Survey on Near-Data Processing: Applications and Architectures

Paulo C. Santos, Luigi Carro
Federal University of Rio Grande do Sul
(pcssjunior, carro)@inf.ufrgs.br

Tiago R. Kepe
Federal Institute of Paraná
tiago.kepe@ifpr.edu.br

Francis B. Moreira, Aline S. Cordeiro, Sairo R. Santos, Marco A. Z. Alves
Federal University of Paraná
(fbm, ascordeiro, srsantos, mazalves)@inf.ufpr.br

***Abstract*—One of the main challenges for modern processors is the data transfer between processor and memory. Such data movement implies high latency and high energy consumption. In this context, Near-Data Processing (NDP) proposals have started to gain acceptance as an accelerator device. Such proposals alleviate the memory bottleneck by moving instructions to data whereabouts. The first proposals date back to the 1990s, but it was only in the 2010s that we could observe an increase in papers addressing NDP. It occurred together with the appearance of 3D-stacked chips with logic and memory stacked layers. This survey presents a brief history of these accelerators, focusing on the applications domains migrated to near-data and the proposed architectures. We also introduce a new taxonomy to classify such architectural proposals according to their data distance.**

***Index Terms*— Near-Data Processing (NDP); Near-Memory Accelerator (NMA); In-Memory Accelerators (IMA); Near-Cell Accelerators (NCA); Taxonomy;**

I. Introduction

Since the 1960s, Moore's law guided processor manufacturing technology, where smaller transistors paved the way for ever-faster processing units. However, main memory technology and manufacturing processes presented much slower advancements due to different trade-offs and design points [1, 2, 3]. This difference in latency led to a performance gap between processing units and memory devices, the well-known memory-wall bottleneck. The processor industry adopted data-latency hiding strategies such as multi-threading [4], bigger Reorder Buffer (ROB) and Memory Order Buffer (MOB) [5], complex memory hierarchies, non-blocking caches and aggressive prefetching [6]. Nevertheless, current processors increased the pressure on the memory due to the increasing number of cores and the trend towards vector instructions (NEON, MMX, SSE, and AVX, among others). By the same time, the memory industry started to provide multiple data channels and memory controllers, introducing parallelism between memory modules, delivering data at higher bandwidth to the processor cores [7].

An emerging approach to enable fast computation on data-driven applications is to move computation closer to the data, reducing the latency and energy consumption of transferring data between memory and processor. This idea dates back to the 1990s [8, 9] when the industry was unable to integrate Dynamic Random Access Memory (DRAM) and logic cells on the same die. Around the 2010s, NDP architectures became a viable solution with the release of 3D-stacked architectures such as Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) [10, 11]. These architectures use Through-Silicon Vias (TSVs) [12] to bundle logic and DRAM layers, adding processing elements to the same chip as DRAM. Meanwhile, the rise of memristor technologies enabled a different approach to processing near-data, with multiple proposals of memory cells capable of performing different logic and arithmetic operations [13, 14].

These new data-centric architectures extend the von Neumann model by adding part of the processing capabilities near-data. Such extension mitigates data movement between memory and processor and enhances time and energy consumption while presenting data parallelism, thus enabling high processing bandwidth near-data. Such architectures are ideal for streaming and vector applications, such as databases, numerical modeling, and other applications with a low temporal locality. These algorithms have low data-reuse and reduce the cache memory's effectiveness, as they stream large data sets and evict data from caches with no reuse. For algorithms that also present high spatial locality (i.e., coalescent pattern), near-data processing has even higher potential by exploring the inherent spatial locality of memory row buffers.

Several years' worth of research in NDP have generated a large body of pertaining scientific papers. In this survey, we present a systematization of such work. While other authors have attempted to adequately classify existing NDP architectures, here we look to previous work from the prism of the application domains that benefit from data proximity. We also aim to highlight how hardware has evolved in a way that enables and supports NDP architectures and elucidate why certain application domains tend to make the move to near-data solutions. Our main objectives are:

- To present application domains most migrated to NDP.
- To propose a taxonomy to the architectural designs to NDP.
- To discuss the main programming and offloading models of these new architectures.

In this paper, we will use four terms to refer to different architectural approaches (explained in Section III): **Near-Data Processing (NDP)** to generically describe any computation model that performs computation closer to the memory; **Near-Memory Accelerator (NMA)** to proposals that place computation outside the memory die; **In-Memory Accelerators (IMA)** to proposals that place computation inside the memory die; **Near-Cell Accelerators (NCA)** to proposals that perform computation by adding small logic near the memory cells.

Digital Object Identifier 10.29292/jics.v16i2.502

II. Near-data Applications

This section discusses the application domains most commonly migrated to NDP. Thereby, we will understand where most efforts are and which domains are being neglected. Although innumerable algorithms could be candidates for near-data processing, NDP devices exploit two main principles: (i) a **data streaming behavior** guarantees a constant flow of data from/to memory. A streaming behavior in the traditional processor reduces the effectiveness of cache memory, increasing power dissipation and energy consumption due to data movement. (ii) by enabling **coalescent data access**, the application allows NDP to access large amounts of data at once, which translates to high usage of memory bandwidth;

To illustrate the migration of applications to NDP, Figure 1 shows a simple experiment performed in a simulation setup equal to previous work [15, 16] modeling a multi-core processor with a 16 MB Last-Level Cache (LLC). Values greater than 1 (as in the figure detail) indicate performance improvement for NDP over the traditional processing architecture. In this experiment, we compare elements in an integer array. This application was executed in an x86 architecture and also using an IMA approach [15]. To understand the cache influence, we vary the number of interactions over the array and the total array size. Besides, to understand the bandwidth utilization, we also vary the number of threads used on the x86 approach.

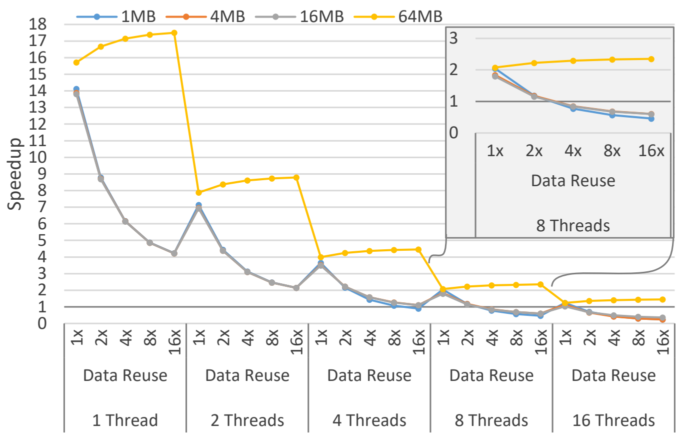


Fig. 1: NDP performance compared to traditional x86.

Results of Figure 1 indicate that applications with a small memory footprint (i.e., smaller than LLC) or high data reuse benefit from the cache hierarchy. Programmers should thus execute these on traditional processors. Conversely, applications with big memory footprints (i.e., greater than LLC size) and low data reuse are the best candidates to migrate to NDP. Therefore, certain application domains are notably suitable for NDP as presented below.

Neural Network: Widely used for pattern recognition and data classification, Neural Network algorithms rely on massive computation over a vast dataset to train model parameters and classify data. The main operation executed is the product of each layer's activation values and connection weights. It happens from input to output layer (i.e., forward-propagation). These algorithms also update the weights' values of each layer's connection from output to input layer (i.e., back-propagation). During the training phase, these operations are executed several times for each instance in a dataset and repeated for a few epochs. Thus, depending on dataset size, these algorithms can behave as a data stream, presenting low performance and low energy efficiency on traditional architectures.

Researchers have used Static Random Access Memory (SRAM) technology to perform IMA in a more familiar technology to fully optimize Neural Network (NN), mainly using binary or ternary neural networks due to the reduced storage capacity. Ando et al., Takamaeda-Yamazaki et al. [17, 18] augment SRAM to efficiently process binary and ternary neural networks, splitting SRAM into lower words (weights from input neurons) and upper words (weights to output neurons) and organizing them into NDP for specialized chips. Eckert et al. [19] propose Neural Cache, a design that adds logic to bit-line peripherals in LLC SRAM arrays to provide bit-serial Functional Units (FUs) for NN computation. Wang et al. [20] extend Neural Cache by adding techniques to leverage sparsity-awareness, NN redundancy, and add new efficient compute algorithms for binary and ternary neural networks. Yin et al. [21] part from the same base idea as Neural Cache, but enable more scalability by using XNOR-Accumulate operations to enable activation of multiple SRAM rows, double buffers to hide in-memory reprogramming latencies, and additional peripheral logic for multi-bit activation. Ramanathan et al. [22] propose the BFree, a bit-line free LUT-based NDP in SRAM subarrays that allows reconfigurable precision and NN layout. Long et al. [23] further optimize NN for these architectures, showcasing their potential for practical usage with LeNet, AlexNet, VGGNet, and ResNet Convolutional Neural Network (CNN) architectures. Aga et al. [24] implement the Compute Cache architecture by adding bit-line computing to support vector processing over large data operands. The authors redesigned memory caches to ensure bit-line operand locality to provide NCA.

Although SRAM offers a promising approach in terms of performance, most of the work that popularized NDP uses DRAM technology for large-scale neural networks. Liu et al. [25], Azarkhish et al. [26], Schuiki et al. [27], and Thottethodi et al. [28] accelerate CNNs in an NDP architecture with additional full cores. For instance, Liu et al. [25] implements programmable ARM-based cores in the 3D-Stacked memory's logic layer, and there are a few functions that are loaded and offloaded to these cores. Gao et al. [29] employ a set of in-memory ARM cores, Translation Look-aside Buffer (TLB) and virtual memory schemes to allow cores to communicate through a vault router by sharing the physical address with the Central Processing Unit (CPU) host. Azarkhish et al. [26] and Schuiki et al. [27] implemented modules composed of multiple RISC-V cores, each one with local cache and Direct Memory Access (DMA), these cores control smaller Processing Elements (PEs). Besides, the authors also connect multiple Hybrid Memory Cubes (HMCs), so they can communicate between themselves. Thottethodi et al. [28] also implements full-cores in 3D-Stacked memory's logic layer, each one with local memory, register file, pipeline, cache, and prefetcher buffers. Gao et al. [30] implement processing cores composed of Arithmetic Logic Units (ALUs) and a register file attached to each vault. These cores are implemented in-memory, share a global buffer, and communicate through a dedicated network to enable parallelism. More focused in communication, Min et al. [31] accelerate a Deep Neural Network (DNN), and Gao et al. [30], Gao et al. [29], and Oliveira et al. [32] accelerate NN in NDP architectures. Min et al. [31] implement vaults' communication with Network-on-Chip (NoC) technology to allow for higher parallelism. Besides, it also schedules and divides functions between the vaults.

