Near-memory Computing on FPGAs with 3D-stacked Memories: Applications, Architectures, and Optimizations

VERONIA ISKANDAR, Technische Universität Dresden, Chair of Adaptive Dynamic Systems, Germany
MOHAMED A. ABD EL GHANY, Electronics Department/IES Lab, German University in Cairo/TU Darmstadt, Egypt/Germany
DIANA GÖHRINGER, Technische Universität Dresden, Chair of Adaptive Dynamic Systems, Germany

The near-memory computing (NMC) paradigm has transpired as a promising method for overcoming the memory wall challenges of future computing architectures. Modern systems integrating 3D-stacked DRAM memory can be leveraged to prevent unnecessary data movement between the main memory and the CPU. FPGA vendors have started introducing 3D memories to their products in an effort to remain competitive on bandwidth requirements of modern memory-intensive applications. Recent NMC proposals target various types of data processing workloads such as graph processing, MapReduce, sorting, machine learning, and database analytics.

In this article, we conduct a literature survey on previous proposals of NMC systems on FPGAs integrated with 3D memories. By leveraging the high bandwidth offered from such memories together with specifically designed hardware, FPGA architectures have become a competitor to GPU solutions in terms of speed and energy efficiency. Various FPGA-based NMC designs have been proposed with software and hardware optimization methods to achieve high performance and energy efficiency. Our review investigates various aspects of NMC designs such as platforms, architectures, workloads, and tools. We identify the key challenges and open issues with future research directions.

CCS Concepts: • **Hardware** → **Reconfigurable logic and FPGAs**; • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → *Parallel architectures;*

Additional Key Words and Phrases: Near-memory computing, 3D stacking, FPGA architectures, high-bandwidth memory

ACM Reference format:
Veronia Iskandar, Mohamed A. Abd El Ghany, and Diana Göhringer. 2022. Near-memory Computing on FPGAs with 3D-stacked Memories: Applications, Architectures, and Optimizations. *ACM Trans. Reconfigurable Technol. Syst.* 16, 1, Article 16 (December 2022), 32 pages.
https://doi.org/10.1145/3547658

The research work presented in this article was funded by the German Research Foundation ("Deutsche Forschungsgemeinschaft") (DFG) under Project ID No. 287022738 TRR 196 for Project S05.
Authors' addresses: V. Iskandar and D. GÖhringer, Technische Universität Dresden, Chair of Adaptive Dynamic Systems, Nöthnitzer Str. 46, 01062 Dresden, Germany; emails: veronia.iskandar@mailbox.tu-dresden.de, diana.goehringer@tu-dresden.de; M. A. Abd El Ghany, Electronics Department, German University in Cairo, Cairo 11835, Egypt and Technische Universität Darmstadt, Merckstr. 25, 64283 Darmstadt, Germany; email: mohamed.abdel-ghany@guc.edu.eg.

1936-7406/2022/12-ART16 $15.00
https://doi.org/10.1145/3547658

1 INTRODUCTION

Modern computing systems exhibit rapidly increasing computational capacity, mainly due to the improvements of CMOS technology. Such improvements are often enabled by either integrating more processing elements within the same area and/or by enhancing the processing cores with extra features (e.g., SGX, AVX, GPGPU, etc.). However, DRAM memory bandwidth has seen only small development over many decades. Therefore, the gap between memory and computation speeds keeps increasing, especially in multicore designs that require concurrent memory access. Additionally, memory technology has not been able to keep up with processor speeds in terms of latency and energy consumption, which is known as the memory wall [101]. At the same time, huge amounts of data are generated in modern workloads. Many applications have high data parallelism and low computational intensity. On conventional systems, these applications require frequent data movement between the memory and the processor. This has a severe impact on performance and energy efficiency.

Many research works aim nowadays to find innovative manufacturing technologies and architectures to overcome the memory wall. The conventional memory hierarchy consists of several levels of cache, the main memory, and disk storage. During computation, the data needed by the processor is moved up to caches from the storage and then processed. However, **near-memory computing (NMC)** changes this paradigm by moving the processing near to where the data resides. This data-centric approach introduces processing units close to the data to reduce the expensive data movements. Prior architectures were based on a processor-centric design, where data is transferred to the CPU for processing, while with near-memory processing the cores are brought to the place where data is.

Innovative memory technologies using three-dimensional stacking are true enablers of processing close to the memory. Three-dimensional stacking allows the integration of logic and memory together on one chip. Several layers are packed vertically using **through-silicon vias (TSVs)**, which helps in decreasing memory access latency, energy consumption and provides significantly higher bandwidth [39]. Micron's **Hybrid Memory Cube (HMC)** [74], **High-bandwidth Memory (HBM)** [55] from AMD and Hynix, and Samsung's Wide I/O [51] are the most prominent 3D memory products. An example for using such technologies in the GPU community is the Nvidia GPU V100, which includes a 32 GB HBM2, allowing up to 900 GB/s memory bandwidth to its processor cores.[1]

Nowadays, architects can realise complete systems that are based on the NMC paradigm in hardware using FPGAs that feature 3D memory variants. In such FPGAs, the 3D memory is situated close to the FPGA programmable logic, thus realising the NMC paradigm. Previously, FPGAs used to have significantly lower memory bandwidth compared to GPUs of the same generation. Most FPGAs featured at most two DRAM channels, each providing up to approximately 19 GB/s memory bandwidth [33]. Consequently, an FPGA-based architecture using DRAM memory could not contend with a GPU for bandwidth-critical workloads. Due to this limitation, FPGA vendors like Xilinx and Intel started to add HBM and HMC memories in their FPGA boards in an effort to remain competitive on bandwidth requirements. Integrating high-bandwidth memories promises to be a game-changing feature by enabling FPGAs to perform significantly better for memory-bound as well as compute-bound workloads such as database [98] and deep learning [62] applications.

The article analyzes and organizes the extensive body of literature related to the novel area of near-memory computing on FPGAs. We make the following contributions:

- We analyze and organize recent literature on near-memory computing on FPGAs under various dimensions.

[1]https://www.nvidia.com/en-us/data-center/v100/.

- We highlight current challenges in FPGA-HBM designs.
- We outline the directions for future research.

2 BACKGROUND AND RELATED WORK

2.1 Near-memory Computing

The concept of processing near to the memory is as old as the 1960s [94]. However, the first NMC systems can be traced back to the early 1990s [13, 15, 23, 45, 52, 73]. One such system was **Vector IRAM (VIRAM)** [53], where a vector processor with an on-chip embedded DRAM was proposed to make use of data parallelism in multimedia workloads. Although research showed promising results, NMC architectures saw limited adoption because of technological shortcomings at that time. For instance, the difficulty of integrating memory with logic due to the difference in technology processes.

