near-Memory Processing (NMP)			
2D NMP	DRAM-based 2D NMP	DIMM (Centralized/Distributed)	[63], [64], [65], [66], [67], [68], [69] / [70], [71]
		Memory controller / Component	[72], [73] / [74], [75], [76], [77], [78]
	SSD-based NDP	Processing in the SSD controller	[79], [80], [81], [82], [83], [84]
2.5D NMP	HBM-based NMP	Accelerator near/in HBM	[85], [86], [87], [88] / [89], [90], [91]
3D NMP	SRAM-based 3D NMP	Array/Device-level 3D integration	[92], [93], [94] / [95], [96]
	DRAM-based 3D NMP	DRAM dies + logic die	[97], [98], [99], [100], [101], [102], [103], [104]
	ENVM-based 3D NMP	ENVM dies + logic die	[105], [106], [107], [108]
In-Memory Processing (IMP)			
In-SRAM processing	In-SRAM accelerator	Dot product / circuit non-idealities	[109], [110], [111], [112] / [113], [114], [115], [116]
	In-cache computing	Simple/Complex/Programmability	[117], [118] / [119], [120] / [121], [122]
In-DRAM processing	In-DRAM data movement	Intra-subarray, inter-subarray	[123], [124], [125], [126], [127]
	Simple bulk bitwise operations	Charge sharing	[128], [129], [130], [131], [132]
	Complex arithmetic operations	Special computing paradigm	[133], [134], [135], [136], [137], [138], [139]
In-NVM processing	In-Flash processing	NOR/NAND Flash	[140], [141] / [142], [143]
	In-ENVM processing	Memristive ANN/SNN accelerators	[144], [145], [146] / [147], [148], [149], [150]
		Mitigating non-idealities	[151], [152], [153], [154], [155]
		Other potential applications	[156], [157], [158], [159], [160], [161]

dividing the DRAM die into many independent units [17] and enabling simultaneous multi-layer access [18]. Besides, 3D-stacking technology has been utilized to increase memory bandwidth by stacking DRAM devices upon the host processor [19], [20].

2) Energy Efficient Cache Techniques: Increasing the cache hit rate and reducing the access latency are two main challenges to optimizing the cache subsystem. General-purpose solutions include *microarchitecture tuning, cache compression, prefetching, and load-related prediction.* **Microarchitecture tuning** [21] is the process of determining the best cache configuration (e.g., total size, line size, and associativity). Moreover, considering that 3D architecture helps in increasing area utilization, SRAM-based [22] and DRAM-based [23] 3D caches are introduced to improve the cache capacity and reduce the latency. **Cache compression** can store data in a smaller size [28], giving the impression of providing a larger effective cache capacity, finally reducing miss rates and improving bandwidth utilization. However, compression and corresponding decompression steps will incur additional latency and energy overheads, leaving compression techniques as open research questions. **Cache prefetching** [27] issues a fetch to the memory system in advance rather than waiting for the arrival of a cache miss, requiring prediction of future access patterns. However, prefetching useless lines will lead to cache pollution and bandwidth waste. **Load-related prediction** techniques including *off-chip load prediction* [24] and *load value prediction* [25], [26], can reduce the latency caused by a cache miss. Off-chip load prediction [24] is used to predict whether a load would hit the cache. Load value prediction [25] techniques maintain the execution history of a load instruction to predict its possible outcome in future execution. The predicted value will be sent to dependent instructions so that the pipeline can proceed speculatively without a stall.

Cache contention, as a severe issue for modern many-core processors (e.g., GPUs), will increase cache miss rates. Common solutions include *thread throttling, cache bypassing, cache partitioning and cache locking.* **Thread throttling** reduces the number of threads contending for caches, finally improving cache hit rates. However, fewer threads will also under-utilize shared resources like NoC and DRAM bandwidth. Researchers have introduced priority-aware thread grouping [29], compiler-based register re-allocation [30] and source-based fairness control [31] techniques to solve this problem. **Cache bypassing** will bypass some access requests to protect hot lines from early eviction, which relies on well-designed bypassing algorithms [32], [33], [34], [35], [36], [37]. **Cache partitioning** [38] can adaptively divide the cache into partitions and assign appropriate partitions to cores on demand, effectively avoiding inter-core interference. **Cache locking** [39] relies on hardware features to flag a given cache line or way as locked, thus preventing its content from being evicted until a successive unlock operation is performed.

B. Exploiting Emerging NVM Technology

ENVM devices can completely replace or only co-work with the original devices at one level of the memory hierarchy, i.e., storage, main memory, cache or register level. For instance, the Intel Optane DC Persistent Memory DIMMs can act as either a main memory device without persistence (Memory mode) or a persistent storage device (App Direct mode) [5]. The *challenge* is how to integrate ENVMs into different levels of the memory hierarchy considering different requirements (e.g., access speed, endurance, or energy consumption) to achieve a tradeoff between their advantages and disadvantages.

1) Storage Level: The storage-class memory (SCM) utilizes ENVM technology to uniform the secondary storage and main

memory into one subsystem. SCM should exhibit the characteristics of random accessibility, non-volatility, low access latency, low cost per bit, low standby energy, and high durability. Many researchers have employed the ENVM technology to design the storage, e.g., leveraging PCM [1], RRAM [3] or ENVM-Flash hybrid SSDs [40] to build the SCM system. For related research published before 2017, the readers can refer to [4].

2) Main Memory Level: ENVMs like PCM and STT-RAM have DRAM-comparable performance [4] and do not require refresh operations because of non-volatility, making them powerful alternatives for the main memory. However, ENVMs have limited write endurance, preventing their practical application. Write reduction and wear-leveling are two common solutions. The *write reduction* scheme usually improves the performance, energy efficiency, and durability simultaneously for ENVMs. Architectural optimization techniques include removing redundant bit-write, leveraging narrow rows, and exploiting multiple buffer rows for write coalescing [41], [42], [43]. Moreover, data compression is a feasible scheme to reduce the number of bits written to ENVM devices [44]. *Wear-leveling* is another optimization direction to improve ENVM write endurance. The main idea is to distribute writes evenly in the line [45] or between lines [46].

DRAM-ENVM Hybrid Main Memory: We can combine low-standby-power ENVM (capacity layer) with low-access-latency DRAM (performance layer) to construct hybrid (or heterogeneous) memory. Here, two strategies are commonly employed, i.e., the *tiering* approach where ENVM and DRAM reside at the same level of the memory hierarchy (e.g., [47], [48]), and the *caching* approach where DRAM acts as the cache of ENVM (e.g., [49], [50]). The primary problem for tiering is how to perform intelligent data management between DRAM and ENVM, including data placement and migration. The main idea is to allocate hot pages (i.e., frequently accessed data) to DRAM for high-performance considerations while distributing cold pages (i.e., infrequently accessed data) to ENVM to save energy and increase durability. For caching, it is critical to design efficient cache replacement algorithms.

3) Cache Level: Among ENVMs, STT-RAM has the best access performance and write endurance, and its cell size is much smaller than that of SRAM [51]. Thus, STT-RAM can potentially replace SRAM to construct a cache with the advantages of low leakage power and small on-chip area size. However, STT-RAM has higher write latency (∼3x) and higher write energy (∼7x) than SRAM [51], indicating it is unsuitable for the latency-critical L1 cache. Most research suggests leveraging STT-RAM as the LLC where the capacity is a more crucial metric. Considering that the cache block is replaced frequently, the data in a bit cell does not need to be retained for a long time. Some researchers suggest decreasing the cell size to relax its non-volatility (i.e., data retention time), finally reducing both the write latency and write energy [2]. Furthermore, Ahn et al. [51] find that a significant amount of data written into the STT-RAM LLC will not be re-referenced during the lifetime of corresponding cache blocks and can bypass such write operations for write reduction.

SRAM-ENVM Hybrid Cache: It is energy efficient to combine write-efficient SRAM with low-standby-power and high-density STT-RAM as a hybrid cache. The hybrid cache has two structures. The *tiering* scheme places SRAM and STT-RAM parallelly at the same cache level (e.g., [52], [53]), and the *caching* scheme leverages SRAM at the higher level with STT-RAM at the lower level (e.g., [54]). The main challenge of the hybrid cache is how to place different blocks between SRAM and ENVM partitions. The SRAM partition commonly has a smaller capacity, acting as a high-performance buffer. The ENVM partition usually has a larger capacity with low standby power. It is more beneficial to place the write-intensive blocks into the SRAM partition to improve performance, energy efficiency, and lifetime.

4) Register Level: Considering that STT-RAM has the anti-radiation feature, some researchers leverage it as the registers of processors working in the rad-hard environment (e.g., the space) for specific purposes [55]. Moreover, for GPUs whose register file (RF) is large and energy-hungry, low-leakage ENVMs like STT-RAM can potentially construct a light RF with lower standby power and smaller size. However, high write latency and high write power are two key challenges. Thus, many optimization techniques focus on reducing the redundant writes to the STT-RAM-based RF, including similarity-based data compression [56], write coalescing [57], adding the write pool/buffer [58], and early termination [59].