Using FUs integration near the Dynamic Random Access Memory (DRAM), Oliveira et al. [32] implement the Neuron In-Memory (NIM), a module composed of a register bank, reconfigurable FUs, and a sequencer that are attached to each vault to simulate biologically meaningful NN. Cordeiro et al. [33] propose the Vector-In-Memory Architecture (VIMA), a IMA processor focused on executing Machine Learning algorithms. It is based on HMC, and it is compound by a small cache memory and vector FUs in the logic layer that execute large vector instructions to exploit data reuse and achieve higher performance. Other proposals accelerate convolutional and binary neural networks considering NDP in a conventional DRAM, by adding logic ports and bitwise operations such as Li et al. [34], citealldeng2019lacc, Sim et al. [35], Deng et al. [36], and Cadambi et al. [37]. Li et al. [34], Deng et al. [36], and Deng et al. [38] implement reconfigurable circuits inside the memory. Sim et al. [35] and Cadambi et al. [37] execute partial computations to achieve better performance during NN layer computations. Sim et al. [35] adds extra digital blocks to the peripheral area of a DRAM, while Cadambi et al. [37] implements a set of interconnected PEs with local memory caches to implement NDP.

In recent years, as the promise of efficient memristor physical implementations comes closer to reality, several researchers started proposing the use of Resistive random-access memory (ReRAM) for neural networks due to the efficient implementation of dot product operation using memristor crossbar arrays. Chi et al. [39] propose PRIME, a ReRAM main memory architecture where a subset of the memory arrays are configured either for neural network inference acceleration or memory storage. Cheng et al. [40] improve PRIME proposing the TIME architecture that adds peripheral circuits to support training and weight update, along with variability-free and gradual writing to reduce the number and impact of writes to the memristors. Shafiee et al. [41] propose the ISAAC, an in-situ analog arithmetic crossbar to efficiently accelerate DNN inference using dot product replacing traditional computation with an efficient, deep pipeline. Song et al. [42] improve on ISAAC proposing the PipeLayer, which enables training DNNs, simplifying the pipeline to reduce the penalty of bubbles, and using spike-based data input and output to eliminate signal conversions. Haj-Ali et al. [43] extend the MAGIC architecture [44] to support complex operations, enabling image-processing tasks. Gupta et al. [45] propose the FELIX, a purely in-cell 1-cycle logic implementation, along with bit line segmentation to further increase parallelism and use the derived complex operations for image classification. Imani et al. [46] propose the mechanism called FloatPIM, implementing floating-point representation and operations through bitwise NOR memristor operations on digital store, eliminating signal conversions and providing better performance and inference accuracy over the PipeLayer and ISAAC. Kwon et al. [47] present the mechanism called FIMDRAM, which proposes integrating an engine capable of large vector operations to the DRAM, thus exploiting bank-level parallelism and achieving performance up to $4\times$ superior processing bandwidth in comparison to an off-chip device. Lee et al. [48] present the architecture Similarity Search Associative Memory (SSAM), an accelerator for similarity search applied to K-Nearest Neighbors (KNN) search, which outperforms Graphics Processing Unit (GPU)- and Field-Programmable Gate Array (FPGA)-based alternatives in terms of throughput and energy efficiency. Drebes et al. [49] propose the TC-CIM, a fully automatic, end-to-end compilation framework based on tensor comprehensions and loop tactics that enables the users to exploit NDP transparently. The goal is to provide NCA interface based in different memory technologies to accelerate machine learning kernels.

Finally, Angizi et al. [50] compare analog and digital implementations of near-cell accelerator designs. They conclude ReRAM offers the smallest area, SRAM offers the lowest latency, and Spin-Transfer Torque Random-Access Memory (STT-RAM) offers the lowest energy consumption when considering NCA implementations.

Graph Traversing / Pointer Chasing: Efforts in different fields of computer architecture have been made to speed up data structure operations for big data workloads. A graph is a conventional data structure that can store the data value and represent the relationships among data. To abstract a graph, we have $G = (V, E)$, where V denote vertex and E the edges from a set of G. When processing a graph, the main problems are random access leading to a poor access locality in the cache memories and unbalance in the workloads when using multiple processing units. Considering that graphs represent relations between elements, depending on the way algorithms traverse the vertexes, different memory access patterns may occur, leading to highly irregular accesses. Such irregularity leads to bandwidth degradation between CPU-DRAM, with poor usage of cache memory. Several graph applications require the accessed data to determine the following pointer address to be accessed, forming a serially dependent chain of loads called pointer chasing. Finally, due to graphs' natural irregularity, applications that rely on such data structures suffer from unbalance and race conditions when parallelism is present.

To illustrate the efforts on graph domain, we show five state-of-the-art mechanisms that couple NDP with pointer-chasing capabilities, in both application and data-structure levels, using DRAM technology. The Tesseract mechanism [51] aims to accelerate large-scale graph processing workloads by exploring the internal bandwidth of HMC with a set of 16 memory cubes. A simple ARM processor was inserted into each memory vault of all memory cubes (the whole mechanism has 512 ARM processors). Hsieh et al. [52] present the In-Memory PoInter Chasing Accelerator (IMPICA), to accelerate pointer-chasing. The authors observed that linked lists and B-trees have sparse memory access patterns, hurting performance by increasing cache memory misses. The mechanism integrates two different engines into the memory cube's vaults. The authors Liu et al. [53] presented a theoretical work to estimate gains when executing concurrent pointer-chasing operations with NDP. Their analytical module obtains estimates using the number of atomic operations, the number of memory accesses, and the maximum number of accesses the cache can deliver. Nai et al. [54] propose a framework called GraphPIM that efficiently utilizes NDP for graph computing by enabling instruction-level NDP offloading for generic graph computing frameworks with minor changes in both software and hardware. The authors implement a NDP offloading unit in the host processor. Thus, atomic operations bypass the cache hierarchy and are sent directly to the IMA. The authors of [55] design a NDP mechanism to accelerate linked-list applications. They observed that simply offloading the computation near-data did not significantly improve performance due to poor data locality. Thus, they came up with two mechanisms, NDP-aware data localization and batching, to take advantage of the internal bandwidth provide by HMC devices.

Proposing architectures specialized for graph applications using ReRAM, Song et al. [56] propose the GraphR, a NDP based on ReRAM and a streaming-apply computation model. It consists of ReRAM for storage and ReRAM crossbars for graph processing, functioning as an out-of-core accelerator given a specific pre-processing mapping. Huang et al. [57] implement an heterogeneous NDP architecture using both ReRAM and traditional logic, using a hardware-software co-design to fully benefit from NDP.

Genome Sequencing/Pattern Matching: Widely used in bioinformatics, genome sequencing involves several tasks related to identifying DNA sequences in samples, including counting, alignment and sequence assembly. However, these tasks are non-trivial due to the size of the datasets involved and the characteristics of the application. For example, a single DNA sample generates tens of millions of sequences that must then be mapped to known datasets with billions of sequences. Thus, genome sequencing tasks suffer from memory-wall bottlenecks due to the massive amount of data movement required. The same issue occurs with other big data applications that similarly rely on pattern matching tasks, such as network security and data mining, increasing the interest for NDP to improve this application domain.

Two proposals use SRAM technology to execute efficient matching algorithms. Sadredini et al. [58] discuss how existing NDP pattern matching accelerators fail to use resources to their fullest extent. The authors propose a general-purpose pattern matching architecture called Impala that implements efficient multi-stride NCA automata processing by addressing these issues. Cali et al. [59] presented the GenASM, a framework for approximate string matching designed for genome sequence analysis and used to accelerate various steps in the sequencing process. We also found two works that explore DRAM technology. Angizi et al. propose the AlignS [60] using Spin-Orbit Torque Magnetoresistive Random-Access Memory (SOT-MRAM) and an assembler [61] using DRAM NCA accelerators to aid DNA sequence alignment and assembly, respectively. Huangfu et al. [62] created the NEST, a NDP architecture that accelerates k-mer counting, which is another important task in DNA sequencing processes. RAPID [63], on the other hand, proposes an NCA memristor-based architecture that also supports DNA alignment tasks considering a parallel version of the state-of-the-art algorithm.

Computational Fluid Dynamics: Multiple areas from science utilize applications that simulate Computational Fluid Dynamics (CFD), solving, for instance, Navier-Stokes, Poisson's equations, and also solving linear systems. Several CFD algorithms utilize similar kernels, such as matrix-multiplication and matrix convolution with stencils. This domain commonly use big arrays performing vector operations.

Zhu et al. [64] create the components called Logic-in-Memory (LIM) to embed as a stack in HMC, underlying the logic layer. Using their previously created specialized custom logic cores [65], the authors enable efficient sparse matrix multiplication, used in many graph applications, and Fast Fourier Transform. Nair et al. [66] implemented the AMC architecture based on HMC, and they incorporate on its logic layer vector registers, a vector instruction set, predicated execution, virtual addressing, and gather-scatter accesses directly to memory. At the same time, it is connected directly to the host processor, without the need for cache hierarchy and hardware scheduling of instructions. Alves et al. [7] [67] implement the Memory Vector eXtension (MVX), in which a set of vector FUs are implemented inside the DRAM to perform near-data computing, reducing the data movement between the processor and main memory. The idea is to access data directly in the sense amplifiers of the open rows and operate it in the vector FUs implemented along an additional register bank inside the DRAM. Alves et al. [15] proposed the HMC Instruction Vector Extensions (HIVE) that integrates vector units inside the HMC. The authors enable the vault parallelism executing vector instructions in their IMA. Ahmed et al. [68] proposed the PRIMO, a compiler to support NDP architectures capable of offloading instructions and exploit FUs automatically at compile-time, thus achieving a speedup of up to $15\times$ for the kernels tested.