Nevertheless, numerous studies demonstrated the potential of near-storage computing facilities for FPGAs. IBEX [100] is an early prototype for offloading database tasks to near-storage processing elements, where an FPGA resides between the host and a SATA SSD. Biscuit [25] presents a general NMC platform. JAFAR [7] similarly demonstrates NMC for DBMS. Caribou [38] and BlueDBM [44] propose RDMA-based KV-Stores exploiting node-local operations, but their frameworks are not capable of dynamic task loading and possess limited OS-level support. Most previous works on persistent KV-Stores and NMC aim for bandwidth optimization. An FPGA-based near-storage computing system was proposed by Ruan et al. [83]. Virtual files map to the contents of conventional files and integrate them with near-storage operations. All are similar in that the near-storage accelerators are managed by applications or custom run-time systems.

Nowadays, research in NMC is resurging, mainly due to the following three reasons. First, manufacturing technology advancements have enabled 3D and 2.5D stacking, which allows integration of logic and memory in the same package. Second, bringing the computation nearer to the data allows for mitigating the performance and energy bottlenecks caused by data movement by avoiding memory-package pin-count limitations. Third, modern data-intensive applications in areas like biology, astronomy, and materials studies call for newer data-centric architectures. Multiple NMC designs [3, 16, 31, 35, 85] have been proposed and their potential to accelerate many applications has been demonstrated. NMC is also referred to as **near-memory processing (NMP)**, and **near-data processing (NDP)**.

Loh et al. [35] presented an initial classification for processing near memory based on computing interface with software, and separately investigated transparent software features. Siegl et al. [90] study the historical evolution of NMC. However, their classification is not systematic and they do not discuss design challenges and future research directions in this field. Ghose et al. [21] provide an overview of the mechanisms and challenges in the domain of near-memory computing. However, they do not provide a systematic literature review. Singh et al. [91] classified NMC proposals by the memory level where near-memory processing is applied and the type of processing units. Although the work of Singh et al. [91] is fairly recent, NMC research on FPGAs is not detailed in it, as such research has only increased in recent years, and is expected to continue to grow. Holzinger et al. [30] investigate the reliability of measurements on a Xilinx HBM FPGA and propose several architectural improvements to increase throughput. A roofline model is developed to accurately predict accelerator performance.

2.2 Processing in Memory

Although our focus is NMC systems on FPGAs, we point out another promising direction of designing architectures that bring data close to processing elements, which is in-memory processing.

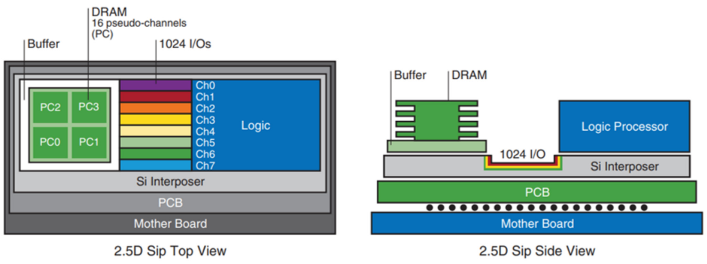

Fig. 1. High-bandwidth memory (HBM). Figure from www.xilinx.com.

Processing-in-memory refers to computing inside memory arrays to reduce data movement. Conventional DRAM and SRAM memories have been used for the frequently-occurring computations in data-intensive applications [59]. However, conventional memory technologies do not supply enough computing and storage required by data-intensive workloads. In addition, their large cell size and high leakage power lead to large area and power consumption. **Non-volatile memories (eNVMs)**, which supply better scaling and near-zero leakage power, have emerged as promising candidates for processing-in-memory. eNVM technologies include **spin-transfer torque RAM (STT-RAM)**, **phase-change memory (PCM)**, and **resistive RAM (ReRAM)**. Among such technologies, STT-RAM provides the fastest access and the lowest energy consumption while the cell area is relatively larger [71]. Both PCMs [56] and ReRAMs [58] can hold several bits in a single memory cell, demonstrating better density with technology scaling. These architectures support inherently parallel workloads. Multiple studies have proposed building in-memory processing using eNVM. Li et al. [57] survey the recent proposals in in-memory processing utilizing eNVM technologies (STT-RAM, PCM, and ReRAM).

2.3 3D-stacked Memories

This section focuses on the basic specifications as well as the chip realization of the two most prominent 3D-stacked memory technologies, HBM and HMC. Both memories make use of 3D-stacking and extended in-memory parallelism. The usage of 3D-stacking increases the density of the memory and therefore reduces the number of memory packages needed. The dies incorporated in a memory package are connected using **Through-Silicon Vias (TSVs)** [43, 74].

High-bandwidth Memory (HBM). Bandwidth is an important characteristic of memory. A high memory bandwidth is required for memory-intensive workloads to utilize the computational power of modern compute elements. HBM increases the memory bandwidth compared to traditional DDR memory, while using much lower clock frequencies. An increase in bandwidth and simultaneously a decrease in clock speed are achieved by connecting four to eight memory channels to the same memory module. Due to the large number of channels connected to each module, a single module can serve multiple requests simultaneously. At the same time, each memory channel has an increased width of 128 bit, compared to the 64 bit of DDR memory.

An HBM module consists of multiple DRAM dies and a logic die, which are all connected via TSVs, as illustrated in Figure 1. Such a stack of HBM can contain up to 16 GiB of memory in products adhering to the second version of the specification, with up to eight DRAM dies stacked.[2] The original specification of HBM specified 16 banks per rank [43]. HBM2 increases this to up to 32 banks. Each DRAM die in an HBM2 package provides two memory channels.[3]

[2]https://www.micron.com/-/media/client/global/documents/products/data-sheet/hmc/gen2/hmc_gen2.pdf.
[3]Micron Technology, Inc. High-bandwidth Memory with ECC. URL: https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/hbm2e/8gb_and_16gb_hbm2e_dram.pdf.

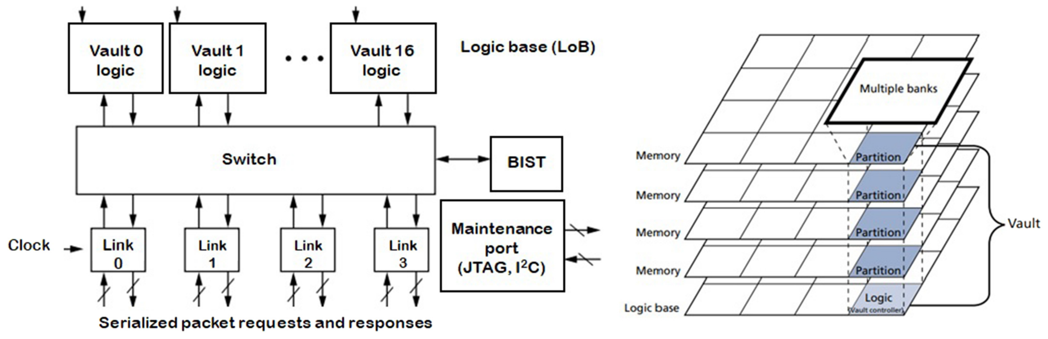

Fig. 2. Hybrid memory cube (HMC). Figure adapted from www.micron.com.