SRAM-ENVM Hybrid Register File: Some researchers propose to integrate SRAM with STT-RAM to construct a hybrid RF for GPUs [60], [61], and the main task is to make the hybrid design more efficient for the warp scheduler. For related research published before 2017, the readers can refer to [162]. Additionally, we should also pay attention to the read disturbance problem of STT-RAM-based RF in GPUs [62]. This problem may lead to high error rates, which the ECC scheme cannot effectively protect.

III. Near-Memory Processing

Near-memory processing (NMP) places processing units (PUs) next to memory units (MUs) to reduce the memory access overhead. Mainstream NMP prototypes have three schools: 2D, 2.5D and 3D structures. However, NMP also faces several *challenges*, such as high fabrication costs, intelligent workload offloading, memory coherence and programming burdens.

A. 2D NMP

1) DRAM-Based 2D NMP: As shown in Fig. 3, modern advanced integration technologies allow integrating processing units (PUs) at various places of DRAM devices, including the memory controller, the DIMM module (outside DRAM chips) and the component (inside DRAM chips). The **memory controller** is the bridge to connect processors and DRAMs, monitoring every data transfer. Some researchers [72], [73] have tried to integrate PUs near the memory controller to reduce the computing burden of the host processor.

DIMM consists of several DRAM chips, and **module-level** NMP schemes usually integrate PUs in the commodity DIMM

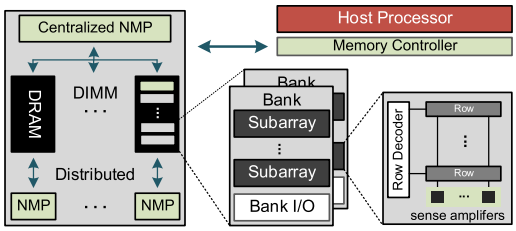


Fig. 3. Potential locations (green blocks) of a DRAM device to integrate NMP acceleration units.

to exploit chip-level parallelism and bandwidth. As shown in Fig. 3, according to the structures of PUs, they can be classified into two categories: centralized [63], [64], [65], [66], [67], [68], [69] and distributed [70], [71] architecture. In the *centralized* architecture, a powerful PU operates the data from multiple DRAM chips. In contrast, the *distributed* architecture integrates discrete PUs near each DRAM chip (e.g., data buffers).

A DRAM chip usually has several banks, and each bank equips sense amplifiers as the row buffer. Compared to module-level NMP, **component-level** schemes will provide higher parallelism and bandwidth. These benefits have encouraged the community to integrate PUs near banks [74], [75] or near sense amplifiers [77], [78]. Recently, a commercial near-bank NMP hardware called UPMEM [75] has been publicly available. UPMEM has full-stack software support and can be programmed in C language. Luna et al. [76] have given a benchmark test on this product.

2) SSD-Based NDP: For applications like Big Data workloads that cannot feed all required data (e.g., dozens of terabytes) into the DRAM cluster (e.g., hundreds of gigabytes), the frequent random access between DRAM and the lower-level storage has limited the overall performance. Many researchers propose to equip the commodity SSD with near-data processing (NDP) capability. The **first** idea is to equip the SSD controller with configurable processors [79], allowing users to define processing engines themselves. **Second**, integrating domain-specific accelerators in SSD is a more lightweight scheme. For example, Biscuit [80] adds pattern matcher hardware IPs in the SSD for simple data filtering. GenStore [81] is an in-storage computing system for genome sequence analysis, whose key idea is exploiting customized accelerators for sequence matching. AQUOMAN [82] integrates three pipelined database accelerators to offload the analytic-query workload into storage servers. The **third** solution utilizes existing wimpy cores (e.g., ARM series) embedded in the SSD for more practical in-storage computation [83], [84].

B. 2.5D NMP

Compared to the 2D integration manner, the 2.5D System-In-Package (SIP) technology [90] is much easier to integrate heterogeneous dies of different processes on one chip. As shown in Fig. 1 (2.5D NMP), different dies can communicate with each other through wires buried in the bottom silicon interposer. In recent years, high bandwidth memory (HBM) has been introduced to satisfy the demand for higher bandwidth. Usually, HBM devices are integrated with processor dies side by side upon a substrate using the 2.5D SIP technology, providing much higher bandwidth (10∼13x more I/O interconnects) at 2∼2.4x lower energy per bit transfer than conventional DRAM [90]. HBM has been applied in some high-throughput processors, including GPGPUs (e.g., NVIDIA Tesla P100/V100/A100), FPGAs (e.g., Intel Stratix 10 MX [85] and Xilinx Alveo U280 [86]), ASICs (e.g., Google TPU v2/v3/v4i [87], Huawei Ascent 910 [88] and NATSA [163]) and even many-core CPUs (e.g., Intel Xeon Max Series). Many researchers prefer HBM-equipped FPGAs for 2.5D NMP acceleration because of the advantages of reconfigurability, customizability and energy efficiency [85], [86].

Although HBM has provided much higher bandwidth than conventional DRAM, the demand for even higher bandwidth never ends. Samsung explains it is costly and impractical to further increase the memory bandwidth by adding more I/O pins [90]. They propose to integrate processing units near the banks on the bottom four DRAM dies of the HBM stack, and this has become an industry product, Samsung HBM-PIM [89], [90]. This product does not require any modification in modern commercial processors, and it can directly replace commodity HBM with full software stack support. Similarly, SK Hynix presents an HBM2E-based accelerator-in-memory (AiM) product called Newton [91] from the standpoint of a DRAM-maker. Newton takes into account both area and power constraints, only additionally integrating MAC units and buffers into commodity DRAMs. Strictly speaking, HBM-PIM and Newton belong to the hybrid NMP where 2D, 2.5D and 3D NMP technologies are all included. However, for simplicity, we classify all HBM-based NMP accelerators as 2.5D NMP architecture.

C. 3D NMP

1) SRAM-Based 3D NMP: For traditional 2D architecture, integrating distributed SRAM blocks near processing units, such as the Wire-Aware Architecture [164], is an intuitive idea to reduce the wire traversal overhead. Recently, the vertical integration of SRAM further reduces the wire length, contributing to space saving and speed enhancement simultaneously. SRAM-based 3D NMP schemes have two categories: array-level and device-level integrations. The **array-level** 3D integration scheme stacks a layer of regular SRAM array over the accelerating layer, which can further reduce the access latency with the benefit of shorter wire length. In the 3D structure, there are two ways to implement the inter-layer communication between the SRAM layer and other layers: *wired* through silicon vias (TSVs) [92] and *wireless* ThruChip Interface (TCI) [93]. It is announced that inductive coupling-based TCI technology has higher yields and lower costs than TSV technology [94]. 3D SRAM array provides high memory bandwidth and large capacity, reducing off-chip data movement overheads significantly.

The **device-level** 3D integration scheme utilizes monolithic 3D (M3D) technology to fabricate multi-layer SRAM bitcells and improve the SRAM density significantly. More aggressively, some works integrate logic above the SRAM cell to design a PIM device. Hsueh et al. [95] present the first M3D-based SRAM cell prototype consisting of 9 transistors, which can compute

simple logic operations within a single memory cycle. Srinivasa et al. [96] integrate additional transistors over the SRAM layer to improve the robustness of the cell and equip the SRAM cell with computing capabilities, achieving much higher energy efficiency compared to processing outside the memory.

2) DRAM-Based 3D NMP: Recent process technology advancements like 3D die-stacking have enabled the vertical integration of logic dies with DRAM dies in one stack (e.g., HMC [165]). Each layer communicates with others using TSVs, significantly reducing the memory access latency and energy compared to off-chip data transfer. These benefits have motivated many works to integrate various processing cores in the logic die of the 3D stack to build up the DRAM-based NMP system. These cores include CPU cores [97], [98], SIMD cores [99], configurable cores [100], and customized ASIC cores [101]. Meanwhile, some research also integrates heterogeneous cores in the logic die, such as CPU + SIMD cores [102] and CPU + ASIC cores [103].

However, it is challenging to integrate powerful processors in the 3D stack because of thermal constraints. Thus, recent HMC-based NMP accelerators prefer integrating simple cores into the logic die, serving as the coprocessors linked to the host processor. The main idea is to use the 3D NMP accelerator to process memory-intensive tasks like MapReduce, graph processing, and DNN workloads [100]. At the same time, the powerful host processor takes charge of processing compute-intensive tasks. This kind of host-slave architecture also requires PIM-aware programming supports, such as specialized PIM-enabled instructions [104], customized PIM programming interface [97], and programmer-transparent intelligent computation offloading/scheduling schemes [99]. Furthermore, considering that the host processor and the NMP accelerator share the same DRAM stack, the coherence problem has to be solved efficiently [98].