Database: Analytical database workloads are a compelling case for NDP. These workloads process many queries that move large amounts of data between memory and computing units to look for patterns and relationships and compute aggregate values. Such algorithms pollute the cache memories with dead-on-arrival data. An analytical query consists of a chain of database operators that interact to generate the final result. The most relevant group of operators are projection, selection, join, and aggregation, corresponding to 90% of execution time and memory usage on a traditional analytic workload [69]. The majority of near-data approaches focused on these operators to exploit the internals of NDP micro-architectures. Traditionally, Database Management Systems (DBMS) handle data in storage such as disk or Solid-State Drive (SSD). Thus, Kim et al. [70] proposed a dedicated scan-processor in the Flash memory controllers of an SSD to execute the selection operator. Below we selected proposals that use memory levels closer to the processor for the more recent in-memory DBMS.

Mirzadeh et al. [71] proposed a near-memory join unit inside the HMC to execute the radix-hash and parallel sort-merge join algorithms. JAFAR [72] adds off-chip dedicated hardware connected to the memory I/O buffer that serves as a proxy between CPU and DRAM to filter data through selection predicates. Ambit [73] requires minor changes to adapt current DRAM memories for bit-wise operations using NCA. Tomé et al. [16] proposed the HMC Instruction Prediction Extensions (HIPE) that takes advantage of the logic layer in 3D-stacked memories to perform near-data memory filters. Santos et al. [74] propose the Reconfigurable Vector Unit (RVU), which enables adaptive in-memory processing through the reconfigurable vector processing units placed on the HMC. Tiago Rodrigo Kepe and Alves [69] carried out a performance analysis on database operators over 3D-memories and x86 processor, delivering helpful insights and distinguishing cases to run the operators either near-data or in the traditional CPU.

Sun et al. [75] implement DBMS restriction, projection, and aggregate operators in a ReRAM-based NDP. Through the capability of accessing the memristor crossbar in both directions, they enable efficient analytic and transaction queries.

MapReduce: MapReduce (MR) is a programming model for automatic parallel processing to extract information from large files. It relies on two primitives: Map and Reduce, inspired by Lisp and Haskell programming languages. The map function classifies the dataset and produces a likely huge intermediate set of key-value pairs. Typically, an MapReduce (MR) framework distributes those intermediate sets around processing units to perform the reduce phase. The Reduce function groups the values that share the same key and aggregates them. Multiple MR instances shall be needed to compute the final result. Therefore, a single MR application requires moving large amounts of data between memory and computing units.

In our survey, we only found MapReduce accelerators that use DRAM memory. NDCores [76] evaluated MR applications over a chain of four 3D-stacked memories to explore data access parallelism and the high memory bandwidth. The authors also design data-sorting hardware to boost the sort phase of MR frameworks [77]. Farmahini-Farahani et al. [78] propose an NMA that stacks FU-based accelerators on top of commodity DRAM devices, connected through Through-Silicon Vias (TSVs). It leverages the high parallelism and localized memory accesses of MapReduce frameworks, reportedly consuming 46% less energy and achieving 1.67× the processor's performance implementing the same logic. Mondrian [79, 80] implements an algorithm-hardware co-design using the 3D-stacked memory for NDP of data analytics operators, which also apply to the data partitioning and shuffling phase of MapReduce. On the other hand, the Memory Channel Network (MCN) [81] presents another perspective of NDP on a cluster of servers. A MCN processor is integrated with a buffer device on a memory to compute MR frameworks Hadoop and Spark.

Multiple Domains: Here we refer to proposals that employ NDP to improve different applications, mainly comprehending the applications explained above, such as Pattern Matching, Graph Search, Hash, Matrix Multiplication, Neural Networks, Image Processing, and other similar applications.

Some authors rely on DRAM memory approaches adding full-cores or implementing near-cell circuitry to achieve better computational performance. Devaux [82] present the UPMEM, which adds full-cores alongside the main memory die, thus avoiding most data movement and increasing energy efficiency. Gao et al. [83] propose changing standard DRAM memory controller by adjusting its operation timings, which allows their solution to perform logical operations in parallel. The strategy uses standard DRAM chips and forces them to open multiple rows concurrently in quick succession. Xin et al. [84] created the ELP2IM. This low power NCA architecture uses DRAM sense amplifier states to implement operations that reduce data movement within memory subarrays and concurrent open rows, thus achieving high performance and energy efficiency. Hajinazar et al. [85] present the SIMDRAM, a general-purpose framework that enables implementation of complex operations in DRAM-based NDP devices.

Jain et al. [86] propose the STT-CiM, an IMA design that uses STT-RAM. The authors explore the possibility of enabling word lines simultaneously and computing data directly in the memory cells improving performance and energy efficiency. Xie et al. [87] propose a set of logical and arithmetic operations implemented on ReRAM and also a novel multiplication algorithm that can also be implemented on their solution. Their results suggest a 46% speedup in comparison to the state-of-the-art ReRAM alternative.

Summary: In order to get a complete picture of NDP for different application domains, Table I summarizes the application domains and architecture dimensions. We classify each proposal using a different application domain according to 4 areas: Accelerator's distance from the data, memory technology, processing model, and programming/offload model. Each category from these areas is further explained in the following sections.

First, we can observe that Machine Learning is the application domain that received the most attention up to date. Second, we could even observe that authors that were focusing on other domains such as CFD or MapReduce, now are focusing their efforts on Machine Learning (ML). We can also observe that all these domains have in common low data reuse. However, only the graph processing/pointer chasing domain focus on non-coalescent memory access.

We could observe that most NCA proposals focus on

TABLE I: Applications and architecture dimensions.

Domain	Papers	Accelerator Placement			Memory Technology			Architecture Model					Offload Model			
		Near-Cell	In-Mem.	Near-Mem.	SRAM	DRAM	ReRAM	Full Proc.	Func. Unit	Near-Cell	FPGA/ CGRA	ASIC	Host Trigger Inst.	Routine	Binary	Not Def.
Neural Network	[39] 2016	x					x			x				x		
	[41] 2016	x					x			x					x	
	[24] 2017	x			x					x			x			
	[34] 2017	x				x				x					x	
	[40] 2017	x					x			x				x		
	[42] 2017	x					x			x					x	
	[19] 2018	x			x					x			x			
	[36] 2018	x				x				x			x			
	[35] 2018	x				x				x			x			
	[43] 2018	x					x			x			x			
	[45] 2018	x					x			x					x	
	[20] 2019	x			x					x			x			
	[21] 2019	x			x					x			x			
	[38] 2019	x				x				x			x			
	[46] 2019	x					x			x				x		
	[22] 2020	x			x					x			x			
	[23] 2020	x			x					x					x	
	[49] 2020	x					x			x					x	
	[37] 2010		x			x		x							x	
	[29] 2015		x			x		x							x	
	[17] 2017		x		x							x			x	
	[32] 2017		x			x			x				x			
	[30] 2017		x			x						x			x	
	[25] 2018		x			x		x				x		x	x	
	[26] 2018		x			x		x				x			x	
	[27] 2018		x			x		x				x			x	
	[48] 2018		x			x		x							x	
	[28] 2018		x			x		x							x	
	[31] 2019		x			x						x	x			
	[47] 2021		x			x						x			x	
	[33] 2021		x			x			x				x			
Graph / Pointer Chasing	[56] 2018	x					x			x					x	
	[57] 2020	x					x			x					x	
	[51] 2015		x			x		x						x		
	[55] 2016		x			x						x		x	x	
	[52] 2016			x		x			x						x	
	[54] 2017			x		x						x		x		
Genetics	[63] 2019	x					x			x					x	
	[58] 2020	x			x					x					x	
	[61] 2020	x				x				x			x			
	[62] 2020		x			x				x					x	
	[59] 2020		x		x							x	x			
Comput. Fluid Dynamics	[64] 2013		x			x						x		x		
	[65] 2013		x			x						x		x		
	[67] 2015		x			x			x				x			
	[7] 2015		x			x			x				x			
	[66] 2015		x			x		x					x			
	[15] 2016		x			x			x				x			
	[68] 2019		x			x			x				x			
Data Base	[73] 2017	x				x				x			x			
	[75] 2017	x					x			x					x	
	[71] 2015		x			x						x	x			
	[74] 2017		x			x			x				x			
	[16] 2018		x			x			x				x			
	[69] 2019		x			x			x				x			
	[72] 2015			x		x			x					x		
Map Reduce	[76] 2014			x		x		x							x	
	[78] 2015			x		x					x		x			
	[77] 2015			x		x						x		x		
	[79] 2017			x		x			x						x	
	[80] 2018			x		x			x						x	
	[81] 2018			x		x		x							x	
Multiple Domains	[86] 2018	x					x			x			x			
	[87] 2019	x					x			x			x			
	[84] 2020	x				x				x						x
	[85] 2021	x				x				x			x			
	[88] 2016		x			x					x		x			
	[82] 2019		x			x		x								x
	[83] 2019		x			x				x				x		
	[89] 2014			x		x		x						x		
Total	**70 papers**	**29**	**31**	**10**	**9**	**47**	**14**	**13**	**13**	**31**	**2**	**14**	**29**	**13**	**28**	**2**

providing a particular functionality for the application, which means that it is likely that such mechanisms will not receive the complete application offloading.

In our analysis we could observe that multiple proposals claims to perform near-data processing using NAND-Flash technology [90, 91, 92, 93, 94, 95, 96]. However, most proposals merely used the full processor with cache memory already coupled next to the flash memory. Such proposals were not considered in the table because we understand that these accelerators are too distant from data, not even closer to the NAND-Flash's memory controller.

We found no CFD or graph accelerator using SRAM technology. The reduced storage capacity constrains the size of applications that can leverage such specialized hardware.

III. Near Data Taxonomy

As aforementioned in Section II, near-data architectures have emerged aiming to accelerate a myriad of applications. In the last decades, these architectures have evolved in several flavors. Due to their intent to achieve high efficiency, different characteristics related to *distance* from the data, *memory technology* adopted and *processing model* applied, *programming model* supported and *code offloading* mechanisms are few from many that can be observed.

NDP generally aims to mitigate the memory-wall problem by taking advantage of the memory's internally available bandwidth. In addition, NDP focuses on reducing the costs caused by the transferring of data between processing units and main memory (i.e., DRAM), as such movement is a significant bottleneck and source of inefficiency for computer systems [3]. By adopting this approach, it is possible to improve overall performance and energy efficiency.