Table 1. Properties of HBM and HMC Memories

Property	HBM	HMC
Stacked Layers	4–8	4–8
Channels	8–16 (HBM 2)	1 (controller in logic die)
Banks	16–32 (HBM 2)	256 (Gen1), 128 (Gen 2)
Vaults	—	32 (Gen 1), 16 (Gen 2)
Capacity	8–16 GB	2–4 GB
Bandwidth	up to 410 GB/s	160 GB/s (10 GB/s per vault)

Hybrid Memory Cube (HMC). Like HBM, HMC is a 3D memory whose DRAM layers and logic die are connected using TSVs. In contrast to other memory technologies, HMC removes the memory controller from the host processor by integrating it into the logic die of the memory module. The structure of the HMC is shown in Figure 2. However, the most important difference between HMC and HBM (and consequently traditional DRAM memory types) is the internal organization of the memory. The memory inside the memory stack is grouped vertically. Across all layers of the stack, memory is organized into vertical vaults, which operate independently. The logic die is partitioned according to the vault structure into small memory controllers, called vault controllers. As a result of this internal organization, many more banks can be utilized concurrently; at the same time, the size of a row is drastically reduced [22]. Since HMC moves the memory controller from the host processor onto the logic die, the processor is not connected to the memory via traditional memory channels, but so-called links. These links implement a packet-based protocol to transmit commands and data. Each packet is composed of 128 bit **Flow Control Digits (FLITs)**.

HBM as well as HMC attempt to increase application performance by increasing some form of parallelism. While HBM increases throughput by widening the interface, i.e., the available channels, HMC increases bank-level parallelism and therefore the number of simultaneous requests. However, both approaches involve trade offs. The key properties of both memories are summarized in Table 1. HBM2 in current products achieves much higher memory density and capacity, and significantly higher peak bandwidth, than HMC. However, the vault organization of HMC provides up to 16 times the number of banks. Interestingly, Micron decreased the number of vaults and therefore the number of banks in the second generation of HMC. This may well be an attempt to improve the performance of the memory in applications, which are less able to exploit the high bank-level parallelism provided by the memory. HBM2, however, increases the number of banks compared to the first generation. According to Ghose et al. [22] HBM and HMC differ in row hit/miss and row conflict latencies with HMC achieving lower latencies in all cases.

Competing high-bandwidth memory technologies like GDDR5 increase clock speeds and the number of connected memory modules to increase bandwidth. However, this comes at a cost of power consumption, which increases with clock speed and voltage. While clock speed alone has a linear relationship with power consumption, voltage increases it quadratically. Therefore, power efficiency worsens with increasing clock speed if no other factors like node shrinks cancel the increased power consumption out. Therefore, HBM has a much lower clock speed than competing technologies. In contrast, HMC has more than double the operating frequency of HBM [22]. In addition to the higher clock speed, HMC needs to implement the link protocol, which potentially consumes significant extra power. There are no recent studies comparing the power consumption and efficiency for different workloads. However, according to Ghose et al., HMC can outperform competing memory technologies significantly in specific workloads. Power efficiency is application-specific and there may well be workloads in which one or the other memory technology or even traditional DDR has better power efficiency.

2.4 FPGA-HBM Platforms

Intel and Xilinx have recently launched HBM2 FPGA boards: Xilinx's Alveo U50 [105], U280 [104], Virtex Ultrascale+ [103], and Intel's Stratix 10 MX [37]. These boards integrate an FPGA and two HBM2 stacks (8 HBM dies).

Xilinx integrates two HBM stacks and an HBM controller inside the FPGA. Each HBM stack has eight independent memory channels, and each memory channel is further divided into two 64-bit **pseudo-channels (PCs)**. Each PC has 256 MB of capacity (8 GB in total). 32 AXI channels provide interaction with the user logic. Each AXI channel implements the AXI3 protocol to ensure a standardized interface to the FPGA programmer. Each AXI channel connects to an HBM pseudo-channel, so each AXI channel is restricted to accessing its own memory region. To enable global access, an optional switch between the 32 AXI channels and the 32 pseudo-channels is implemented. It has been proved difficult to fully implement the switch, because of its large resource requirement. Instead, Xilinx's eight mini-switches can be used. Each mini-switch is connected to 4 AXI channels. Each AXI channel can communicate with any pseudo-channel connected to the same mini-switch with the same latency and throughput. Moreover, there are two bidirectional links between two adjacent mini switches to allow global addressing. The **thermal design power (TDP)** of Alveo U280 is 200W. The Alveo U280 also includes traditional DDR DRAM [9].

In Stratix 10 MX, each PC is connected to the FPGA PHY layer through 64 data I/Os that run at 800 MHz (double data rate). The data communication between the user logic and the HBM2 memory is managed by the HBM controller, which runs at 400 MHz. AXI4 [6] and Avalon [36] interfaces (256 bitwidth) are used to communicate with the user logic. The clock frequency of kernels can reach at most 450 MHz. Since the frequency of the HBM memory controller cannot exceed 400 MHz, rate matching FIFOs are implemented between the user logic and the memory controllers.

The Xilinx Alveo U280 FPGA is divided into three **super logic regions (SLRs)**. The FPGA connects to the HBM2 stacks on SLR0. The data I/Os to the HBM run at 900 MHz. Data transmission is handled by the HBM memory controllers. A memory controller connects with the user logic by an AXI3 interface (256b) operating at 450 MHz. An 512b AXI3 master interface is available to the user logic. The clock frequency of the user logic is limited at 300 MHz. Therefore, we can calculate the ideal memory bandwidth as 460 GB/s (= 256b * 32PCs * 450 MHz).

Data can be sent between the four user logic AXI masters and any of the four adjacent PC AXI slaves using a fully-connected switch. For instance, the first AXI master directly connects to PCs 0–3 (Figure 3). In case an AXI master requires to read/write in non-adjacent PCs, it crosses lateral

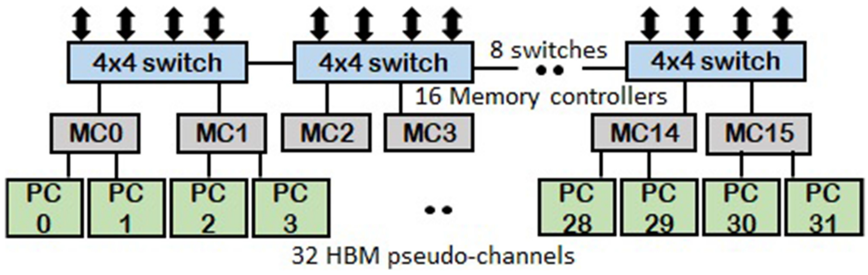

Fig. 3. Xilinx alveo U280 HBM subsystem. Figure adapted from www.xilinx.com.

connections among the unit switches. However, network contention could constrict the resultant bandwidth [106].