3) ENVM-Based 3D NMP: ENVM provides higher density and lower static power than SRAM. Dong et al. [105] propose a 3D-stacked MRAM (magnetic RAM) cache model and discuss circuit-level issues. However, MRAM has long write latency and high write power, damaging system-level performance and energy efficiency. Sun et al. [106] design a 3D-stacked MRAM-SRAM hybrid L2 cache and add the write buffer to reduce negative effects of write operations. Mishra et al. [107] propose a cache management policy that will reallocate write operations to other idle MRAM cache banks if original banks are currently serving long-latency write requests.

We can stack ENVM devices atop processing cores to construct an NMP system. Among the ENVMs, PCM is the most mature one whose memory density and access latency are competitive with DRAM. Moreover, PCM has zero standby power and better technology scaling. Hosseini et al. [108] compare a conventional CPU system with PCM-based 3D NMP and DRAM-based 3D NMP tested on sufficient data-intensive applications, finding the NMP architecture will perform better when the application has poorer data locality. Moreover, the NMP architecture has lower energy consumption than the conventional CPU system, and the PCM-based 3D NMP has higher energy efficiency than the DRAM-based 3D NMP in most cases.

IV. In-Memory Processing

In-memory processing (IMP), as a different computing paradigm beyond the von Neumann model, performs in-place processing in the memory device to eliminate the data movement fundamentally. We have found instances of IMP based on different memory technologies, such as SRAM, DRAM, and NVM. Many IMP researchers focus on dealing with the circuit non-ideality and programmability problems.

A. In-SRAM Processing

Extending the peripheral circuit with logic functions is a general idea to equip SRAM with in-place computation capability. A common way is to integrate the computation logic at the end of the bit line (BL), and massively parallel BLs can perform wide-vector bit-wise operations. In-SRAM processing (or computable SRAM) has two research directions: in-SRAM accelerator and in-cache computing. The former aims at domain-specific applications with low programmability. The latter is for general-purpose applications with high programmability.

1) In-SRAM Accelerator: The first kind of computable SRAM often architects the SRAM array as a compute engine in the low-power ASIC accelerator. It sacrifices some of the programmability for higher energy efficiency. In-SRAM accelerator leverages the analog computing property for higher efficiency, but the computation accuracy is sensitive to noise and process voltage temperature (PVT) variations. Thus, low-precision computation is more suitable for this kind of architecture, and it reduces energy and performance overheads due to less computation and memory requirements.

Dot Product: The key computation component of a DNN model is the dot product, namely, a kind of multiply-accumulate (MAC) operation in nature. SRAM-based analog dot product computation has two ways: current-based and charge-based approaches. The *current-based* approach stores a vector of weights into the SRAM cells in the same bit line, with the inputs converted into the driven voltages of word lines. Then, the final results are represented by the current accumulated in the bit line of the SRAM array [109], [110]. On the other hand, the *charge-based* approach uses capacitors to augment the SRAM cells, with the input values transformed as the charges in the capacitors. Then, the dot product operations are performed through charge sharing in the bit line [111], [112].

Circuit Non-Idealities: Many in-SRAM ML inference accelerators use a standard 6 T SRAM array as the compute units [110], [111]. The parameters are trained offline and then mapped to the accelerators without considering the chip-specific variations. However, the analog computing nature makes the computation accuracy highly susceptible to the circuit non-idealities and various random variations [113]. Thus, it is necessary to solve the robustness problem. Related approaches include (1) extending the ML accelerator with on-chip training capabilities [114] which can adapt to various hardware variations and achieve higher accuracy; (2) using 8 T/10 T SRAM to construct the array [115], [116].

2) In-Cache Computing: The second kind of computable SRAM often equips the cache in a general-purpose processor

(e.g., CPU or GPU) with processing ability, and it is more programmer-friendly.

Simple Logical Operations: Jeloka et al. [117] propose a configurable SRAM architecture that can be configured with content addressability and logical function, enabling related computational operations off-loaded into the cache. Aga et al. [118] present the Compute Cache architecture to support simple vector processing (e.g., copy, compare, and logical operations) through bit-line computing. However, in-place bit-line computing faces the constraint of operand locality, namely that data operands have to be stored in the same set of bit lines. To solve this issue, they also propose a near-place computing scheme for Compute Cache, and the operation is performed in the logic unit placed close to the cache controller.

Complex Arithmetic Computation: Supporting simple logical operations is not enough to equip the cache with more general-purpose computing capabilities. Eckert et al. [119] design the Neural Cache architecture utilizing simple logical operations to perform regular arithmetic computations like addition, multiplication, and reduction, providing more functions than Compute Cache. Neural Cache supports bit-serial computing, making it good at vector operations with many elements where the element number is much larger than the bit precision. However, Neural Cache focuses more on low-precision (8-bit) computation, limiting its application scenarios. Then, Wang et al. [120] propose a general-purpose hybrid in-/near-memory compute SRAM (CRAM) architecture. CRAM performs simple logical operations using in-memory bit-serial computing and processes the complex high-precision (32-bit or 64-bit) integer or floating-point operations in a near-memory Cortex-M0 CPU. Recryptor [166] shares similar ideas with CRAM but aims at cryptographic algorithms.

Programmability: Fujiki et al. [121] propose the bit-serial computing enabled Duality Cache architecture and design a holistic system stack to provide higher programmability, including the ISA and compiler support. This software stack accepts existing CUDA programs and makes the compute-capable caches more available in general-purpose computing. However, the fixed bit-precision architecture may lead to hardware underutilization in applications requiring the computation of diverse bit precisions, such as ML inference. To satisfy the requirement of more flexible bit precisions, Lee et al. [122] introduce a precision-reconfigurable bit parallel in-SRAM computing architecture. It adopts short word lines with bit line (BL) boosting circuits to enhance the BL computing performance and avoid read disturbance.

B. In-DRAM Processing

DRAM works as the main memory in modern computing systems, and it has a large internal bandwidth due to the wide row (several KBs). However, DRAM inherently consumes much dynamic energy, and there is little free space to integrate complex computation logic in the peripherals of the DRAM array. Thus, it is more suitable to equip the DRAM with simple processing abilities, such as in-DRAM data movement and bitwise operations.

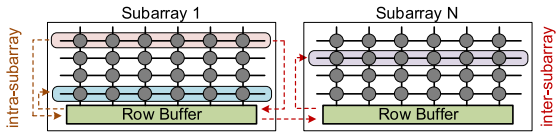


Fig. 4. Two kinds of in-DRAM data movement tasks with the help of row buffer: intra-subarray and inter-subarray movement.

1) In-DRAM Data Movement: Many system-level programs will trigger bulk data operations that do not require any computation but just data movement, such as copy and initialization. However, in the conventional system, the bulk data will be first transferred from the source position of the memory to the processor and then transferred back to the target position in the memory. This kind of off-chip data movement consumes much time and energy. An intuitive solution is to directly move the data between different rows with the help of row buffers in the DRAM. Fig. 4 shows the process of moving a row of data to another row within the subarray and between two subarrays. Seshadri et al. [123] present the RowClone mechanism to support bulk copy and initialization operations in the DRAM without transferring the data through the memory channel. However, RowClone is not good at inter-subarray data movement where the rows do not share a row buffer. Then, Chang et al. [124] present a new DRAM substrate called LISA, which enables fast data movement between adjacent subarrays by adding low-cost connections. Recently, RowClone has also inspired some other in-DRAM data movement techniques, such as CROW [125], HrDRAM [126] and FIGARO [127].

2) Simple Bulk Bitwise Operations: Bitwise operations between large vectors are commonly seen in applications like databases and web searching. Many researchers propose in-DRAM computing mechanisms to reduce the data transferring overheads of bulk bitwise operations. The key idea is to exploit the bit line (BL) analog computing property of DRAM. DRAM stores data as the charge in the capacitor and can perform bitwise operations through charge sharing between different cells in the same BL. Finally, the result is represented as the charge in one capacitor and can be detected by the sense amplifier (SA).

Seshadri et al. [128] design a simple but efficient mechanism to perform bulk bitwise operations (including AND and OR) between two rows in DRAM through the charge sharing mechanism. In detail, three cells are connected simultaneously to a BL, and controlling the value of one will force the other two cells to perform AND/OR operations. The main change to commodity DRAM is extending the row decoding logic with the function of simultaneously activating three rows. More practically, to realize simultaneous multi-row activations in a commercial off-the-shelf DRAM without any hardware change to the DRAM design, Gao et al. [129] propose to violate the nominal timing specification and activate multiple rows in rapid succession. This method is easy to be implemented at a low cost. However, AND/OR operations alone are not logically complete. To cover more bitwise functions than AND/OR, Seshadri et al. [130] propose the Ambit mechanism leveraging the inverters inside SA to perform bitwise NOT operations. Thus, Ambit supports a broader range of applications.