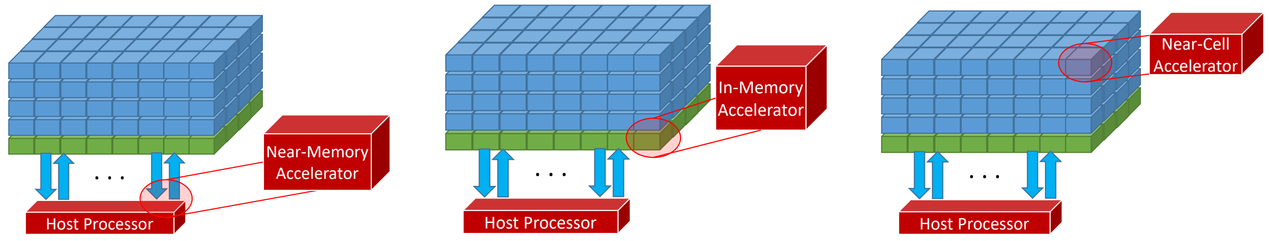
To allow processing mechanisms to speedup and increase efficiency on applications as cited in Section II, different Near-Data Accelerators (NDAs) can be placed at distinct points depending on memory technology and hierarchy, and application target. Figure 2 illustrates our taxonomy that classifies all the NDP proposals regarding their data distance.

Near-Cell Accelerators (NCA): NCA approaches position the accelerators within the memory cells, either by modifying them or by taking advantage of their analog behavior to compute data while transferring data via internal buses toward sense amplifiers. This approach has been experimented on typical DRAM-based memories [34, 35, 36, 38, 61, 62, 73]. Also, new memory technologies (e.g., ReRAM, STT-RAM) have been gaining space due to its inherent capacity to process data [39, 41, 40, 42, 43, 46, 56, 57, 60, 63, 75, 86, 87]. Figure 2c illustrates how close to the data this type of accelerator can be found. The main advantage of this approach is to thoroughly reduce data movement, since it operates inside the memory as close as possible to the memory cells. Therefore, it can explore the maximum internal bandwidth, achieving near-optimal energy efficiency in data transfer. However, by being that close to the memory cells, it is limited to few operations, such as *OR, AND, COPY* [83, 97]. To support complex operations (e.g., *ADD, MUL*), in-cell's approach demands either additional hardware or a special data layout that stores data in a columnar way [19, 83, 98].

In-Memory Accelerators (IMA): Accelerators are placed within memory devices on the same silicon piece, either by placing logic between memory layers [47], or by taking advantage of the 3D-stacked integration technologies to accommodate NDP capabilities on the logic layer. Considering Single Data Rate (SDR) and Double Data Rate (DDR) memories, several techniques were proposed to process data inside these memories by integrating the processing logic into the DRAM row-buffers. However, integrating processing logic inside the traditional SDR or DDR memories is a challenging task, because logic circuits fabricated in DRAM-optimized process technology are much slower than similar circuits in a logic-optimized process technology [99]. Recently, several proposals take advantage of 3D integration technologies to insert a custom logic in a layer of DRAM devices. In both 2D and 3D scenarios, we could observe full-core integration proposals [26, 25, 27, 28, 29, 37, 48, 51, 66, 76, 81, 82, 89]. A different approach is to move only the functional units near-data. Several proposals [7, 15, 16, 32, 33, 52, 68, 69, 74, 79, 80] instead of the full processors integration, they add only logical and arithmetic units near-data, enabling large vector processing. Other proposals integrate fine and coarse grain reconfigurable logic inside a logic layer [78, 88]. Finally, several proposals integrate custom Application-Specific Integrated Circuits (ASICs) able to accelerate only specific applications [25, 26, 27, 30, 31, 47, 64, 65, 71].

Figure 2b illustrates where this type of NDP element is commonly placed. This type of NDP aims to leverage the internal memory bandwidth, but this type may have more area and power dissipation since it is placed on a logic layer. The main advantage of these approaches are to obtain data straight from the row-buffers. The row-buffers offer large data parallelism which can feed multiple processing elements. Such proposals does not require off-chip data transfers which consumes time and energy. They relies on wider data buses of TSVs to transfer data between the memory device and the processing logic [100, 101]. However, by being that close to the memory array present limitations in terms of power consumption and heat dissipation, as well as limited area. Such limitations cannot be neglected by the architects, especially when their proposals implements full-processors, complex cache hierarchies, on a logic layer within the 3D-chip.

Near-Memory Accelerator (NMA): Accelerators are placed outside the memory devices in a different silicon die, either by placing logic next to the device using an *interposer* or off-chip interconnections [52, 54, 72, 76, 77, 78, 79, 80, 81, 89, 102, 103, 104]. Nevertheless, some proposals may also propose to add logic inside the memory controller inside the processor. Hashemi et al. [103] propose to migrate instructions to the functional units inside the memory controller to fast solve data dependency in long dependency chains, reducing thus the average memory stalls. Figure 2b illustrates where this type of NDP element is commonly placed. This type of NDP aims to reduce the memory hierarchy latency. Such



(a) Near-Memory Accelerators (b) In-Memory Accelerators (c) Near-Cell Accelerators

Fig. 2: Most common types of NDP. The blue blocks represent storage cells, green blocks represent logic layer (for 3D designs) or the sense amplifier (for 2D designs) and the arrows are the interconnection to outside the memory chip.

proposals present less constraints in terms of area and energy. However, these proposals may suffer from latency and energy consumption for off-chip communications.

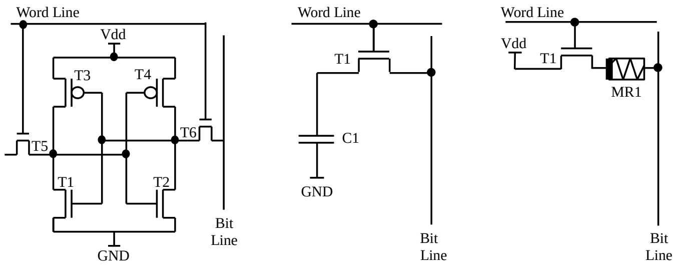
IV. Near-Data Technology and Architectures

Although there are a wide variety of accelerators that adopt the same principles of approaching logic and data, these designs are delineated by the adopted memory technology and architecture. In this section we focus on the memory technology and architectures generally used for NDP. In the next section we will discuss the NDParchitecture itself.

A. *Memory Technologies and Volatility*

Memories can be implemented based on different technologies and purpose of use, hence typically they are classified as volatile and non-volatile.

Volatile memories, such as SRAM and DRAM-based memories, usually are near to the host processors storing temporary data. These technologies are widely present in modern computer systems and are the essence of their memory hierarchy. Furthermore, these memory technologies have changed little in recent decades, benefiting only from the shrinkage provided by the manufacturing technology evolution (i.e., currently 5 nm for SRAM and 10 nm for DRAM). Figures 3a and 3b illustrates the differences between these memory cells.



(a) SRAM Cell (b) DRAM Cell (c) ReRAM Cell

Fig. 3: Circuit of the different memory cells technologies.

The major NDP proposals are coupled with these memory technologies, 66% of related work present on Table I relies on DRAM-based technology, mainly due to their well-understood technology and manufacturing processes. Among volatile memories, DRAM-based devices are the most common for NDP implementation, due to their density, and the

TABLE II: Memory technologies comparison.

	SRAM	DRAM	ReRAM
Maturity	Product	Product	Early Dev.
Cell Size	$> 100\ F^2$	$6 - 8\ F^2$	$> 5\ F^2$
Read Lat.	$< 10ns$	$10 - 60ns$	$10 - 100ns$
Write Lat.	$< 10ns$	$10 - 60ns$	$10 - 100ns$
Dyn. Energy*	$> 1pJ$	$2pJ$	$0.02pJ$
Stat. Power	Yes	Yes	No
Endurance**	$> 10^{15}$	$> 10^{15}$	$> 10^{12}$
Volatile	Yes	Yes	No

* Energy per bit access. ** Number of accesses.
Data extracted from Perez et al. [105]

large amount of data that can be directly accessed by those accelerators. Moreover, as these modules are separated from the processor, they may have fewer constraints, which allow more room to be explored as an independent device module.

Non-Volatile memories, on the other hand, (i.e., ReRAM, Performance Counter Monitor (PCM), STT-RAM, NAND Flash) differ from volatile memories in that they retain data after powering off. Moreover, these memories have a significant density when compared to volatile memories. For instance, the size of a SRAM cell is 100 F^2 (5 nm), while a ReRAM cell is 4 F^2 (22 nm) (Table II). Thus, for applications that demand vast volumes of data, it is interesting to avoid data movement from these non-volatile memories to volatile memories and then to the processor's core. Figure 3c illustrates memory cells used for ReRAM devices (transistor and memristor).

Although many new non-volatile technologies have emerged in the last decades, ReRAM have appeared as the most prominent for NCA adoption, since memristors' inherent characteristics allow between-cells computation. In these cases, NCA designs coupled with non-volatile memories can take advantage of data movement reduction to improve overall efficiency, 20% of related work present on Table I relies on ReRAM-based technology with NCA.

B. *Memory Architectures and NDP*

To better explore the available memory technologies, memory cells are arranged in a determined architecture, which adopts distinct techniques to allow large bandwidth, and to provide the highest performance. Commonly, memory cells comprises matrices that can be accessed by row. Matrices can be grouped to form banks, and a set of banks can be placed to form a device. This technique can be applied for many memory technologies, from SRAMs to the new ReRAM.

Figure 2 illustrates a modern DRAM-based memory device designed in a 3D-stacked fashion. However, regardless the architecture and integration method adopted, all modern memories comprise many memory banks that can be accessed concurrently, hence mitigating the memory cells' latencies and improving parallel data access.

In a NDP implementation, proposals try to directly access these large matrices to improve bandwidth, computing over large portions of data and avoiding data movement, then improving performance and overall efficiency. NDPproposals have explored many memory architectures. In the literature, SRAM-based NDP proposals mostly aim to insert logic capabilities to the host's cache memories or to the host's memory controllers [17, 19, 20, 21, 22, 23, 24, 58, 59, 103, 106]. This work modify the cache hierarchy trying to avoid moving data from the main memory and cache memories to the host's core.

In case of typical 2D DDR DRAM-based modules, NDP designers have placed either entire cores alongside memory devices [81, 82], or placed customized logic units to exploit large vector accesses [7, 67, 72], or explored the internal analog mechanisms by changing the memory access pattern [34, 62, 73, 83, 84, 85, 97].