2.5 FPGA-HMC Platforms

One of the most used boards in literature is the AC-510 evaluation board by Micron/ PicoComputing. The AC-510 [78] features a 4 GB Micron Hybrid Memory Cube connected to a Xilinx Kintex Ultrascale XCKU060- FFVA1156 FPGA, which forms the Pico Computing AC-510 daughter card. The FPGA is connected to the HMC via two quarter-width HMC links, producing a single half-width link. The FPGA has access only to a quarter of the maximum HMC throughput, which supports two full-width links. The Micron/Pico Computing AC-510 daughter card is integrated on a Pico EX-700, which can support a maximum of six cards. PCIe interconnect between the FPGAs as well PCIe interconnect to the host are provided.

The Micron SB-800 board was produced by Intel, featuring 4 Altera Stratix V FPGAs, and a 4 GB HMC with a single high speed serial link to each FPGA. To achieve the best performance, each link can only access memory inside its own four HMC vaults. The AC-520 platform features an Intel Arria 10 GX 1150 FPGA and a 4 GB HMC module. The HMC interfaces with FPGA with four half-width links, which provide a two-way bandwidth of 60 GB/s. The AC-520 additionally exposes the interface for power consumption measurement.

3 A MOTIVATION FOR NEAR-MEMORY COMPUTING ON FPGAS

Modern FPGA boards are ideal for releasing the NMC paradigm to accelerate data-intensive workloads. This is because such FPGAs can deal with irregular memory access patterns efficiently and can achieve considerably higher memory bandwidth than the CPU due to their on-chip **URAMs (UltraRAM)**, **BRAMs (block RAM)**, and off-chip high-bandwidth memory. Thanks to their architectural flexibility, FPGAs are popularly exploited for various data processing tasks including machine and deep learning [17, 46, 97], database processing [8, 47], and networking [84].

Additionally, recent FPGAs can deploy new cache-coherent interconnects, such as IBM CAPI [95], **Cache Coherent Interconnect for Accelerators (CCIX)** [95], and **Compute Express Link (CXL)** [89], which allow tight integration of FPGAs with CPUs at high bidirectional bandwidth (in the order of tens of GB/s). Such interconnects allow acceleration kernels to be efficiently offloaded from a host CPU to the FPGA to be processed near the memory. Figure 4 shows a typical architecture of an FPGA-based NMC system.

Specialized hardware architectures are being proposed by major cloud providers to provide an alternative to general-purpose processors to achieve high performance and energy efficiency. Examples of such approaches are the manufacturing of **Tensor Processing Unit (TPU)** by Google to accelerate deep learning [42], the usage of FPGAs by Microsoft in their datacenters to offload

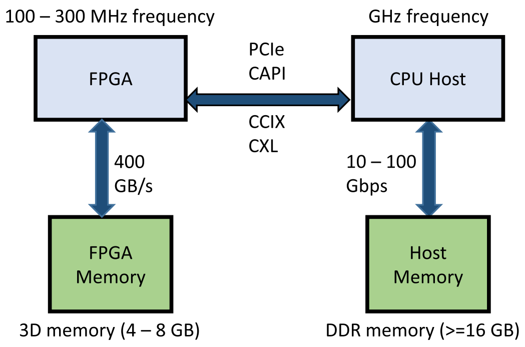

Fig. 4. A typical structure of an FPGA-based NMC system.

computation to the network [80] and perform low latency inference [10], the design of efficient interconnects such as OpenCAPI [95], which simplify the integration of accelerators into host systems, and FPGAs being incorporated in the AWS cloud [1] and Baidu [2].

However, leveraging the full potential of FPGAs for accelerating an application is not a trivial task. FPGAs must achieve at least one order of magnitude more parallelism in a given workload. This requires considerable FPGA programming knowledge to map the application kernel and optimize the design for the FPGA microarchitecture.

4 CLASSIFICATION AND EVALUATION

This section introduces the classification and evaluation metrics that are used in the next sections and are summarized in Table 2. For each architecture, the following main categories are evaluated and classified:

(1) **FPGA platform.** The decision of using which type of FPGA and 3D memory is one of the most primary questions on which the near-memory architecture depends, as HBM and HMC vary greatly in their implementation, and vendors also vary in provided interfaces and tools.
(2) **Workload.** NMC architectures are data-centric. A given architecture is usually specialized for a particular application. Hence, in our classification, we include the domain of the application.
(3) **Implementation language.** Using low-level hardware description languages like Verilog requires good knowledge of hardware design details and hands-on experience. However, higher-level languages such as HLS can reduce programming efforts needed by FPGA programmers. An example for this is the Vitis framework [107] provided by Xilinx. Even though HLS increases programmability, high performance cannot be simply obtained without careful design of the hardware pipeline and the memory subsystem.
(4) **Processing.** The implemented processing units on the FPGA and the granularity of their processing are investigated.
(5) **Optimization target.** The goal of implementing the NMC architecture could be to improve throughput, reduce latency, or reduce power consumption of the workload.
(6) **Evaluation comparison.** We investigate the baseline architectures, such as CPUs, GPUs or other FPGAs to which the NMC architectures are compared.
(7) **Open source.** The research community benefits most by open-source proposals.

We first show a classification of NMC architectures based on their purpose, 3D memory integrated on the FPGA platform, and their implementation language (Figure 5). In Figure 5, the classification starts by classifying the architectures by looking into their target and their processing