Until now, Ambit has inspired many other DRAM-based IMP designs. For example, in the motion planning task of intelligent agents like robots, collision detection [167] is the performance bottleneck that exhibits obvious memory-intensive patterns. Based on the Ambit approach, Yang et al. [131] propose an IMP accelerator called Dadu-CD for collision detection with the hardware/software co-design, eliminating most off-chip memory accesses by moving the computation into DRAM. Xin et al. [132] point out that Ambit has five aspects of problems, and they propose new techniques to deal with them.

3) Complex Arithmetic Operations: It is essential to cover the arithmetic computation functions, such as addition and multiplication, to equip the in-DRAM accelerators with more general-purpose computing capabilities. Li et al. [133] present a DRAM-based reconfigurable in-situ accelerator (DRISA) that performs basic bitwise logical operations through bit line computation. Bitwise AND/OR operations can be performed by the DRAM cell itself through charge sharing, while other bitwise operations (like NOR) require augmenting the SA with logic gates. DRISA can be re-configured to compute various arithmetic functions (e.g., addition and multiplication) via serially running the functionally complete boolean logic operations with the help of hierarchical internal data movement circuits. Deng et al. [134] mention that DRISA is a heavyweight design that results in large area overheads, and they propose a lightweight DRAM-based IMP design called DrAcc based on the 3D-stacked WideIO2 architecture. DrAcc exploits the in-DRAM bit operation to implement ternary weight neural networks with high inference accuracy and energy efficiency while adding less than 2% area overhead.

Since logic units and DRAM cells are usually incompatible in the technology process, implementing complex arithmetic computation in DRAM will increase integration complexity and power overheads. Thus, **special in-DRAM computing schemes** are required to replace complex arithmetic operations. The *stochastic computing* technology [135] can convert integer multiplications into simple bitwise AND operations, which can be implemented in DRAM without modifying the original memory architecture. This computing paradigm naturally matches error-tolerant applications like deep learning. SIMDRAM [136] leverages *bit-serial* operations to implement complex computations and provides an end-to-end programming framework to improve flexibility. LAcc [137] and pLUTo [138] utilize the *lookup table* (LUT) technique to hold precomputed values in DRAM. Then, complex operations are transferred to memory read operations (i.e, LUT queries). Kim et al. [139] emphasize that many previous SRAM/DRAM-based IMP designs have paid much additional overhead to support multiplication. Then, they propose the NAND-Net architecture with simple *bitwise NAND* operations to realize multiplications. NAND can be further converted to AND and NOR operations without modifying original memory bit cells.

C. In-NVM Processing

NVM devices can be utilized to process the stored data based on the analog computation mechanism. The imprecise property of analog computation makes in-NVM processing more suitable for some simple bulk bitwise operations and error-tolerant applications like neural networks (NN).

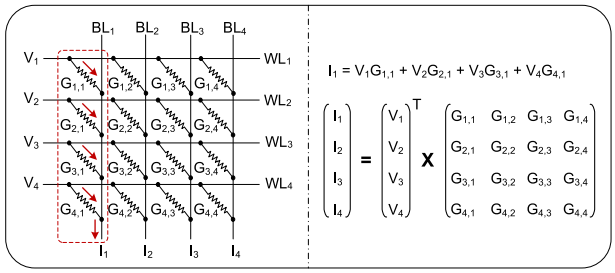


Fig. 5. Using the analog computing capability of memristor crossbar to implement dot production and vector matrix multiplication operations.

1) In-Flash Processing: Flash-based memory is inherently analog and has mature fabrication technology, suitable for practical IMP implementation based on analog computation. When floating gate memory devices work in the subthreshold mode, they can be configured to perform multiplication under the control of the input voltages directly applied to the gates, and the output currents represent the final results. Bayat et al. [140] redesign a commercial **NOR Flash** memory array to perform vector-by-matrix multiplication (VMM) with high performance and low power, although exhibiting low programming speed and relatively low precision. Then, to reduce the temperature drift and improve the robustness, Guo et al. [141] implement and characterize a different version of the VMM based on the NOR Flash memory array, minimizing the output signal drift. Besides NOR Flash, the intrinsic computation capability of **NAND Flash** array is leveraged to enable parallel bitwise operations in SSD [142]. Park et al. [143] propose Flash-Cosmos to solve the bit-level parallelism underutilization and unreliability problems of the NAND Flash-based IMP.

2) In-ENVM Processing: Resistive ENVMs store the data value as resistance and leverage current sensing for data access. These properties make them inherently suitable for analog computing. Resistive ENVMs belong to the category of the memristor. In recent years, memristive arrays have been successfully architected for various applications, especially neural networks. Research about neural networks (NNs) has two schools: computer science-based artificial neural network (ANN) and neuroscience-based spiking neural network (SNN). ANN and SNN have fundamental differences in the formulations, coding schemes, and learning rules. SNN encodes data into the timing of spikes where spatial and temporal information are both involved, and a sequence of spikes will transmit more information than the binary encoding scheme of ANN. ANN is commonly trained through the global Back-Propagation (BP) algorithm based on the gradient descent, while SNN is trained via the local Spike-Timing Dependent Plasticity (STDP) learning rule.

Memristive ANN accelerators: ANNs involve massive multiply-accumulate (dot-product) operations which can be processed in the memristor crossbar. As shown in Fig. 5, dot-product operations are performed by leveraging Ohm's Law and Kirchoff's Law to accumulate the bit line current as the sum of

products. Shafiee et al. [144] present a pipelined architecture called ISAAC with the crossbar memory as the dot-product engine and different crossbars dedicated for different ANN layers. Similarly, Li et al. [145] leverage large memristor crossbars for image processing, whose main computation is matrix-vector multiplication (MVM). In the meantime, Chi et al. [146] propose a configurable architecture called PRIME with the RRAM crossbar arrays either as ANN accelerators or as the normal memory, and they also design a software-hardware interface to develop various ANNs (e.g., CNN and RNN) more easily.

Memristive SNN Accelerators: Neurons and synapses are two major components in the SNN model, and synapses serve to connect neurons. Neurons communicate with each other by transmitting voltage spikes through the synaptic connections, and the synaptic weights can modulate the voltage spikes. Moreover, a neuron will change its membrane potential after receiving voltage spikes from other neurons. If the membrane potential exceeds a threshold, this neuron will fire and send voltage spikes to its followers. From the perspective of computer science, neurons are computing units that integrate the inputs and generate voltage spikes. Synapses are memory units whose weights represent the strength of connections between neurons.

CMOS-based SNN hardware uses SRAM or eDRAM as artificial synapses, but these conventional volatile memory devices consume much energy. Memristors have regulable resistances making them suitable to emulate the functionalities of synapses. Thus, many researchers leverage memristors as synapses to obtain higher energy efficiency [147], [148]. Memristive synapses are commonly organized in the form of crossbar arrays to provide high density. However, CMOS-based neurons are incompatible with the memristive synapses, requiring extra peripheral circuits. Therefore, some researchers explore memristors as artificial neurons, with the membrane potential represented by the conductance of memristors. Tuma et al. [149] present a PCM-based artificial neuron model which realizes the temporal integration function of the neuron by exploiting the reversible phase transitions of phase-change materials. Lashkare et al. [150] present an RRAM-based neuron model where increasing the analog conductance enables the integration function, and the conductance will decrease when exceeding the threshold.

Non-Idealities: Although the memristive crossbar architecture naturally matches NN applications, exhibiting very high energy efficiency, the approximate nature of analog computation has unfortunately arisen as a key challenge to the system-level performance and accuracy. First, device-level and circuit-level non-idealities will make the conductance deviate from the desired network weight. Then, numerous accumulated errors may lead to significant accuracy reduction, especially for large-scale memristive circuits. These non-idealities include non-linear current-voltage characteristics, device-to-device process variations, cycle-to-cycle variations, writing endurance, data retention time, parasitic resistances, and sneak path. To learn more about the impacts of these non-idealities, Sun et al. [151] take a comprehensive investigation and provide several suggestions. Chakraborty et al. [152] utilize NNs to accurately capture the non-idealities of the memristive crossbars, outperforming a typical analytical model, and the accuracy of this method is comparable to that of HSPICE.