With the advent of 3D-stacked technologies, NDP designs was leveraged by the idea of integrating logic and memory on the same chip. The most well-known 3D-stacked memory commercial examples are HMC [10] and High Bandwidth Memory (HBM) [11]. These architectures stack DRAM layers and a logic layer that can implement memory controllers, or a Physical Layer (PHY) to properly connect external links to the memory cells. The communication between all memory layers and the logic layer is done through TSV, a vertical path that can cross all and provide seamless access from any to any layer. Originally, HMC [10, 107], was the first commercial memory able to present a NDP integrated to the memory. HMC integrates memory controllers within its logic device (on per memory vault [107, 108]). Additionally, each memory controller has a special set of FUs able to compute arithmetic and logic operations over up 16 Bytes [109]. Recently, Samsung' HBM presented the HBM-PIM [47], a NDP integrated to a HBM module. It comprises of several sets of FUs placed within each memory bank in each layer, which means that for a device consisted of 4 memory layers and 32 vaults (or channels) with 32 banks per layer, 128 independent NDP are available [47].

Both 3D-stacked memory architectures largely increase bandwidth, as shown in Table III, which may be limited by the host's capacities and links. Therefore, the insertion of NDP within these devices can exploit the huge internal bandwidth and improve overall performance and efficiency for many applications (Section II). In the literature, many proposals have relied on 3D integration technology. Similar to NDP on typical DDR memory modules, 3D-stacked memories are also target for NDP of different flavors [15, 16, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 47, 48, 64, 65, 66, 68, 69, 71, 74, 76, 79, 80, 88, 110], which will be discussed on Section V.

Nonetheless, non-volatile memories also adopt matrices organization to increase the bandwidth. For instance, in case of memristors, a crossbar connects many memory cells, thus allowing to access an entire row at once. On these memory devices, the large the row is, the bigger will be the theoretical processing capabilities, since the computation is done by accessing multiple distinct rows [39, 41, 40, 42, 43, 45, 46, 49, 56, 57, 60, 63, 75, 86, 87]

In general, NDP architectures primarily take advantage of the internally available memory bandwidth, thus theoretically, memories capable of large bandwidth allows NDP to achieve highest performance and overall efficiency. To illustrate the difference between memory technologies and architectures, Table III presents information characteristic of typical memories used in NDP studies and implementations.

TABLE III: DRAM architectures comparison.

Memory Name	Maximum Bandwidth	Maximum Speed*	Energy Usage	JEDEC Compliant
DDR	3.2 GB/s	0.4 GT/s	257.13 pJ/b	Yes
DDR2	6.4 GB/s	0.8 GT/s	121.44 pJ/b	Yes
DDR3	14.9 GB/s	1.8 GT/s	64.70 pJ/b	Yes
DDR4	25.6 GB/s	3.2 GT/s	38.67 pJ/b	Yes
DDR5	41.6 GB/s	5.2 GT/s	N.A.	Yes
HMC	320 GB/s	2.5 GT/s	10.82 pJ/b	No
HBM1	128 GB/s	1.2 GT/s	N.A.	Yes
HBM2	256 GB/s	2.0 GT/s	N.A.	Yes
HBM2**	310 GB/s	2.4 GT/s	N.A.	Yes
HBM2e	410 GB/s	3.2 GT/s	N.A.	Yes

* Data rate/pin. ** From 2018.

V. NDP's Architecture Model

There have been several solutions presented in the literature for the memory-wall problem by adopting processing near or in memory. Despite the similar objective, NDP designs possess fundamentally different approaches [111]:

A. Full-Stack Processor

These NDP proposals bring entire CPU-like units to the memory chip. While taking advantage of more common programming models and data coherence mechanisms, these solutions face substantial constraints from power and area in the memory chips [112]. The solutions proposed in [25, 51, 79, 82, 89, 110] added full general-purpose cores to the logic layer. In traditional CPU-like accelerators (e.g., General Purpose Processor (GPP), GPU, Tensorflow), although the performance is commonly measured from the processing logic's point of view, the LLC is the default data entry point, making it the main bottleneck in terms of on-chip bandwidth [113, 114]. For instance, using a full GPP or GPU as a memory logic elevates the complexity of the design and does not intrinsically solve the main problems faced with NDP integration, namely programming model, virtual address translation, and cache coherence, as these challenges are objectives in many studies [115, 116, 117]. Moreover, the costs in terms of power and area are severe, being an actual limiting factor [112].

B. Simple Functional Units

Implementing simple FUs within memory modules, either in same memory PCB or by taking advantage of 3D-stacking integration, is an approach that can allow to access the internally available memory bandwidth. A set of FUs can work in a Single Instruction Multiple Data (SIMD) manner, therefore exploiting SIMD operations to extract highest bandwidth. Computational-RAM (C-RAM) [9] proposed adding multiple functional units alongside the DRAM's sense amplifiers, thus allowing computation at the bit level. This strategy appeared in 1990's, however it had significant requirements such as controlling and powering new processing elements according to DRAM sub-array activation patterns, adding a large number of functional units [118], and expecting the operating system to maintain a very specific mapping of the data used by applications. The main challenge for this work was the integration between logic and memory.

In the 2010's, mainly due to the advent of 3D-stacking integration, this approach reappeared as a better fit for power and area constrained devices. However, this type of NDP requires innovative solutions for programming models, cache coherence, and virtual memory support [115, 116, 117]. Many works [7, 15, 16, 32, 52, 67, 68, 69, 72, 74, 79, 80], proposed the use of custom FU-like logic to exploit the bandwidth of memories. However, all of these works rely heavily on host-side hardware modifications for integration, hence presenting a severe limitation, as no current system can natively support such NDP. Several authors avoid tackling these requirements by reserving memory space exclusively for the NDP devices, changing the logical memory type (e.g., *uncacheable*) fully bypassing cache hierarchies, or explicitly neglecting virtual-memory and data coherence [79, 82].

C. Computing in-Memory Cells

The most disruptive approach to overcome the memory-wall is to think of the memory as a computational device. This approach takes advantage of the memory's analog circuitry to process data by allowing multiple cells to be accessed simultaneously, then computing while transferring data between the memory cells and the sense-amplifiers. In the DRAM memory, this is accomplished by sharing capacitor charges. [36, 73, 83, 85, 97]. In the same way, new technologies (e.g., ReRAM and STT-RAM) use electrical resistance, exploiting Kirchhoff's Law, to compute on stored data [19, 24, 34, 39, 41, 42, 49, 56, 84, 86, 87].

Similarly to the simple FUs, this solution also require modifications on host's side to provide instructions offloading (Instruction Set Architecture (ISA) extension, memory controllers must be aware of the new memory functionalities), cache coherence and virtual memory support [83, 85, 115]. Moreover, since some of these proposals rely on not yet commercially available technology, and others require heavy modifications on the host or in connections to the memory cells, there are gaps in solutions for data coherence, code offloading, and virtual memory.

D. FPGA/CGRA

Although area and power constraints are important limitations for NDP designs, some works experiment placing FPGA and Coarse-Grain Reconfigurable Array (CGRA) in memory [78, 88, 119, 120]. This class of programmable arrays can allow the implementation of on-demand logical units, hence theoretically providing near-optimal hardware-software coupling for certain applications.

To keep general-purpose complaint, these approach require similar concerns present on simple FU implementation, since they need to keep data coherent between host and accelerator, as well as the accelerator must have a solution to support virtual memory. Additionally, by adopting programmable arrays, it is necessary to emit the *bitstream*, which is the file that describe the hardware to be configured. For this, a efficient solution also must be provided.

E. Specialized Accelerators

ASIC and Specialized Units are usually provided to compute determined class of application, for example to efficiently compute certain classes of neural networks, or for operate over huge graphs [17, 25, 26, 27, 30, 31, 47, 54, 59, 64, 65, 71, 77]. In the literature, some works provide specialized hardware to compute such applications. Although these NDP may adopt similar hardware as presented in previous sections (i.e., FUs), in these cases, general-purpose compatibility and similar issues are left aside. Since they are intended to specialized systems, these types of NDP avoid workarounds to keep cache coherence, virtual memory support, and usually do not consider overheads any modifications on the host.

VI. Programming Model and Code Offloading

The programming model depends directly on the NDP design, since each type requires different code generation and code offloading strategies.

A. Host Triggers Instructions

The NCA designs exploit the memories' inherent capabilities, while IMA may reserve area and power budget for execution units. Both approaches usually cannot afford complex mechanisms such as those present in *superscalar*, out-of-order processors (e.g., large *fetch*, *multiple-instruction decode*, and *reordering buffers* stages), as well as *virtual to physical* memory address translation. Therefore, such approaches must rely on code emission directly from the host processor. For such, the host processor must support the NDP ISA, thus requiring modifications to the host as aforementioned in Section V. Moreover, since the host is responsible for triggering all instructions, the binary will present a host and accelerator instruction mix. In these cases, two main approaches can be highlighted:

Manual code generation: may use *intrinsics* explicitly written by the programmer [121]. This approach allows for fine control of the generated code, giving specific instructions to the compiler that must appear in the final binary. The proposal presented in [121] is inspired by Intel Intrinsics [122], a library available in C language with a set of routines. When a program calls a routine from this library, it embeds its internal assembly x86 code directly in the compiler to optimize the execution. Thus, these routines allow low-level code optimization, including code vectorization, and the adoption of SIMD instructions. Nevertheless, vendors create and distribute intrinsic libraries accordingly to the ISA extension available inside each processor.

Automatic code generation: relies on compiler support to automatically generate code. The compiler must tackle the instruction offloading decision and automatically optimize the code for a given architecture. When code involves two or more dependent or independent ISAs, the compiler back-end must be aware of the host and the NDP's ISA and their relationship to provide the correct instruction sequence. In the literature, several proposals address NDP offloading decisions, while others deal with hardware exploitation.

Regarding offloading decisions, Hadidi et al. [123] presents Compiler-Assisted InstRuction-level Offloading (CAIRO), a compiler to support native HMC instructions [109]. In CAIRO, the IMA instructions offloading candidates are decided by the number of cache misses, bandwidth savings, and the overhead of host atomic instructions respectively. These decisions are taken offline because it requires a cache profiling tool to record the traces of previous execution of applications. Moreover, each version of an application needs to be compiled, profiled, and then analyzed by CAIRO. After these steps, the HMC-atomic instructions are included if found suitable.