Table 2. Classification Metrics, Legend for Table 3

Property	Abbreviation	Description
FPGA Platform	ac510	Xilinx AC-510
	ac520	Altera AC-520
	SC-6	Pico SC-6 Mini
	V6	Xilinx Virtex6
	VU37P	Xilinx VU37P
	SB	Altera SB-800
	MX	Intel Stratix 10 MX 2100
	ADM	Alpha Data ADM-PCIE-9H7
	VU+	Xilinx Virtex Ultrascale+
	U280	Xilinx Alveo U280
	Emul	Emulation
	Indp	Platform-independent
3D Memory	HMC	Hybrid Memory Cube
	HBM	High-bandwidth Memory
Workload	—	Target application for acceleration
	C	Characterization of the 3D memory
Implementation Language	V	Verilog
	SV	SystemVerilog
	Ch	Chisel
	CL	OpenCL
	HDL	Unspecified hardware description language
	HLS	High-Level Synthesis
NMC processing elements	Acc	App specific accelerator
	CPU	Central Processing Unit
	MC	Memory controller
Optimization Target	P	Power
	T	Performance (throughput or execution time)
	C	Characterization of the 3D memory
Comparison	DDR	DDR-attached FPGA
	CPU	CPU-based software implementation
	GPU	Graphics Processing Unit
	CGRA	Coarse-Grained Reconfigurable Array
Open Source	—	Availability of the working code

elements, to highlight the main goals of designing FPGA-based NMC architectures. The three main categories are application-specific accelerators, generic architectures, and performance characterization architectures. Application-specific accelerators implement processing elements targeting the acceleration of a specific application. Generic architectures implement general-purpose cores that can run any type of workload. Performance characterization architectures target stressing the memory system for extraction of its characteristics. Performance characterization architectures also require the implementation of simple processing elements to interact with the memory. However, the focus is not accelerating a specific application, but to investigate the memory properties.

Afterwards, we detail the properties of each architecture in Table 3. From Table 3, we observe that the majority of proposals target homogeneous application-specific accelerators near the

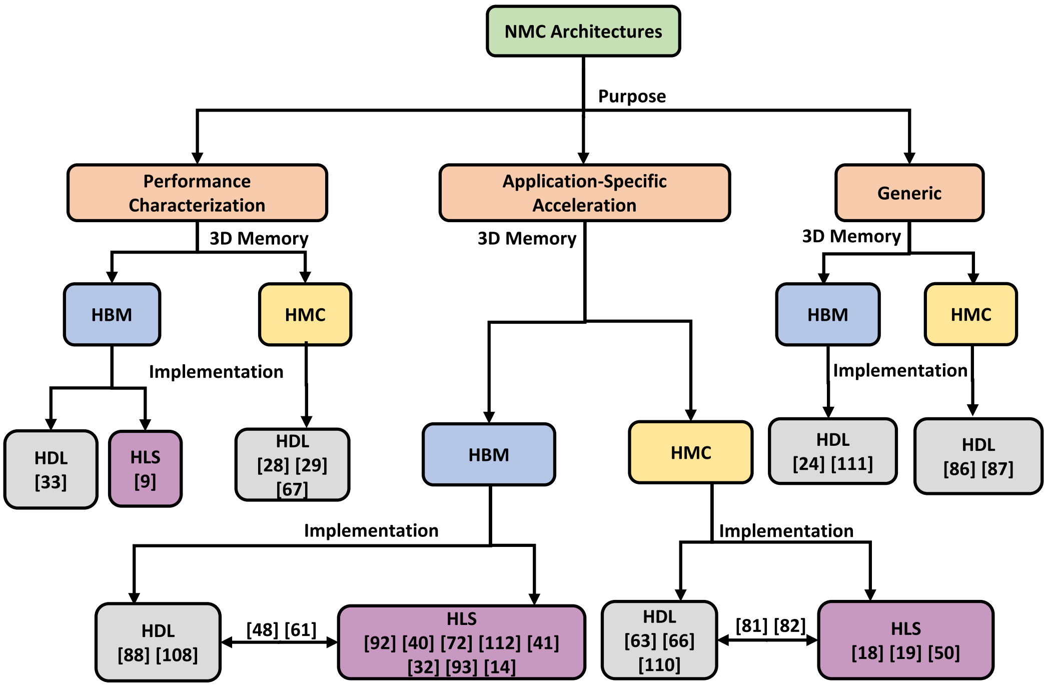

Fig. 5. Classification of NMC architectures based on purpose, 3D memory used, and implementation language.

memory. Most NMC proposals target various types of data processing workloads such as graph processing, MapReduce, sorting, machine learning, and database analytics. Additionally, we notice that prior proposals of NMC architectures on FPGA have started with HMC-based FPGAs, but nowadays the majority of papers published in this field target the HBM. This is mainly due to the shift of FPGA vendor support towards the HBM, and also the similarity of the memory protocols of HBM to traditional DDR memory. For this reason, we focus on FPGA-HBM proposals when we discuss design challenges and open research directions.

5 APPLICATION-SPECIFIC ACCELERATION ARCHITECTURES

Most NMC application-specific accelerators target performance improvement of certain applications. Few of such works [72, 88] also investigate power consumption as an important factor besides throughput maximization. Application-specific NMC architectures extensively use both the HBM [40, 41, 72, 88, 112] and the HMC [18, 19, 50, 63, 66, 70, 82, 110]. A typical HBM-based FPGA-NMC accelerator system is shown in Figure 6.

Sparse matrix operations are one of the most popular applications for FPGA-NMC acceleration. Sgherzi et al. [88], Parravicini et al. [72], Sextans [93], HiSparse [14], and Graphlily [32] present HBM-based NMC architectures to improve the throughput of sparse matrix operations using the Xilinx Alveo U280. Sgherzi et al. [88] propose an approximate a solution to the Top-K eigenproblem on sparse matrices, which is memory-intensive. Additionally, a Systolic Array design is implemented for the computationally-intensive Jacobi eigenvalue algorithm. Similarly, Parravicini et al. [72] propose a Top-K SpMV FPGA design that leverages reduced precision. Top-K SpMV is highly memory intensive and includes sequential and random memory accesses, rendering it unsuitable for traditional caching. Sextans [93] is an accelerator for general-purpose SpMM processing.