Considering that non-idealities of memristive NN accelerators negatively affect the system-level performance, it is essential to mitigate such imperfections. Zhang et al. [153] present a device-circuit-algorithm joint *analysis framework* for memristive processors and propose an approach to evaluate these non-idealities. On the other hand, equipping the memristive processor with *on-chip training* capability will make the circuits adapt to imperfections. For instance, JAIN et al. [154] propose a hardware-software compensation model including a fast one-time re-training method and a low-overhead circuit to recover accuracy loss. However, it is challenging and expensive to train a complicated NN entirely on the memristor hardware. Then, Yao et al. [155] present a *hybrid in-situ training* method for memristor-based CNN accelerators. In detail, the weights of convolutional layers are trained off-chip using software on the general-purpose platform and kept unchanged when deployed on-chip. Only the weights of fully connected (FC) layers will be updated on-chip. This method balances improving the accuracy (by on-chip re-training) and effectively reducing the re-training overheads (via maintaining the existing high-performing parameters trained off-chip). Moreover, Imani et al. [168] propose the FloatPIM architecture to support floating-point-based accurate CNN training.

Other Potential Applications: Besides NN tasks, many researchers have applied ENVM-based IMP accelerators into broader domains, such as DNA short read alignment in genomic analyses [156], graph processing [157], clustering algorithms [158] and database (e.g., compression, encryption and format conversion) [159]. These domain-specific accelerators have achieved great success by virtue of massive parallelism and high energy efficiency. However, many works rely heavily on the manual mapping of specialized kernels to the memristive array, limiting their flexibility. To combat this problem and to extend the ENVM-based IMP for more general-purpose computation, Fujiki et al. [160] propose a programmable in-memory processor architecture and a compact instruction set. They also design the software stack including a data-parallel programming framework and a compilation framework, making it more programmer-friendly. Nevertheless, Zha et al. [161] elaborate that most existing ENVM-based IMP works will be bottlenecked by the power-hungry and area-inefficient ADC/DAC. Then, they propose an RRAM-based IMP accelerator which executes the Associative Processing (AP) model, minimizing the modifications on the memory peripheral circuits. AP essentially utilizes a lookup table to search results for arbitrary-precision computation in the memory.

V. Programming Model

Both NMP and IMP devices can be viewed as processing-in-memory (PIM) accelerators which can be roughly classified into two categories: *fixed-function accelerators* controlled by a pre-defined finite-state machine (FSM), and *programmable accelerators* which enable simple customized operations [104]

to be interfaced as simple instruction set architecture (ISA) extensions. For the former one, a common flow includes allocating data at the designated memory location and invoking the accelerator by setting up the control state register (CSR). Then, the accelerator will automatically execute following a fixed data flow. Such a starting mode is not trivial but will not significantly affect current programming models. Thus, here we focus more on the programming model of the programmable PIM accelerators.

The first question is how to apply current programming models to PIM accelerators without modifications. The answers include: 1) Making the PIM accelerators support main-stream programming languages (e.g., C [161]) or models (e.g., CUDA [99], TensorFlow [90], [160] and PyTorch [90]) will not increase the learning cost. Practical ways include customizing a *compilation* framework [160], [161] and providing the optimized high-level *library* [90]. (2) Sometimes, a *runtime* is required to intelligently offload the IMP kernels/instructions into the PIM accelerators [99]. Such a toolchain can make the PIM device transparent to the programmer. **The second question** is about the restriction faced when extending current programming models to PIM models. The limitations include workload offloading, memory coherence and security considerations about potentially unforeseen data leakage. (1) Offloading which part of the workload to the memory accelerator is an open topic. The instruction-level [104] and kernel-level [99] offloading schemes have both been discussed in public literature. Not every kind of workload is suitable to execute on the memory side, and memory-intensive kernels are more welcome to be offloaded because more data movement will be eliminated. 2) The cache-equipped host processor and memory-side accelerators (without cache) usually share the main memory, and that will cause the cache coherence problem [98] if there is no hardware to perform coherence actions between them. As alternatives, the programmer can leverage software methods like cache flush operations to keep memory coherence, but it will cause additional programming burdens. 3) Since PIM accelerators usually reside outside the host processor core, there exists a data leakage risk during their interaction. Thus, the runtime system should provide security guarantees when offloading kernels to the memory side. An intuitional idea is to perform isolation between different processes through techniques like partitioning and virtualization [169], preventing malicious processes from stealing the data of others.

VI. Future Research Trend

Analog computing is reviving and replacing digital computing in applications with low-precision requirements: This kind of computing paradigm achieves significant energy efficiency. We can find many works that use analog computing for simple logical and arithmetic operations. For example, binarized neural networks involve many bitwise operations, making analog computing suitable in this area. However, analog computing has a circuit non-ideality problem that may reduce the accuracy of the calculation. This problem has attracted much attention in this field.

Near/In-Memory Processing (NMP/IMP) technology is increasingly popular, making the distance between processing units and memory units shorter and shorter: NMP technology evolves from 2D to 2.5D/3D structures. The 3D NMP architecture provides high internal bandwidth and low access latency. Moreover, the 3D structure has severe thermal problems, and a promising research direction is to mitigate this problem. IMP technology integrates logic units in the memory array or utilizes the analog computing property of the memory array to equip the memory with processing abilities. The IMP architecture eliminates a large amount of data movement and can potentially mitigate the memory wall problem. However, current IMP accelerators lack a friendly programming model, and it is an urgent problem.

Artificial Intelligence (AI) and computer architecture communities are interplaying deeply: Nowadays, we can find research about designing hardware for machine learning (ML) applications and applying ML algorithms for hardware. ML applications involve a large amount of computation, and ML accelerators can improve system-level energy efficiency. On the other side, ML algorithms usually have a good capability of knowledge representation, namely that they are suitable to perform a powerful design space exploration for the computer architecture. ML algorithms like the perceptron, linear regression, and neural networks are appropriate for the prediction tasks. We can leverage these ML models for cache prefetching and branch predicting.

Security, privacy, and reliability should be considered when designing energy-efficient architectures: Both the processor and memory may encounter some hardware or software faults, making them vulnerable to being attacked through physical or software methods. For instance, Rowhammer is a famous hardware DRAM vulnerability where frequently accessing a row will cause bit flips in neighbouring rows. It sources from circuit-level disturbance errors and will worsen as the DRAM density scales. More unfortunately, Rowhammer can be leveraged to attack system security. The reliability and security problems deserve our attention.

Lastly, since current mature process technologies (e.g., CMOS technology) are gradually approaching the physical limit, many emerging devices and materials are required as candidate technologies: For example, some emerging resistive non-volatile memory technologies (e.g., 3D XPoint) have been exploited in the conventional memory hierarchy. To some extent, advances in computer technology are usually accompanied by breakthroughs in the physical layer.

VII. Conclusion

This paper focuses on memory-centric energy efficient techniques. According to the distance between processing and memory, we classify the computer architecture into three categories: off-memory processing (OMP), near-memory processing (NMP), and in-memory processing (IMP). The OMP architecture still belongs to the traditional von Neumann architecture, and the main optimization ideas lie in optimizing the conventional memory technology and exploiting emerging

NVM technology. The NMP architecture integrates processing and memory units more closely to reduce the distance of data movement, providing high bandwidth and low latency. We introduce three NMP implementations in detail: 2D, 2.5D and 3D NMP. The IMP technology performs in-place processing in the memory device (e.g., SRAM, DRAM, and NVM) and eliminates the data movement fundamentally. Lastly, we discuss the programming model of NMP/IMP and several research trends in energy-efficient computer architecture. We hope this paper can provide a general understanding for the newcomers and some novel inspirations for the researchers in this community.

REFERENCES

[1] T. Kim and S. Lee, "Evolution of phase-change memory for the storage-class memory and beyond," *IEEE Trans. Electron Devices*, vol. 67, no. 4, pp. 1394–1406, Apr. 2020.

[2] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 50–61.

[3] Q. Luo et al., "8-layers 3D vertical RRAM with excellent scalability towards storage class memory applications," in *Proc. Int. Electron Devices Meeting*, 2018, pp. 2.7.1–2.7.4.

[4] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1537–1550, May 2016.

[5] J. Izraelevitz et al., "Basic performance measurements of the Intel Optane DC persistent memory module," 2019. [Online]. Available: http://arxiv.org/abs/1903.05714

[6] A. Foong and F. Hady, "Storage as fast as rest of the system," in *Proc. IEEE 8th Int. Memory Workshop*, 2016, pp. 1–4.

[7] J. Yang et al., "An empirical guide to the behavior and use of scalable persistent memory," in *Proc. USENIX Conf. File Storage Technol.*, 2020, pp. 169–182.

[8] O. Mutlu et al., "A modern primer on processing in memory," 2020, *arXiv: 2012.03112*.

[9] T. P. Xiao et al., "Analog architectures for neural network acceleration based on non-volatile memory," *Appl. Phys. Rev.*, vol. 7, no. 3, 2020, Art. no. 031301.

[10] 2023. [Online]. Available: https://github.com/ChangwuZhang

[11] J. Liu et al., "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. Int. Symp. Comput. Archit.*, 2012, pp. 1–12.