Ahmed et al. [68] proposed Processing-In-Memory cOmpiler (PRIMO), a complete compiler support for NDP, including both the offloading and architecture-oriented optimization steps. In PRIMO, both the offloading and exploitation of FUs happens automatically at compile time. PRIMO is intended for hardware exploitation without the burden of using *pragma* or directive-based approach, and its instruction footprint is comparable to the footprint of a native *AVX-512* code.

B. *Function or Binary Migration*

NDP designs that adopt full-processors [25, 51, 79, 82, 89, 110], can take advantage of traditional programming models, and they can be supported by libraries to provide code offloading and communication between host and accelerators. These designs usually rely on well-known libraries such as Open Multi-Processing (OpenMP), Compute Unified Device Architecture (CUDA).

For exploiting NDP hardware, Asynchronous Memory Compiler (AMC) is presented by Nair et al. [66], which adopts a directive-based approach. A program annotated to run on parallel systems with OpenMP 4.0 directives is analyzed to discover which parts will run on the AMC. AMC can also exploit complete vector units and pipeline features present on the accelerator lanes to derive an effective schedule of instructions for each lane.

Similarly, the work from Hsieh et al. [124] presents Transparent Offloading and Mapping (TOM), which is a compiler-based solution that allows computation offloading to multiple 3D-stacked memories in a GPU-based system. This approach is concerned about the amount of code that will be sent to the GPU's main memory. In TOM, instruction blocks with potential for maximum memory savings are initially identified. It then decides whether the selected candidates shall be offloaded based on runtime system conditions. TOM is based on CUDA annotated code.

Further, Khaldi and Chapman [125] present a new compiler pass called Bandwidth-Critical Data Analysis (BCDA) for detecting code dealing with bandwidth critical data to allocate data to HBM. The allocation calls are transformed into specific HBM allocation calls which are transparent to the user. This solution is automatic in the manner that *malloc* calls are changed to *memkind_alloc* for memory bandwidth bound codes. Although the work of Khaldi and Chapman [125] is intended for automatic data organization, its technique relies on special functions, which requires programmer attention. Moreover, this work is not concerned with code offloading, and hence it is limited to a more generic NDP interface.

Another work that relies on the *function offloading* model is HBM-PIM [47]. It demands all memory to be stopped, reconfigured, and then the host processor is responsible for sending the code snippet or the addresses from where the code will be computed by the IMA. This code is previously annotated since several restrictions must be considered, such as data sharing between many NDP units, hence requiring a specific Application Programming Interface (API).

C. *Hardware Driven*

Another approaches rely on in-flight instruction offloading, by monitoring code behavior, or hardware metrics (e.g., cache misses). In Hashemi et al. [103], the proposal aims to migrate instructions to the NDP placed within the memory controller for execution as soon as source data arrives from DRAM. This technique adopts a mix of ISA extension and metrics monitoring. This migration technique allows to minimize dependent cache miss latency by 20%.

VII. Related Work

In this section, we summarize the relevant surveys on NDP from the last six years, between 2014 and 2020. The first relevant survey published by Balasubramonian et al. [126] presents key insights captured by the organizers and keynote speakers of the first near-data processing workshop organized during the 46th MICRO conference. The authors' first argument for the resurgence of NDP is the limitation of current architectures to continue shrinking transistor sizes and increasing the number of cores. The second argument from the authors for NDP is the development of technologies that enable or facilitate NDP, such as 2.5D or 3D die stacking technology that enables easy integration of logic and memory layers. Besides, distributed software frameworks, such as MapReduce, create easy support and demand for such architectures. These frameworks already address challenging NDP software issues, such as data layout, scheduling, and fault tolerance. At the same time, it provides ready-to-use software that requires efficient architectures, sparking the interest in the hardware industry to provide such needs. Finally, the industry has shown maturity in heterogeneity handling in recent years with research on GPUs and ARM's big.LITTLE, which provides an excellent background to support NDP accelerators.

Since this publication, applications ranging from genome string processing to database management systems have all found successful implementations in NDP [29, 59, 63, 69, 127, 128, 129], which increasingly motivates the move to a NDP-enabled architecture. All the potential shown by such proposals motivated the creation of commercially available NDP architecture, such as the UPMEM's [82] and the Samsungs' NDP-enabled HBM for artificial intelligence[1]. This rise of NDP hardware led the current research focuses on improving programming models and usability [117] so that NDP can be used by any programmer.

The survey from Singh et al. [130] analyzes and organizes the NDP proposals using four distinct classes: (i) Computation in-Memory; (ii) Processing Near Heterogeneous Memory; (iii) Processing Near Storage-Class Memory, and; (iv) Processing Near Main-Memory. A subdivision is made depending on type of implementation: (a) programmable; (b) fixed functions or; (c) reconfigurable computation. The authors identify that the main challenges of NDP are two, first, in providing support for virtual memory and cache coherence which is heavily neglected in the proposals. Another challenge is the lack of compatibility between processor ISA and NDP ISA. The proposals miss information regarding the interaction between the host and the NDP and most do not address the code generation for NDP ISA. Moreover, the authors attempt to define a design space exploration for NDP, performing application characterization and performance evaluation to estimate the potential of NDP. They conclude that higher cache miss ratios imply increased benefits from NDP.

Mutlu et al. [131] survey's makes a major distinction considering Processing Using Memory (PUM) and Processing In-Memory (PIM). PUM refers to techniques that use the current DRAM substrate or new memory technology substrates to perform specialized functions with minimal changes, adding no logic to the substrate. These techniques focus on data initialization, data movement, and basic boolean logic operations. PIM refers to research on additional logic near or inside memory modules that aim to provide high-bandwidth and low latency for specific instructions, functions, or applications. Thus, the authors Du Nguyen et al. [132] have a similar classification, although they focus on the programming model aspect of this classification. The authors conclude that it is still unclear which granularity of NDP provides the best trade-off. Coarser granularities are harder to program and are usually application-specific but allow much more efficient memory usage. Meanwhile, finer granularities such as instructions can seamlessly replace current instructions through the compiler but require more communication and overhead. Besides, it might generate hard-to-tackle problems in cache coherence and virtual memory.

[1]https://news.samsung.com/global/samsung-develops-industrys-first-high-bandwidth-memory-with-ai-processing-power

In contrast to other surveys, Du Nguyen et al. [132] includes NCA model. Their classification takes into account three metrics: (i) where the computation is performed; (ii) the memory technology, and; (iii) the computation parallelism. The first metric generates four classes: Computation In-Memory Array or Periphery (CIM-A / CIM-P), Computation Out-of-Memory Near or Far (COM-N / COM-F). The distinction made here between Computation In-Memory (CIM) and Computation Out-of-Memory (COM) is analogous to the one made by Mutlu et al. [131]. The second metric could generate multiple classes due to the varied number of memory technologies (SRAM, DRAM, ReRAM, Magnetoresistive Random-Access Memory (MRAM), PCM, STT-RAM, etc), but the authors distinguish only "charge-based" and "non-charge based" memories. Lastly, the third metric, the computation parallelism, further divides NDP research into three classes: task parallelism, data parallelism, or instruction parallelism, which can be otherwise seen as programming models. The authors indicate the infeasibility of NCA for SRAM, defending thus the emerging technologies such as memristors for the next NCA architectures. The authors also note several gaps in the classification under which they were not able to find any work and thus present a direction for several works in NDP.

Considering these previous surveys we could notice the main conclusions regarding NDP are:
• The best memory technology for NDP is still unclear;
• Most architectures require specific changes to the code;
• The programming and interface must evolve to support adaptive scheduling, data mapping, and access/sharing control;
• Few researches provide answers for virtual memory and cache coherence support, which still an issue;
• Interconnection between the NDP units is often neglected;
• 3D stacking requires novelty in heat sinks, as computing units will increase energy drastically;
• Infrastructure to assess benefits and feasibility must be improved. Very few tool-flow are open source, and usually, these are restricted to specific architectures;

TABLE IV: Survey comparison

	NDP Architectures	Memory Techs.	Processing Model	Offload Model	Application Domain
Singh et al. [130]	x	x	x		x
Du Nguyen et al. [132]	x	x			
Mutlu et al. [131]	x			x	x
Ours	x	x	x	x	x

Considering the multiple surveys over the NDP theme, we compare their primary focus on Table IV. The first survey is not cited in this table as their focus was primarily to share their insights from the first near-data processing workshop.

We could observe on the previous surveys that as NDP research evolves, so does the classification that applies to the results of such efforts. We believe the classification proposed in this present survey adequately categorizes and encompasses most, if not all, current NDP research. The two main distinguishing features of NDP proposals are computation location (NCA, IMA or NMA) and computation granularity (basic operations through instructions, specialized functions, or generic processing for any application). Furthermore, the ”using memory” category proposed by Mutlu et al. [131] is safely addressed by the computation granularity of specialized functions. We also propose a more descriptive categorization of memory technologies that reflect current NDP research tendencies, along with classification for the offloading models commonly used.

VIII. Conclusions

This survey intended to collect and highlight the most notable work presented in recent years, providing insights from the main application domains and also introducing a new taxonomy for Near-Data Processing (NDP).

The presented taxonomy classifies NDP work in Near-Cell Accelerators (NCA), In-Memory Accelerators (IMA) or Near-Memory Accelerator (NMA), according to the accelerator placement in relation to the memory it is coupled. NCA proposals started to flourish after 2016 along with the rise of Resistive random-access memory (ReRAM) proposals. Both were enabled by advances in the signal sensing logic of memory technologies. Nevertheless, IMA started to appear three years earlier with the advent of 3D integration technologies.

We could observe that the main application domain being recently migrated for NDP is Machine Learning (ML), in particular the neural networks, where most of the proposals are using NCA. Moreover, we noticed that several authors working with Computational Fluid Dynamics (CFD), MapReduce, and other domains are now researching on ML proposals.

Regarding memory technologies, although most initiatives still use Dynamic Random Access Memories (DRAMs), ReRAM approaches have been gaining strength in the last few years. Besides, Static Random Access Memory (SRAM) models are less frequent and limited mainly by their low storage capabilities.