Table 3. Architectures Classification and Evaluation

NMC Architecture	Year	FPGA	3D Mem.	Workload	Impl.	PEs	Target	Evaluation	Open Source
openHMC [86]	2015	Indp	HMC	–	V	MC	–	–	yes
Lloyd et al. [63]	2015	Emul	HMC	Analytics	HDL	Acc	T	CPU	no
Schmidt et al. [87]	2016	Emul	HMC	HPC	HDL	Acc	C	–	no
HRL [18]	2016	V6	HMC	Analytics	HLS	Acc	P, T	DDR, CGRA	no
Garg et al. [19]	2017	ac510	HMC	SuperLU [60]	CL	Acc	T	–	no
Hadidi et al. [28]	2017	ac510	HMC	GUPS	V	Acc	C	–	no
Rozhko et al. [82]	2017	ac510	HMC	Packet matching	HLS, SV	Acc	T	–	no
Mcvicar et al. [66]	2017	SB	HMC	K-mer Counting	V	Acc	T	CPU, DDR	no
CoolPIM [67]	2018	ac510	HMC	Graph processing	V	Acc	C	–	no
Khan et al. [50]	2018	ac510	HMC	Packet processing	HLS	Acc	T	–	no
Hadidi et al. [29]	2018	SC6	HMC	GUPS, stream	V	Acc	C	–	no
Pantho et al. [70]	2018	ac510	HMC	Image processing	–	Acc	T	CPU, DDR	no
Zhang et al. [110]	2018	ac510, ac520	HMC	Graph processing	HDL	Acc	T	CPU, DDR, GPU	no
Volcan [81]	2019	ac510	HMC	HPC	V, HLS	Acc	T	CPU, DDR	no
2GRVI Phalanx [24]	2019	VU37P	HBM	Generic	SV	CPU	–	–	no
Kara et al. [48]	2020	ADM	HBM	Analytics	HLS, SV	Acc	T	CPU	no
MEG [111]	2020	VU+	HBM	Generic	Ch	CPU	EM	–	yes
Nero[92]	2020	ADM	HBM	Weather prediction	HLS	Acc	T, P	DDR	no
Yang et al. [108]	2020	MX	HBM	Hash table	HDL	Acc	T	GPU, DDR	no
MICROREC [40]	2021	U280	HBM	Inference	HLS	Acc	T	CPU	yes
Scalabfs [61]	2021	U280	HBM	BFS	CL, Ch	Acc	T	DDR, GPU	yes
Sgherzi et al. [88]	2021	U280	HBM	Top-K eigenproblem	HDL	Acc	T,P	CPU	no
Parravicini et al. [72]	2021	U280	HBM	Top-K SpMV	HLS	Acc	T, P	CPU	yes
Zhu et al. [112]	2021	U280	HBM	Inference	HLS	Acc	T	CPU	no
FleetRec [41]	2021	U280	HBM	Inference	HLS	Acc	T	CPU, DDR	no
HBM Connect [9]	2021	U280	HBM	Sorting	HLS	Acc	C	–	yes
Shuhai [33]	2021	U280	HBM	RST	V	Acc	C	–	yes
Graphlily [32]	2021	U280	HBM	Graph processing	HLS	Acc	T	GPU, CPU, DDR	yes
Sextans [93]	2022	U280	HBM	SpMM	HLS	Acc	T	GPU, CPU, DDR	yes
HiSparse [14]	2022	U280	HBM	SpMV	HLS	Acc	T	GPU, CPU, DDR	yes

Refer to Table 2 for a legend.

HiSparse [14] is a high-performance SpMV accelerator optimized for maximizing performance on multi-die HBM-equipped FPGA device. GraphLily [32] is an overlay to accelerate graph processing using the GraphBLAS interface, which constructs graph algorithms as sparse linear algebra operations. HMC-based acceleration of sparse matrix operations is also studied by Garg et al. [19] on the AC-510 platform for investigating potential performance improvements for the sparse direct solver library, SuperLU. The work focuses on migration of kernels to OpenCL and concludes that more careful optimization for SuperLU are needed, as well as more hardware support for bursty HMC read/write accesses.

MicroRec [40], Zhu et al. [112], and FleetRec [41] use an HBM to accelerate recommendation inference systems using Xilinx FPGAs. MicroRec [40] utilizes both hardware and data structure

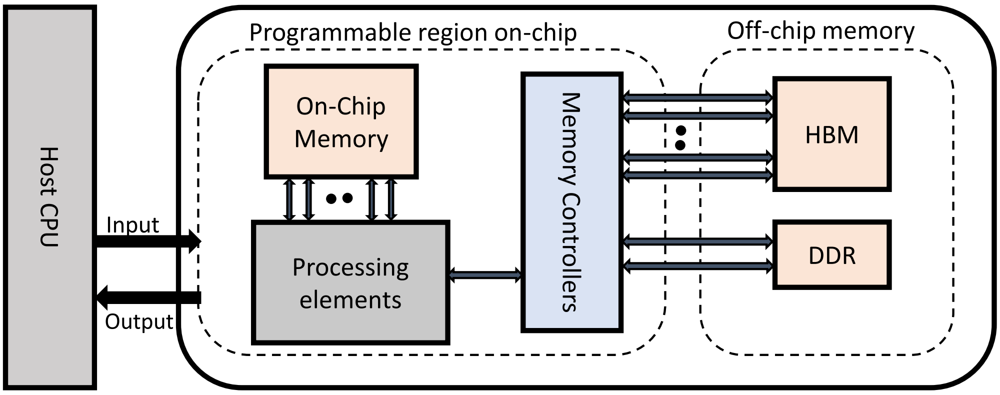

Fig. 6. Block graph of a typical HBM-based accelerator.

mechanisms to mitigate the memory bottleneck due to embedding lookups. However, the computation highly constrains the overall performance on a single FPGA after the memory bottleneck is avoided. As an improvement over the previous work, Zhu et al. [112] leverage an FPGA cluster for recommendation inference operations to gain high throughput on the embedding layer computation while keeping a low inference latency. For reliable data transmission, an open-source 100 Gbps TCP/IP stack customized for Vitis HLS is leveraged. FleetRec [41] additionally introduces using a configurable heterogeneous computing cluster for recommendation inference. On the embedding table lookup side, FPGAs equipped with HBMs are employed to perform parallel lookup operations, and CPUs with sufficient DRAM memory storage are used for large tables. On the computation side, GPUs perform DNN computations to avoid the irregular memory accesses that decrease the GPU performance. The GPUs, FPGAs, and CPU servers are connected through a high-speed network.

Improving the performance of packet processing applications using HMC memory is the focus of the work of Rozhko et al. [82] and Khan et al. [50]. Rozhko et al. [82] implement a packet processing architecture that uses an HMC memory to stream rules to a packet matching engine. The processing elements of the matching engine are implemented as a systolic array to reduce the fanout of the Rule Stream data signal and avoid large multiplexers and demultiplexers. A Micron AC-510 FPGA accelerator card and the Pico Framework are used. The Pico Framework is a wrapper for the FPGA to include the PCIe and HMC links in hardware designs, as well as software libraries and drivers for the host CPU. The memory bandwidth is 10 GB/s. Though more bandwidth is available, it is a trade-off with routability. More research is required to investigate methods to use the full memory bandwidth, such as leveraging multi-FPGA structures. Khan et al. [50] use the HMC memory for assisting packet processing look-up on FPGAs. The authors present the suitability of HLS as an intermediate language for packet processing. An AC-510 platform is leveraged, which integrates a 4 GB Micron Hybrid Memory Cube connected to a Xilinx Kintex Ultrascale FPGA. HLS supports ease of programming and allows a close correlation between the packet processing kernels and the corresponding processing units. The disadvantage, however, is the limited expression and abstraction of hardware.

Pantho et al. [70] propose a framework for automatic mapping of OpenCV image processing kernels to transparently benefit from the performance and concurrent memory channels of HMC. The mapping process identifies kernel functions in software that are mapped then to hardware components, pre-synthesized in the FPGA. Data transfer and memory access scheduling are also handled automatically by the framework. A workstation with four HMC-enabled AC-510 FPGAs attached through PCIe connections is used.