[12] M. K. Qureshi, D. H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2015, pp. 427–437.

[13] K. K. W. Chang et al., "Improving DRAM performance by parallelizing refreshes with accesses," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2014, pp. 356–367.

[14] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Improving energy efficiency of DRAM by exploiting half page row access," in *Proc. Annu. Int. Symp. Microarchit.*, 2016, pp. 1–12.

[15] S. Kim, W. Kwak, C. Kim, D. Baek, and J. Huh, "Charge-aware DRAM refresh reduction with value transformation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 663–676.

[16] N. Chatterjee et al., "Architecting an energy-efficient DRAM system for GPUs," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2017, pp. 73–84.

[17] M. O'Connor et al., "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems," in *Proc. IEEE/ACM 50th Annu. Int. Symp. Microarchit.*, 2017, pp. 41–54.

[18] D. Lee et al., "Simultaneous multi-layer access: Improving 3D-stacked memory bandwidth at low cost," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 1–29, 2016.

[19] T. Kgil et al., "PicoServer: Using 3D stacking technology to enable a compact energy efficient chip multiprocessor," in *Proc. Int. Conf. Architectural Support Prog.. Lang. Operating Syst.*, 2006, pp. 117–128.

[20] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 453–464.

[21] W. Zang and A. Gordon-Ross, "A survey on cache tuning from a power/energy perspective," *ACM Comput. Surveys*, vol. 45, no. 3, pp. 1–49, 2013.

[22] K. Puttaswamy and G. H. Loh, "3D-integrated SRAM components for high-performance microprocessors," *IEEE Trans. Comput.*, vol. 58, no. 10, pp. 1369–1381, Oct. 2009.

[23] G. H. Loh, "Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy," in *Proc. Annu. Int. Symp. Microarchit.*, 2009, pp. 201–212.

[24] R. Bera et al., "Hermes: Accelerating long-latency load requests via perceptron-based off-chip load prediction," in *Proc. IEEE/ACM 55th Int. Symp. Microarchit.*, 2022, pp. 1–18.

[25] M. H. Lipasti et al., "Value locality and load value prediction," in *Proc. 7th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 1996, pp. 138–147.

[26] J. Yang and R. Gupta, "Load redundancy removal through instruction reuse," in *Proc. Int. Conf. Parallel Process.*, 2000, pp. 61–68.

[27] S. Mittal, "A survey of recent prefetching techniques for processor caches," *ACM Comput. Surveys*, vol. 49, no. 2, pp. 1–35, 2016.

[28] S. Mittal and J. S. Vetter, "A survey of architectural approaches for data compression in cache and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1524–1536, May 2016.

[29] D. Li et al., "Priority-based cache allocation in throughput processors," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 89–100.

[30] X. Xie et al., "CRAT: Enabling coordinated register allocation and thread-level parallelism optimization for GPUs," *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 890–897, Jun. 2018.

[31] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multicore memory systems," *ACM Trans. Comput. Syst.*, vol. 30, no. 2, pp. 335–346, 2012.

[32] X. Xie, Y. Liang, Y. Wang, G. Sun, and T. Wang, "Coordinated static and dynamic cache bypassing for GPUs," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 76–88.

[33] H. Dai, C. Li, H. Zhou, S. Gupta, C. Kartsaklis, and M. Mantor, "A model-driven approach to warp/thread-block level GPU cache bypassing," in *Proc. Des. Automat. Conf.*, 2016, pp. 1–6.

[34] G. Koo et al., "Access pattern-aware cache management for improving data utilization in GPU," in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 307–319.

[35] S. Gupta, H. Gao, and H. Zhou, "Adaptive cache bypassing for inclusive last level caches," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 1243–1253.

[36] T. L. Johnson, D. A. Connors, M. C. Merten, and W.-M. W. Hwu, "Run-time cache bypassing," *IEEE Trans. Comput.*, vol. 48, no. 12, pp. 1338–1354, Dec. 1999.

[37] M. Kharbutli et al., "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Apr. 2008.

[38] S. Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–39, 2017.

[39] G. Gracioli et al., "A survey on cache management mechanisms for real-time embedded systems," *ACM Comput. Surveys*, vol. 48, no. 2, pp. 1–36, 2015.

[40] C. Matsui, C. Sun, and K. Takeuchi, "Design of hybrid SSDs with storage class memory and NAND flash memory," in *Proc. IEEE*, vol. 105, no. 9, pp. 1812–1821, Sep. 2017.

[41] B. C. Lee, E. Ipek, O. MutIu, and D. Burger, "Phase change memory architecture and the quest for scalability," *Commun. ACM*, vol. 53, no. 7, pp. 99–106, 2010.

[42] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Int. Symp. Comput. Archit.*, 2009, pp. 2–13.

[43] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 256–267.

[44] Z. Chen, Y. Hua, P. Zuo, Y. Sun, and Y. Guo, "Reducing bit writes in non-volatile main memory by similarity-aware compression," in *Proc. Des. Automat. Conf.*, 2020, pp. 1–6.

[45] M. Asadinia, M. Jalili, and H. Sarbazi-Azad, "BLESS: A simple and efficient scheme for prolonging PCM lifetime," in *Proc. Des. Automat. Conf.*, 2016, pp. 1–6.

[46] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. IEEE/ACM 42nd Annu. Int. Symp. Microarchit.*, 2009, pp. 14–23.

[47] L. Liu, S. Yang, L. Peng, and X. Li, "Hierarchical hybrid memory management in OS for tiered memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2223–2236, Oct. 2019.

[48] C. Wang et al., "Panthera: Holistic memory management for Big Data processing over hybrid memories," in *Proc. ACM SIGPLAN Conf. Prog.. Lang. Des. Implementation*, 2019, pp. 347–362.

[49] X. Liu, D. Roberts, R. Ausavarungnirun, O. Mutlu, and J. Zhao, "Binary star: Coordinated reliability in heterogeneous memory systems for high performance and scalability," in *Proc. IEEE/ACM 52nd Annu. Int. Symp. Microarchit.*, 2019, pp. 807–820.

[50] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 24–33.

[51] J. Ahn, S. Yoo, and K. Choi, "DASCA: Dead write prediction assisted STT-RAM cache architecture," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2014, pp. 25–36.

[52] J. Ahn, S. Yoo, and K. Choi, "Prediction hybrid cache: An energy-efficient STT-RAM cache architecture," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 940–951, Mar. 2016.

[53] J. Zhang, M. Jung, and M. Kandemir, "FUSE: Fusing STT-MRAM into GPUs to alleviate off-chip memory access overheads," in *Proc. 25th IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 426–439.

[54] K. Korgaonkar et al., "Density tradeoffs of non-volatile memory as a replacement for SRAM based last level cache," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 315–327.

[55] Z. Gong et al., "Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment," *Sustain. Comput. Inform. Syst.*, vol. 22, pp. 206–218, 2019.

[56] H. Zhang, X. Chen, N. Xiao, and F. Liu, "Architecting energy-efficient STT-RAM based register file on GPGPUs via delta compression," in *Proc. 53rd Annu. Des. Automat. Conf.*, 2016, pp. 1–6.

[57] N. Goswami, B. Cao, and T. Li, "Power-performance co-optimization of throughput core architecture using resistive memory," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 342–353.

[58] J. Wang and Y. Xie, "A write-aware STTRAM-based register file architecture for GPGPU," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, pp. 1–12, 2015.

[59] X. Liu, M. Mao, X. Bi, H. Li, and Y. Chen, "Exploring applications of STT-RAM in GPU architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 238–249, Jan. 2021.

[60] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "Towards warp-scheduler friendly STT-RAM/SRAM hybrid GPGPU register file design," in *Proc. IEEE/ACM Int. Conf. Comput. -Aided Des.*, 2017, pp. 736–742.

[61] Q. Deng, Y. Zhang, Z. Zhao, S. Zhang, M. Zhang, and J. Yang, "FRF: Toward warp-scheduler friendly STT-RAM/SRAM fine-grained hybrid GPGPU register file design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2396–2409, Oct. 2020.

[62] H. Zhang et al., "Shielding STT-RAM based register files on GPUs against read disturbance," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 2, pp. 1–17, 2016.

[63] J. Cong et al., "AIM: Accelerating computational genomics through scalable and noninvasive accelerator-interposed memory," in *Proc. Int. Symp. Memory Syst.*, 2017, pp. 3–14.

[64] M. Alian et al., "Application-transparent near-memory processing architecture with memory channel network," in *Proc. IEEE/ACM 51st Annu. Int. Symp. Microarchit.*, 2018, pp. 802–814.

[65] L. Ke et al., "RecNMP: Accelerating personalized recommendation with near-memory processing," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, vol. 2020-May, pp. 790–803, 2020.