Observing the proposals, we could observe a lack of infrastructure for programming most of the NDP architectures. Moreover, no standard programming models are available. Even commercially available products are still in their infancy in terms of Application Programming Interfaces (APIs) and programming support.

Acknowledgements

This work was partially supported by FAPERGS, CAPES, CNPq and Serrapilheira Institute (grant number Serra-1709-16621).

References

[1] W. A. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” *SIGARCH Computer Architecture News*, 1995.

[2] A. Nowatzyk, F. Pong, and A. Saulsbury, “Missing the memory wall: The case for processor/memory integration,” in *Int. Symp. on Computer Architecture (ISCA)*, 1996.

[3] D. P. Zhang, N. Jayasena *et al.*, “A new perspective on processing-in-memory architecture design,” in *SIGPLAN Workshop on Memory Systems Performance and Correctness*, 2013.

[4] D. M. Tullsen, S. J. Eggers, and H. M. Levy, “Simultaneous multithreading: Maximizing on-chip parallelism,” in *Int. Symp. on Computer Architecture (ISCA)*, 1995.

[5] A. Cristal, O. J. Santana *et al.*, “Toward kilo-instruction processors,” *Trans. on Architecture and Code Optimization (TACO)*, 2004.

[6] T.-F. Chen and J.-L. Baer, “Reducing memory latency via non-blocking and prefetching caches,” *ACM SIGPLAN Notices*, 1992.

[7] M. A. Alves, P. C. Santos *et al.*, “Saving memory movements through vector processing in the dram,” in *Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2015.

[8] D. Patterson, T. Anderson *et al.*, “A case for intelligent ram,” *IEEE Micro*, 1997.

[9] D. G. Elliott, M. Stumm *et al.*, “Computational ram: Implementing processors in memory,” *IEEE Design & Test of Computers*, 1999.

[10] Hybrid Memory Cube Consortium, “Hybrid memory cube specification 2.1,” 2014, http://www.hybridmemorycube.org/.

[11] H. Jun, S. Nam *et al.*, “High-bandwidth memory (hbm) test challenges and solutions,” *IEEE Design & Test*, 2017.

[12] J. V. Olmen, A. Mercha *et al.*, “3D stacked IC demonstration using a through silicon via first approach,” in *Int. Electron Devices Meeting*, 2008.

[13] E. Lehtonen and M. Laiho, “Stateful implication logic with memristors,” in *Int. Symp. on Nanoscale Architectures (ISNA)*, 2009.

[14] T. M. Taha, R. Hasan *et al.*, “Exploring the design space of specialized multicore neural processors,” in *Int. Joint Conference on Neural Networks (IJCNN)*, 2013.

[15] M. A. Z. Alves, M. Diener *et al.*, “Large vector extensions inside the hmc,” in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2016.

[16] D. G. Tomé, P. C. Santos *et al.*, “HIPE: HMC instruction predication extension applied on database processing,” in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2018.

[17] K. Ando, K. Ueyoshi *et al.*, “Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w,” *Journal of Solid-State Circuits*, 2017.

[18] S. Takamaeda-Yamazaki, K. Ueyoshi *et al.*, “Accelerating deep learning by binarized hardware,” in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conf. (APSIPA ASC)*, 2017.

[19] C. Eckert, X. Wang *et al.*, “Neural cache: Bit-serial in-cache acceleration of deep neural networks,” in *Int. Symp. on Computer Architecture (ISCA)*, 2018.

[20] X. Wang, J. Yu *et al.*, "Bit prudent in-cache acceleration of deep convolutional neural networks," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2019.

[21] S. Yin, Z. Jiang *et al.*, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *Trans. on Very Large Scale Integration (VLSI) Systems*, 2019.

[22] A. K. Ramanathan, G. S. Kalsi *et al.*, "Look-up table based energy efficient processing in cache support for neural network acceleration," in *Int. Symp. on Microarchitecture (MICRO)*, 2020.

[23] Y. Long, E. Lee *et al.*, "Q-pim: A genetic algorithm based flexible dnn quantization method and application to processing-in-memory platform," in *Design Automation Conf. (DAC)*, 2020.

[24] S. Aga, S. Jeloka *et al.*, "Compute caches," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2017.

[25] J. Liu, H. Zhao *et al.*, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *Int. Symp. on Microarchitecture (MICRO)*, 2018.

[26] E. Azarkhish, D. Rossi *et al.*, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *Trans. on Parallel & Distributed Systems*, 2018.

[27] F. Schuiki, M. Schaffner *et al.*, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *arXiv preprint arXiv:1803.04783*, 2018.

[28] M. Thottethodi, T. Vijaykumar *et al.*, "Millipede: Die-stacked memory optimizations for big data machine learning analytics," in *Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2018.

[29] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Parallel Architecture and Compilation (PACT)*, 2015.

[30] M. Gao, J. Pu *et al.*, "Tetris: Scalable and efficient neural network acceleration with 3d memory," *ACM SIGOPS Operating Systems Review*, 2017.

[31] C. Min, J. Mao *et al.*, "Neuralhmc: an efficient hmc-based accelerator for deep neural networks," in *Asia and South Pacific Design Automation Conf. (ASPDAC)*, 2019.

[32] G. F. Oliveira, P. C. Santos *et al.*, "Nim: An hmc-based machine for neuron computation," in *Int. Symp. on Applied Reconfigurable Computing (ARC)*, 2017.

[33] A. S. Cordeiro, S. R. dos Santos *et al.*, "Machine learning migration for efficient near-data processing," in *Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, 2021.

[34] S. Li, D. Niu *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Int. Symp. on Microarchitecture (MICRO)*, 2017.

[35] J. Sim, H. Seol, and L.-S. Kim, "Nid: processing binary convolutional neural network in commodity dram," in *Int. Conf. on Computer-Aided Design (ICCAD)*, 2018.

[36] Q. Deng, L. Jiang *et al.*, "Dracc: a dram based accelerator for accurate cnn inference," in *Design Automation Conf. (DAC)*, 2018.

[37] S. Cadambi, A. Majumdar *et al.*, "A programmable parallel accelerator for learning and classification," in *Int. Conf. on Parallel architectures and Compilation Techniques (PACT)*, 2010.

[38] Q. Deng, Y. Zhang *et al.*, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *Design Automation Conf. (DAC)*, 2019.

[39] P. Chi, S. Li *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, 2016.

[40] M. Cheng, L. Xia *et al.*, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Design Automation Conf. (DAC)*, 2017.

[41] A. Shafiee, A. Nag *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, 2016.

[42] L. Song, X. Qian *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2017.

[43] A. Haj-Ali, R. Ben-Hur *et al.*, "Not in name alone: A memristive memory processing unit for real in-memory processing," *IEEE Micro*, 2018.

[44] S. Kvatinsky, D. Belousov *et al.*, "Magic—memristor-aided logic," *Trans. on Circuits and Systems*, 2014.

[45] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *Int. Conf. on Computer-Aided Design (ICCAD)*, 2018.

[46] M. Imani, S. Gupta *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Int. Symp. on Computer Architecture (ISCA)*, 2019.

[47] Y.-C. Kwon, S. H. Lee *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *Int. Solid-State Circuits Conf. (ISSCC)*, 2021.

[48] V. T. Lee, A. Mazumdar *et al.*, "Application codesign of near-data processing for similarity search," in *Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2018.

[49] A. Drebes, L. Chelini *et al.*, "Tc-cim: Empowering tensor comprehensions for computing-in-memory," in *Int. Workshop on Polyhedral Compilation Techniques (IMPACT)*, 2020.

[50] S. Angizi, Z. He *et al.*, "Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?" in *Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.

[51] J. Ahn, S. Hong *et al.*, “A scalable processing-in-memory accelerator for parallel graph processing,” in *Int. Symp. on Computer Architecture (ISCA)*, 2015.

[52] K. Hsieh, S. Khan *et al.*, “Accelerating pointer chasing in 3d-stacked memory: Challenges, mechanisms, evaluation,” in *Int. Conf. on Computer Design (ICCD)*, 2016.

[53] Z. Liu, I. Calciu *et al.*, “Concurrent data structures for near-memory computing,” in *Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

[54] L. Nai, R. Hadidi *et al.*, “Graphpim: Enabling instruction-level pim offloading in graph computing frameworks,” in *Int. Symp. on high performance computer architecture (HPCA)*, 2017.

[55] B. Hong, G. Kim *et al.*, “Accelerating linked-list traversal through near-data processing,” in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, 2016, pp. 113–124.

[56] L. Song, Y. Zhuo *et al.*, “Graphr: Accelerating graph processing using reram,” in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2018.

[57] Y. Huang, L. Zheng *et al.*, “A heterogeneous pim hardware-software co-design for energy-efficient graph processing,” in *Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2020.

[58] E. Sadredini, R. Rahimi *et al.*, “Impala: Algorithm/architecture co-design for in-memory multi-stride pattern matching,” in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2020.

[59] D. S. Cali, G. S. Kalsi *et al.*, “Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis,” in *Int. Symp. on Microarchitecture (MICRO)*, 2020.

[60] S. Angizi, J. Sun *et al.*, “Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram,” in *Design Automation Conf. (DAC)*, 2019.

[61] S. Angizi, N. A. Fahmi *et al.*, “Pim-assembler: A processing-in-memory platform for genome assembly,” in *Design Automation Conf. (DAC)*, 2020.

[62] W. Huangfu, K. T. Malladi *et al.*, “Nest: Dimm based near-data-processing accelerator for k-mer counting,” in *Int. Conf. On Computer Aided Design (ICCAD)*, 2020.

[63] S. Gupta, M. Imani *et al.*, “Rapid: A reram processing in-memory architecture for dna sequence alignment,” in *Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2019.

[64] Q. Zhu, T. Graf *et al.*, “Accelerating sparse matrix-matrix multiplication with 3d-stacked logic-in-memory hardware,” in *High Performance Extreme Computing Conf. (HPEC)*, 2013.

[65] Q. Zhu, B. Akin *et al.*, “A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing,” in *Int. 3D Systems Integration Conf. (3DIC)*, 2013.

[66] R. Nair, S. F. Antao *et al.*, “Active memory cube: A processing-in-memory architecture for exascale systems,” *IBM Journal of Research and Development*, 2015.

[67] M. A. Alves, P. C. Santos *et al.*, “Opportunities and challenges of performing vector operations inside the dram,” in *Int. Symp. on Memory Systems (MEMSYS)*, 2015.