ScalaBFS [61] and Kara et al. [48] both propose accelerators for data analytics workloads. ScalaBFS [61] is an open-source accelerator for graph processing, with focus on the Breadth-First Search algorithm. ScalaBFS is implemented on Xilinx Alveo U280 FPGA card and tested with real-world graphs. Kara et al. [48] analyse the benefits of HBM on FPGAs for data analytics workloads such as range selection, hash join, and stochastic gradient descent for linear model training. The designs are integrated into MonetDB to study the data movement and partitioning trade-offs. The target platform is the Alpha Data ADM-PCIE-9H7 [76] card attached to a POWER9 machine via Open-CAPI [95]. Yang et al. [108] propose a parallel hash table accelerator implemented on an HBM-based FPGA system. The accelerator has a dedicated processing engine and hash table copy designed to connect to one or more adjacent HBM channels. The Intel Stratix 10 MX 2100 FPGA is used. Zhang et al. [110] aim to accelerate graph processing workloads using an Intel-based HMC design. The architecture uses the PicoFramework, and is implemented on both an AC-510 and an AC-520 FPGA-HMC platform. Improvements to the graph algorithm can include degree-aware adjacency list reordering to decrease the number of unneeded edge checks. A compression technique is also applied to the adjacency list to decrease the data access to the external HMC memory.

Nero [92] accelerates complex memory-bound weather prediction stencils. Scalability analysis is conducted for both DDR4 and HBM-based FPGA boards. It sends/receives data to/from the host using **Coherent Accelerator Processor Interface (CAPI2)**. COSMO API manages offloading jobs to NERO. SNAP (Storage, Network, and Analytics Programming) supports integration of the COSMO API.

Mcvicar et al. [66] use a Micron SB-800 board featuring four Altera Stratix V FPGAs, and a 4 GB HMC with a single high- speed serial link to each FPGA to accelerate K-mer counting. The HMC's high random-access rate is suitable for large Bloom filters used in k-mer counting. For optimizing performance, a given HMC link can only access its four HMC vaults, so each of the four FPGAs works independently on a small Bloom filter.

Lloyd et al. [63] investigate accelerating several applications such as RandomAccess, PageRank, and Image differencing through in-memory data restructuring. The rearrangement hardware executes data reduction, not compute offload. An FPGA emulator on a Xilinx Zynq 7000 System-on-Chip is used to evaluate the near-memory hardware structures that organize data into a cache-friendly layout. Applications are partitioned between restructuring data in memory and computing with data in the CPU. A more comprehensive use case would include multiple data rearrangement engines, managed by multiple CPUs. HRL [18] proposes a reconfigurable array NMC system tailored for analytics workloads including MapReduce, graph, and deep neural networks. The architecture integrates coarse-grained and fine-grained logic blocks. HRL arrays are leveraged as NMC processing elements, with support for efficient communication and synchronization across multiple stacks and memory channels. Each vault in the memory stacks contains multiple processing elements to take advantage of the large bandwidth and high parallelism. HRL has the power efficiency of an FPGA and the area efficiency of CGRA. Although both architectures are not implemented on an FPGA board with an HMC included, the PEs are designed based on Xilinx devices. The performance of the NMC systems in HRL are modeled using simulation.

Volcan [81] explores the interactions and issues of compatibility and performance between HLS and HMC. An interface of HLS-FPGA-HMC technologies is developed. It consists of a hardware design and a software methodology. The hardware block design is implemented in Verilog and HLS and the software applications are designed in C/C++. The hardware of FPGA and HMC memory (AC-510 board) is connected to the x86 Linux system host via PCIe. Pico framework includes the software driver on the host machine and a hardware interface. The HLS compatibility problems with HMC memory are due to HMC FLIT addressing (16 bytes) and non-burst support. The HLS compilers designed for DDR as off-chip memory generate an incompatible byte addressable memory request for HMC. The Volcan framework converts the incompatible memory address from the HLS into a compatible HMC memory address. To improve performance, the hardware design is augmented with a software methodology.

6 GENERIC ARCHITECTURES

2GRVI Phalanx [24] and MEG [111] are generic HBM-based NMC architectures using general-purpose Risc-V cores as processing elements. 2GRVI Phalanx [24] is a parallel processor and accelerator array overlay framework. Processing elements and accelerator cores are integrated into a shared memory architecture on Xilinx FPGAs equipped with an HBM. Communication across elements is achieved by message passing through the Hoplite torus soft NoC. Parallel programming is achieved by an OpenCL-like programming model and tools. The system is a kilocore RV64I SoC. MEG [111] is an open source, cycle-exact, RISC-V-based full-system emulation infrastructure using FPGA and HBM. A Linux image is available for the system, and can be booted to allow software-hardware co-design in a full-system environment. MEG provides performance monitoring utilities to control the performance optimization. A lightweight IOMMU provides virtual memory support for near-memory accelerators. A prototype of MEG is implemented on a HTG-937 board that includes a combination of a Xilinx Virtex Ultrascale+ FPGA and an HBM module. Four RV64GC out-of-order BOOM cores are integrated into MEG. There is plenty of opportunity left for improving the platform, such as the usability, flexibility, portability, and easy-to-use software interface.

OpenHMC [86] and Schmidt et al. [87] investigate an efficient memory controller on an FPGA for the HMC. OpenHMC [86] is a vendor-agnostic, fully synthesizable, open-source Verilog implementation of an HMC host controller that can be configured for different datapath widths and HMC link variations. Due to its modular design, the controller can be integrated in many different system environments. It includes a **Universal Verification Methodology (UVM)**-based test environment. The functionality of openHMC is completely independent of HMC link width. OpenHMC outperforms a comparable controller by Altera in terms of configurability and resource utilization. However, the openHMC controller still requires optimization for core latency. Schmidt et al. [87] investigate a system consisting of an FPGA in combination with an implementation of the openHMC host controller. The study observes that maximizing the performance of the HMC requires properly ordered request streams to avoid flow-control issues. The authors investigate offloading computation using atomic operations to mitigate computation and communication overhead. Complex operations, however, still require processor interaction. An evaluation of sustained bandwidth of chained topologies is yet needed.