[66] W. Sun, Z. Li, S. Yin, S. Wei, and L. Liu, "ABC-DIMM: Alleviating the bottleneck of communication in DIMM-based near-memory processing with inter-DIMM broadcast," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit*, 2021, pp. 237–250.

[67] S. Feng et al., "MeNDA: A near-memory multi-way merge solution for sparse transposition and dataflows," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 245–258.

[68] G. Dai et al., "DIMMining: Pruning-efficient and parallel graph mining on near-memory-computing," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 130–145.

[69] L. Ke et al., "Near-memory processing in action: Accelerating personalized recommendation with AxDIMM," *IEEE Micro*, vol. 42, no. 1, pp. 116–127, Jan./Feb. 2022.

[70] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *Proc. IEEE/ACM 49th Annu. Int. Symp. Microarchit.*, 2016, pp. 1–13.

[71] W. Huangfu et al., "MEDAL: Scalable DIMM based near data processing accelerator for DNA seeding algorithm," in *Proc. Annu. Int. Symp. Microarchit.*, 2019, pp. 587–599.

[72] V. Seshadri et al., "Gather-scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses," in *Proc. 48th Int. Symp. Microarchit.*, 2015, pp. 267–280.

[73] G. Singh et al., "NERO: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *Proc. IEEE 30th Int. Conf. Field-Programmable Log. Appl.*, 2020, pp. 9–17.

[74] A. Yazdanbakhsh et al., "In-DRAM near-data approximate acceleration for GPUs," in *Proc. 27th Int. Conf. Parallel Archit. Compilation Techn.*, 2018, pp. 1–14.

[75] F. Devaux, "The true processing in memory accelerator," in *Proc. IEEE Hot Chips 31 Symp.*, 2019, pp. 1–24.

[76] J. Gómez-Luna et al., "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," 2021, *arXiv:2105.03814*.

[77] M. A. Alves et al., "Opportunities and challenges of performing vector operations inside the DRAM," in *Proc. 2015 Int. Symp. Memory Syst.*, 2015, pp. 22–28.

[78] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "McDRAM: Low latency and energy-efficient matrix computations in DRAM," *IEEE Trans. Comput. -Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2613–2622, Nov. 2018.

[79] S. W. Jun et al., "BlueDBM: An appliance for Big Data analytics," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 1–13.

[80] B. Gu et al., "Biscuit: A framework for near-data processing of Big Data workloads," in *Proc. ACM/IEEE 43rd Int. Symp. Comput. Archit.*, 2016, pp. 153–165.

[81] N. Mansouri Ghiasi et al., "GenStore: A high-performance in-storage processing system for genome sequence analysis," in *Proc. 27th ACM Int. Conf. Architectural Support Prog. Lang. Operating Syst.*, 2022, pp. 635–654.

[82] S. Xu, T. Bourgeat, T. Huang, H. Kim, S. Lee, and A. Arvind, "AQUOMAN: An analytic-query offloading machine," in *Proc. IEEE/ACM 53rd Annu. Int. Symp. Microarchit.*, 2020, pp. 386–399.

[83] G. Koo et al., "Summarizer: Trading communication with computing near storage," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 219–231.

[84] Y. Kang, Y. -S. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart SSD," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–12.

[85] C. Jiang, D. Ojika, B. Patel, and H. Lam, "Optimized FPGA-based deep learning accelerator for sparse CNN using high bandwidth memory," in *Proc. IEEE 29th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2021, pp. 157–164.

[86] N. Brown, "Exploring the acceleration of nekbone on reconfigurable architectures," in *Proc. IEEE/ACM Int. Workshop Heterogeneous High-Perform. Reconfigurable Comput.*, 2020, pp. 19–28.

[87] N. P. Jouppi et al., "Ten lessons from three generations shaped Google's TPUv4i: Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 1–14.

[88] H. Liao, J. Tu, J. Xia, and X. Zhou, "DaVinci: A scalable architecture for neural network computing," in *Proc. Hot Chips Symp.*, 2019, pp. 1–44.

[89] Y.-C. Kwon et al., "25.4 A 20nm 6GB function-in-memory DRAM, based on HBM2 with a 1.2 TFLOPS programmable computing unit using bank-level parallelism, for machine learning applications," in *Proc. IEEE Int. Solid-Statist. Circuits Conf.*, 2021, pp. 350–352.

[90] S. Lee et al., "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 43–56.

[91] M. He et al., "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 372–385.

[92] C.-Y. Lo, P. -T. Huang, and W. Hwang, "Energy-efficient accelerator design with 3D-SRAM and hierarchical interconnection architecture for compact sparse CNNs," in *Proc. IEEE 2nd Int. Conf. Artif. Intell. Circuits Syst.*, 2020, pp. 320–323.

[93] K. Ueyoshi et al., "QUEST: Multi-purpose log-quantized DNN inference engine stacked on 96-MB 3-D SRAM using inductive coupling technology in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 186–196, Jan. 2019.

[94] K. Shiba et al., "A 3D-stacked SRAM using inductive coupling technology for AI inference accelerator in 40-nm CMOS," in *Proc. 26th Asia South Pacific Des. Automat. Conf.*, 2021, pp. 97–98.

[95] F.-K. Hsueh et al., "TSV-free FinFET-based monolithic 3D+-IC with computing-in-memory SRAM cell for intelligent IoT devices," in *Proc. IEEE Int. Electron Devices Meeting*, 2017, pp. 12.6.1–12.6.4.
[96] S. Srinivasa et al., "ROBIN: Monolithic-3D SRAM for enhanced robustness with in-memory computation support," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 66, no. 7, pp. 2533–2545, Jul. 2019.
[97] J. Ahn et al., "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 105–117.
[98] A. Boroumand et al., "CoNDA: Efficient cache coherence support for near-data accelerators," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit.*, 2019, pp. 629–642.
[99] K. Hsieh et al., "Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems," in *Proc. ACM/IEEE 43rd Int. Symp. Comput. Archit.*, 2016, pp. 204–216.
[100] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2016, pp. 126–137.
[101] X. Zhang, S. L. Song, C. Xie, J. Wang, W. Zhang, and X. Fu, "Enabling highly efficient capsule networks processing through a PIM-based architecture design," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit*, 2020, pp. 542–555.
[102] M. Drumond et al., "The Mondrian data engine," in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 639–651.
[103] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *Proc. Annu. Int. Symp. Microarchit.*, 2018, pp. 655–668.
[104] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 336–348.
[105] X. Dong et al., "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in *Proc. Des. Automat. Conf.*, 2008, pp. 554–559.
[106] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM l2 cache for CMPs," in *Proc. IEEE 15th Int. Symp. High-Perform. Comput. Archit.*, 2009, pp. 239–249.
[107] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das, "Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 69–80.
[108] M. S. Hosseini, M. Ebrahimi, P. Yaghini, and N. Bagherzadeh, "Near volatile and non-volatile memory processing in 3D systems," *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 3, pp. 1657–1664, Third Quarter 2022.
[109] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.
[110] W.-S. Khwa et al., "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *Proc. IEEE Int. Solid-Statist. Circuits Conf.*, 2018, pp. 496–498.
[111] J. Lee, D. Shin, Y. Kim, and H. -J. Yoo, "A 17.5-fJ/bit energy-efficient analog SRAM for mixed-signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2714–2723, Oct. 2017.
[112] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.
[113] M. Ali, A. Jaiswal, S. Kodge, A. Agrawal, I. Chakraborty, and K. Roy, "IMAC: In-memory multi-bit multiplication and ACcumulation in 6T SRAM array," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2521–2531, Aug. 2020.
[114] S. K. Gonugondla, M. Kang, and N. R. Shanbhag, "A variation-tolerant in-memory machine learning classifier via on-chip training," *IEEE J. Solid-State Circuits*, vol. 53, no. 11, pp. 3163–3173, Nov. 2018.
[115] S. Yin, Z. Jiang, J. -S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.
[116] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *Proc. IEEE Int. Solid-Statist. Circuits Conf.*, 2018, pp. 488–490.
[117] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016.
[118] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2017, pp. 481–492.
[119] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. Int. Symp. Comput. Archit.*, 2018, pp. 383–396.
[120] J. Wang et al., "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, Jan. 2020.
[121] D. Fujiki et al., "Duality cache for data parallel acceleration," in *Proc. Int. Symp. Comput. Archit.*, 2019, pp. 397–410.
[122] K. Lee, J. Jeong, S. Cheon, W. Choi, and J. Park, "Bit parallel 6T SRAM in-memory computing with reconfigurable bit-precision," in *Proc. Des. Automat. Conf.*, 2020, pp. 1–6.
[123] V. Seshadri et al., "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2013, pp. 185–197.
[124] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (LISA): Enabling fast inter-subarray data movement in DRAM," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2016, pp. 568–580.
[125] H. Hassan et al., "CROW: A low-cost substrate for improving DRAM performance, energy efficiency, and reliability," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit.*, 2019, pp. 129–142.
[126] X. Xin et al., "Reducing DRAM access latency via helper rows," in *Proc. Des. Automat. Conf.*, 2020, pp. 1–6.
[127] Y. Wang et al., "FIGARO: Improving system performance via fine-grained in-DRAM data relocation and caching," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 313–328.
[128] V. Seshadri et al., "Fast bulk bitwise AND and OR in DRAM," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 127–131, Jul./Dec. 2015.
[129] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-memory compute using off-the-shelf DRAMs," in *Proc. Annu. Int. Symp. Microarchit.*, 2019, pp. 100–113.
[130] V. Seshadri et al., "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. Annu. Int. Symp. Microarchit.*, 2017, pp. 273–287.
[131] Y. Yang, X. Chen, and Y. Han, "Dadu-CD: Fast and efficient processing-in-memory accelerator for collision detection," in *Proc. Des. Automat. Conf.*, 2020, pp. 6–11.
[132] X. Xin, Y. Zhang, and J. Yang, "ELP2IM: Efficient and low power bitwise operation processing in DRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 303–314.
[133] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proc. IEEE/ACM 50th Annu. Int. Symp. Microarchit.*, 2017, pp. 288–301.
[134] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM based accelerator for accurate CNN inference," in *Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf.*, 2018, pp. 1–6.
[135] S. Li et al., "SCOPE: A stochastic computing engine for DRAM-based in-situ accelerator," in *Proc. IEEE/ACM 51st Annu. Int. Symp. Microarchit*, 2018, pp. 696–709.
[136] N. Hajinazar et al., "SIMDRAM: A framework for bit-serial SIMD processing using DRAM," in *Proc. 26th ACM Int. Conf. Architectural Support Prog.. Lang. Operating Syst.*, 2021, pp. 329–345.
[137] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "LAcc: Exploiting lookup table-based fast and accurate vector multiplication in DRAM-based CNN accelerator," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.
[138] J. D. Ferreira et al., "pLUTo: Enabling massively parallel computation in DRAM via lookup tables," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit.*, 2022, pp. 900–919.
[139] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "NAND-Net: Minimizing computational complexity of in-memory processing for binary neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 661–673.
[140] F. M. Bayat, X. Guo, H. A. Om'Mani, N. Do, K. K. Likharev, and D. B. Strukov, "Redesigning commercial floating-gate memory for analog computing applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2015, pp. 1921–1924.
[141] X. Guo et al., "Temperature-insensitive analog vector-by-matrix multiplier based on 55 nm NOR flash memory cells," in *Proc. Custom Integr. Circuits Conf.*, 2017, pp. 1–4.
[142] C. Gao et al., "ParaBit: Processing parallel bitwise operations in NAND flash memory based SSDs," in *Proc. IEEE/ACM 54th Annu. Int. Symp. Microarchit.*, 2021, pp. 59–70.