[68] H. Ahmed, P. C. Santos *et al.*, “A compiler for automatic selection of suitable processing-in-memory instructions,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[69] E. C. d. A. Tiago Rodrigo Kepe and M. A. Z. Alves, “Database processing-in-memory: An experimental study,” in *Proc. VLDB Endow.*, 2019.

[70] S. Kim, H. Oh *et al.*, “Fast, energy efficient scan inside flash memory ssds,” in *Int. Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS VLDB)*, 2011.

[71] N. Mirzadeh, Y. O. Koçberber *et al.*, “Sort vs. hash join revisited for near-memory execution,” in *5th Workshop on Architectures and Systems for Big Data (ASBD 2015)*, no. POST_TALK, 2015.

[72] S. L. Xi, A. Augusta *et al.*, “Beyond the wall: Near-data processing for databases,” in *Int. Workshop on Data Management on New Hardware (DaMoN)*, 2015.

[73] V. S. et al., “Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology,” in *Int. Symp. on Microarchitecture (MICRO)*, 2017.

[74] P. C. Santos, G. F. Oliveira *et al.*, “Operand size reconfiguration for big data processing in memory,” in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2017.

[75] Y. Sun, Y. Wang, and H. Yang, “Energy-efficient sql query exploiting rram-based process-in-memory structure,” in *Non-Volatile Memory Systems and Applications Symp. (NVMSA)*, 2017.

[76] S. H. Pugsley, J. Jestes *et al.*, “NDC: analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads,” in *Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2014.

[77] S. H. Pugsley, A. Deb *et al.*, “Fixed-function hardware sorting accelerators for near data mapreduce execution,” in *Int. Conf. on Computer Design (ICCD)*, 2015.

[78] A. Farmahini-Farahani, J. H. Ahn *et al.*, “Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules,” in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[79] M. Drumond, A. Daglis *et al.*, “The mondrian data engine,” in *Int. Symp. on Computer Architecture (ISCA)*, 2017.

[80] M. Drumond, A. Daglis *et al.*, “Algorithm/architecture co-design for near-memory processing,” *Operating Systems Review*, 2018.

[81] M. Alian, S. W. Min *et al.*, “Application-transparent near-memory processing architecture with memory channel network,” in *Int. Symp. on Microarchitecture*

(MICRO), 2018.

[82] F. Devaux, "The true processing in memory accelerator," in *Hot Chips Symp.*, 2019.

[83] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Int. Symp. on Microarchitecture (MICRO)*, 2019.

[84] X. Xin, Y. Zhang, and J. Yang, "Elp2im: Efficient and low power bitwise operation processing in dram," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2020.

[85] N. Hajinazar, G. F. Oliveira *et al.*, "Simdram: a framework for bit-serial simd processing using dram," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[86] S. Jain, A. Ranjan *et al.*, "Computing in memory with spin-transfer torque magnetic ram," *Trans. on Very Large Scale Integration (VLSI) Systems (TVLSI)*, 2018.

[87] L. Xie, H. Cai, and J. Yang, "Real: Logic and arithmetic operations embedded in rram for general-purpose computing," in *Int. Symp. on Nanoscale Architectures (NANOARCH)*, 2019.

[88] M. Gao and C. Kozyrakis, "Hrl: Efficient and flexible reconfigurable logic for near-data processing," in *Int. Smyp. on High Performance Computer Architecture (HPCA)*, 2016.

[89] D. Zhang, N. Jayasena *et al.*, "Top-pim: Throughput-oriented programmable processing in memory," in *Int. Symp. on High-performance Parallel and Distributed Computing (HPDC)*, 2014.

[90] D. Tsirogiannis, S. Harizopoulos *et al.*, "Query processing techniques for solid state drives," in *SIGMOD Int. Conf. on Management of Data*, 2009.

[91] G. Graefe, S. Harizopoulos *et al.*, "Designing database operators for flash-enabled memory hierarchies," *IEEE Data Eng. Bull.*, 2010.

[92] S. Boboila, Y. Kim *et al.*, "Active flash: Out-of-core data analytics on flash storage," in *Symp. on Mass Storage Systems and Technologies*, 2012.

[93] D. Tiwari, S. Boboila *et al.*, "Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines," in *USENIX conference on File and Storage Technologies*, 2013.

[94] J. Do, Y.-S. Kee *et al.*, "Query processing on smart ssds: Opportunities and challenges," in *ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD, 2013.

[95] K. Park, Y. Kee *et al.*, "Query processing on smart ssds," *Data Eng. Bull.*, 2014.

[96] J. Do, I. L. Picoli *et al.*, "Better database cost/performance via batched I/O on programmable SSD," *VLDB Journal*, 2021.

[97] V. Seshadri, Y. Kim *et al.*, "Rowclone: Accelerating data movement and initialization using dram," *arXiv preprint arXiv:1805.03502*, 2018.

[98] M. F. Ali, A. Jaiswal, and K. Roy, "In-memory low-cost bit-serial addition using commodity dram technology," *Trans. on Circuits and Systems (TCS)*, 2020.

[99] B. Jacob, D. Wang, and S. Ng, *Memory systems: cache, DRAM, disk.* Morgan Kaufmann, 2010.

[100] M. Wei, M. Snir *et al.*, "A near-memory processor for vector, streaming and bit manipulation workloads," Dept. of Computer Science, UIUC, Tech. Rep., 2005.

[101] A. Farmahini-Farahani, J. H. Ahn *et al.*, "Drama: An architecture for accelerated processing near memory," *Computer Architecture Letters (CAL)*, 2014.

[102] O. Kocberber, B. Grot *et al.*, "Meet the walkers accelerating index traversals for in-memory databases," in *Int. Symp. on Microarchitecture (MICRO)*, 2013.

[103] M. Hashemi, E. Ebrahimi *et al.*, "Accelerating dependent cache misses with an enhanced memory controller," in *Int. Symp. on Computer Architecture (ISCA)*, 2016.

[104] A. J. Awan, M. Brorsson *et al.*, "Micro-architectural characterization of apache spark on batch and stream processing workloads," in *Int. Conf. on Big Data and Cloud Computing (BDCloud)*, 2016.

[105] T. Perez, N. L. V. Calazans, and C. A. De Rose, "System-level impacts of persistent main memory using a search engine," *Microelectronics Journal*, 2014.

[106] A. J. Awan, M. Ohara *et al.*, "Identifying the potential of near data processing for apache spark," in *Int. Symp. on Memory Systems (MEMSYS)*, 2017.

[107] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *Symp. on VLSI Technology (VLSIT)*, 2012.

[108] J. Pawlowski, "Hybrid memory cube (hmc)," *Hot Chips*, 2011.

[109] Hybrid Memory Cube Consortium, "Hybrid memory cube specification rev. 2.0," 2013, http://www.hybridmemorycube.org/.

[110] A. Boroumand, S. Ghose *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[111] G. H. Loh, N. Jayasena *et al.*, "A processing in memory taxonomy and a case for studying fixed-function pim," in *Workshop on Near-Data Processing (WoNDP)*, 2013.

[112] J. a. P. Lima, P. C. Santos *et al.*, "Design space exploration for pim architectures in 3d-stacked memories," in *Computing Frontiers (CF)*, 2018.

[113] P. C. Santos, M. A. Alves *et al.*, "Exploring cache size and core count tradeoffs in systems with reduced memory access latency," in *Int. Conf. Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 2016.

[114] A. Shahab, M. Zhu *et al.*, "Farewell my shared llc! a case for private die-stacked dram caches for servers," *Int. Symp. on Microarchitecture (MICRO)*, 2018.

[115] P. C. Santos, J. P. Lima *et al.*, "Solving datapath issues on near-data accelerators," in *Int. Embedded Systems Symp. (IESS)*, ser. IESS '19, September 2019.

[116] P. C. Santos, J. P. C. de Lima *et al.*, "A technolog-

ically agnostic framework for cyber-physical and iot processing-in-memory-based systems simulation," *Microprocessors and Microsystems (MICPRO)*, 2019.

[117] P. C. Santos, B. E. Forlin, and L. Carro, "Providing plug n'play for processing-in-memory accelerators," in *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2021.

[118] D. Lee, Y. Kim *et al.*, "Tiered-latency dram: A low latency and low cost dram architecture," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2013.

[119] K. Kara, D. Alistarh *et al.*, "Fpga-accelerated dense linear machine learning: A precision-convergence trade-off," in *Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2017.

[120] G. Singh, M. Alser *et al.*, "Fpga-based near-memory acceleration of modern data-intensive applications," *IEEE Micro*, 2021.

[121] A. S. Cordeiro, T. R. Kepe *et al.*, "Intrinsics-hmc: an automatic trace generator for simulations of processing-in-memory instructions," *Symp. Sistemas Computacionais de Alto Desempenho (WSCAD)*, 2017.

[122] C. Lomont, "Introduction to intel advanced vector extensions," *Intel White Paper*, 2011.

[123] R. Hadidi, L. Nai *et al.*, "Cairo: A compiler-assisted technique for enabling instruction-level offloading of processing-in-memory," *Trans. on Architecture and Code Optimization (TACO)*, 2017.

[124] K. Hsieh, E. Ebrahimi *et al.*, "Transparent offloading and mapping (tom) enabling programmer-transparent near-data processing in gpu systems," *SIGARCH Computer Architecture News*, 2016.

[125] D. Khaldi and B. Chapman, "Towards automatic hbm allocation using llvm: a case study with knights landing," in *Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, 2016.

[126] R. Balasubramonian, J. Chang *et al.*, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, 2014.

[127] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *Trans. on Very Large Scale Integration (VLSI) Systems (TVLSI)*, 2018.

[128] A. Boroumand, S. Ghose *et al.*, "Conda: Efficient cache coherence support for near-data accelerators," in *Int. Symp. on Computer Architecture (ISCA)*, 2019.

[129] C. Sudarshan, J. Lappas *et al.*, "An in-dram neural network processing engine," in *Int. Symp. on Circuits and Systems (ISCAS)*, 2019.

[130] G. Singh, L. Chelini *et al.*, "A review of near-memory computing architectures: Opportunities and challenges," in *Conf. on Digital System Design (DSD)*, 2018.

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

[132] H. Du Nguyen *et al.*, "A classification of memory-centric computing," *Journal on Emerging Technologies in Computing Systems (JETCS)*, 2020.