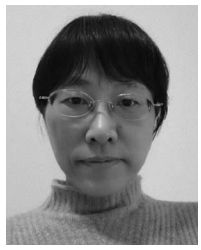
[143] J. Park et al., "Flash-cosmos: In-flash bulk bitwise operations using inherent computation capability of NAND flash memory," in *Proc. IEEE/ACM 55th Int. Symp. Microarchit.*, 2022, pp. 937–955.
[144] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Int. Symp. Comput. Archit.*, 2016, pp. 14–26.
[145] C. Li et al., "Analogue signal and image processing with large memristor crossbars," *Nat. Electron.*, vol. 1, no. 1, pp. 52–59, 2018.
[146] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. IEEE/ACM 43rd Int. Symp. Comput. Archit*, 2016, pp. 27–39.
[147] M. Hu, Y. Chen, J. J. Yang, Y. Wang, and H. H. Li, "A compact memristor-based dynamic synapse for spiking neural networks," *IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.*, vol. 36, no. 8, pp. 1353–1366, Aug. 2017.
[148] I. Boybat et al., "Neuromorphic computing with multi-memristive synapses," *Nat. Commun.*, vol. 9, no. 1, pp. 1–12, 2018.
[149] T. Tuma, A. Pantazi, M. Le Gallo, A. Sebastian, and E. Eleftheriou, "Stochastic phase-change neurons," *Nat. Nanotechnol.*, vol. 11, no. 8, pp. 693–699, 2016.
[150] S. Lashkare, S. Chouhan, T. Chavan, A. Bhat, P. Kumbhare, and U. Ganguly, "PCMO RRAM for integrate-and-fire neuron in spiking neural networks," *IEEE Electron Device Lett.*, vol. 39, no. 4, pp. 484–487, Apr. 2018.
[151] X. Sun and S. Yu, "Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 9, no. 3, pp. 570–579, Sep. 2019.
[152] I. Chakraborty, M. Fayez Ali, D. Eun Kim, A. Ankit, and K. Roy, "GENIEx: A generalized approach to emulating non-ideality in memristive xbars using neural networks," in *Proc. IEEE/ACM 57th Des. Automat. Conf.*, 2020, pp. 1–6.
[153] W. Zhang et al., "Design guidelines of RRAM based neural-processing-unit: A joint device-circuit-algorithm analysis," in *Proc. IEEE/ACM 56th Des. Automat. Conf.*, 2019, pp. 1–6.
[154] S. Jain et al., "CxDNN: Hardware-software compensation methods for deep neural networks on resistive crossbar systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 6, pp. 1–23, 2019.
[155] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.
[156] S. Angizi et al., "AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.
[157] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 531–543.
[158] M. Imani et al., "DUAL: Acceleration of clustering algorithms using digital-based processing in-memory," in *Proc. IEEE/ACM 53rd Annu. Int. Symp. Microarchit.*, 2020, pp. 356–371.
[159] J. Nider, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Processing in storage class memory," in *Proc. 12th USENIX Conf. Hot Top. Storage File Syst.*, pp. 13–13, 2020.
[160] D. Fujiki et al., "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.
[161] Y. Zha and J. Li, "Hyper-Ap: Enhancing associative processing through a full-stack optimization," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 846–859.
[162] S. Mittal, "A survey of techniques for architecting and managing GPU register file," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 16–28, Jan. 2017.
[163] I. Fernandez et al., "NATSA: A near-data processing accelerator for time series analysis," in *Proc. IEEE 38th Int. Conf. Comput. Des.*, 2020, pp. 120–129.
[164] S. Gudaparthi et al., "Wire-aware architecture and dataflow for CNN accelerators," in *Proc. Annu. Int. Symp. Microarchit.*, 2019, pp. 1–13.
[165] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. IEEE Hot chips 23 Symp.*, 2011, pp. 1–24.
[166] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, "Recryptor: A reconfigurable cryptographic cortex-M0 processor with in-memory and near-memory computing for IoT security," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, Apr. 2018.
[167] C. Zhang, Y. Tang, and H. Liu, "Late line-of-sight check and partially updating for faster any-angle path planning on grid maps," *Electron. Lett.*, vol. 55, no. 12, pp. 690–692, 2019.
[168] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-memory acceleration of deep neural network training with high precision," in *Proc. Int. Symp. Comput. Archit.*, 2019, pp. 802–815.
[169] N. Hajinazar et al., "The virtual block interface: A flexible alternative to the conventional virtual memory framework," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 1050–1063.

Changwu Zhang received the BE degree from the Northwestern Polytechnical University, Xian, China, in 2017, and the MS degree from the National University of Defense Technology (NUDT), Changsha, China, in 2019. He is currently working toward the PhD degree with NUDT. His research interests include computer architecture, energy efficient computing, robotic planning and intelligent algorithms.

Hao Sun received the MS degree from the College of Computer from National University of Defense Technology, Changsha, China, in 2020. He is currently working toward the PhD degree in computer science and technology with the College of Computer, National University of Defense Technology. His research interest concentrates on design space exploration for deep learning accelerator.

Shuman Li received the BE and MS degrees from the National University of Defense Technology (NUDT), Changsha, China. She is currently working toward the PhD degree in computer science and technology with NUDT. Her research interests include high performance computing, computer vision, computational fluid dynamics and underwater robots.

Yaohua Wang received the PhD degree from the National University of Defense Technology (NUDT), in 2013, where he is currently a professor with the Computer College. His research interests include computer architecture, memory system and hardware-software codesign.



Haiyan Chen received the MS degree in mechanical and electronic engineering from the National University of Defense Technology, Changsha, China, in 1992. She is a professor of National University of Defense Technology. Her main research interests include microprocessor architecture and VLSI design.

Hengzhu Liu received the PhD degree in electrical engineering from the National University of Defense Technology (NUDT), in 1999, where he is currently a professor with the Computer College. His research interests include microprocessor architecture, high-performance computing and computer vision.