7 PERFORMANCE CHARACTERIZATION ARCHITECTURES

Hadidi et al. [28], CoolPIM [67], and Hadidi et al. [29] aim to characterize the behavior of HMC memory on FPGA. Hadidi et al. [28] characterize the thermal behavior of HMC and identify temperature as a limitation for HMC-based designs. A Verilog realisation of GUPS and the Pico API [77] are used to run synthetic workloads on the AC-510. The authors conclude that optimizing HMC bandwidth requires large accesses with a mix of reads and writes. Furthermore, HMC accesses do not benefit from spatial locality. CoolPIM [67] is a study of thermal constraints of NMC using an HMC 1.1 prototype. The study concludes that simply offloading applications to NMC results in a thermal bottleneck and decreases performance. They propose a source throttling approach that adjusts offloading intensity so that the temperature does not peak. Hadidi et al. [29] run GUPS on Micron's AC-510 accelerator board on a Pico SC-6 Mini [79] machine to investigates the HMC access and bandwidth characteristics. The authors conclude that there is a trade-off between optimizing HMC bandwidth and latency. Another important design point is to optimize queuing delays introduced by the HMC internal packet-based structure. The internal NoC complicates QoS for memory accesses because of variations in latency even within an access pattern. However, it ensures QoS for smaller packets at a cost of decreasing bandwidth.

Similarly, HBM Connect [9] and Shuhai [33] characterize and benchmark the behavior of the HBM memory on Xilinx's Alveo U280 accelerator card. HBM Connect [9] aims to design a high performance interconnect for FPGA HBM boards. HLS-based optimization methods are leveraged to improve the performance of AXI bus masters and switches. The authors also present a high-performance customized crossbar, which can be used as an alternative for the built-in crossbar. Shuhai [33] is a benchmarking tool to investigate the underlying details of HBM on an FPGA. The authors observe that HBM can theoretically provide up to 425 GB/s memory bandwidth but how the HBM is used has a significant impact on performance. Shuhai can be adapted to other FPGA boards or other generations of memory. Choosing the suitable address mapping policy is crucial to achieving high bandwidth. Various address mapping policies provide an order of magnitude throughput differences for workloads with a typical memory access pattern (Repetitive sequential traversal) on HBM, indicating that the address mapping policy must be matched to a particular application. Additionally, they note that the latency of HBM is much higher than DDR4. The connection between HBM chips and the associated FPGA is done via serial I/O connection, introducing extra processing for parallel-to-serial-to-parallel conversion. They also observe that spatial locality greatly improves performance. Random memory access that does not keep spatial locality leads to low memory throughput.

8 PERFORMANCE COMPARISON OF RECENT NMC ARCHITECTURES

In Table 4, we observe the performance and power consumption of recent FPGA-based NMC architectures grouped by similar workloads. It can be seen from the table that in most proposals, the achieved bandwidth is still much smaller than the theoretical bandwidth. One reason for that is random and irregular access patterns, which limit the achievable bandwidths of each HBM PC using DRAM. Additionally, bandwidth is limited if the FPGA logic runs at relatively low clock frequencies. We also analyse the resource utilization in Table 4. Four kinds of resources (DSP, LUT, FF, and BRAM) are considered.

Before the emergence of high-bandwidth memories and the near-memory computing paradigm, most FPGA-based designs achieved less speedup than GPUs, but had the advantage of better energy efficiency. FPGA-NMC designs have changed this trend as the high bandwidth offered by 3D-stacked memories enables a large number of parallel computations, which sometimes outperform GPUs. As an example, Liu et al. [61] compare the performance of Gunrock GPU (64 HBM PCs) against their FPGA-HBM solution (32 HBM PCs). Their results show that for sparse graphs with short neighbor lists, the performance is almost similar for both architectures. This can be attributed to the small burst lengths of the memory accesses exhibited by such graphs, which make the HBM the system bottleneck. The FPGA solution implements customized BRAM bitmaps for the many memory accesses, and thus achieves a performance as high as the GPU. Additionally, the power efficiency of the FPGA is 5× to 10.× better than the GPU, due to the low power consumption of the Alveo U280. Figure 7 shows the speedup and power efficiency values of several benchmarks compared to a GPU solution. The x-axis shows the benchmark name and the paper reference.

Some works also report the speedup over CPU solutions. For example, Parravicini et al. [72] report a speedup for their FPGA-HBM architecture as high as 100× over a parallelized CPU baseline, and a power efficiency improvement of 400×. Compared to a GPU, their architecture is 2× faster and 14× more power-efficient. The power efficiency of the HBM architecture proposed by Sgherzi et al. [88] is 6.5× better than a CPU baseline. Mcvicar et al. [66] measure the speedup of their HMC-based architecture over a CPU baseline as 9.31× to 17.6×.

Table 4. Comparison of NMC Architectures Grouped by Workload

Work	Benchmark	FPGA	Freq.	Mem.	Peak Perf.	Power	Peak BW	Resource %			
								DSP	**LUT**	**FF**	**BRAM**
					Data Analytics						
[110]	RMAT graphs [5]	Arria 10 GX 1150	125 MHz	4 GB HMC	79.8 GTEPS	43.6 W	120 GB/s	4	64		7
[61]	RMAT graphs [5, 12]	Ultrascale+	90 MHz	8 GB HBM	19.1 GTEPS	32 W	46 GB/s	—	39	13	46
[67]	GraphBIG [68]	Kintex	—	8 GB HMC	—	—	320 GB/s	—	—	—	—
[48]	Database analytics	XCVU37P-2E	200 MHz	8 GB HBM	—	—	190 GB/s	38	55	47	56
[40]	Inference [27]	XCU280	140 MHz	8 GB HBM	379.45 GOPS	—	—	57	45	30	85
[70]	OpenCV filters	XCVU060	250 MHz	4 GB HMC	—	14 W	—	—	—	—	—
[66]	SAX, SHA-256 hash	Stratix V	—	4 GB HMC	57.5 KMERS/s	—	—	—	—	—	—
[108]	1M hash table	Stratix 10 MX	225 MHz	8 GB HBM	3575 MOPS	—	—	—	8	—	1
					Sparse Matrix Operations						
[88]	Top-K eigensolver	XCU280	225 MHz	8 GB HBM	—	38 W	72 GB/s	16	42	13	15
[72]	Top-K SpMV	XCU280	225 MHz	8 GB HBM	—	35 W	460 GB/s	17	35	33	20
[93]	SuiteSparse [11]	XCU280	189 MHz	8 GB HBM	181.1 GOPS	52 W	460 GB/s	36	29	26	76
[32]	PageRank (orkut)	XCU280	165 MHz	8 GB HBM	6973 MTEPS	44 W	285 GB/s	8	35	21	24
[14]	Googleplus, hollywood, pokec, OGB, transformer	XCU280	237 MHz	8 GB HBM	16.65 GOPS	45 W	258 GB/s	8	47	22	7
					Packet Processing						
[82]	1M rules	Kintex Ultrascale 060		4 GB HMC	16.4 Mbps	—	10.6 GB/s	40	35	81	42
					Weather Prediction Stencils						
[92]	Horizontal diffusion	XCVU37P-2	—	8GB HBM	120 GFLOP/s	1.5 Gflop/W	—	4	11	6	58

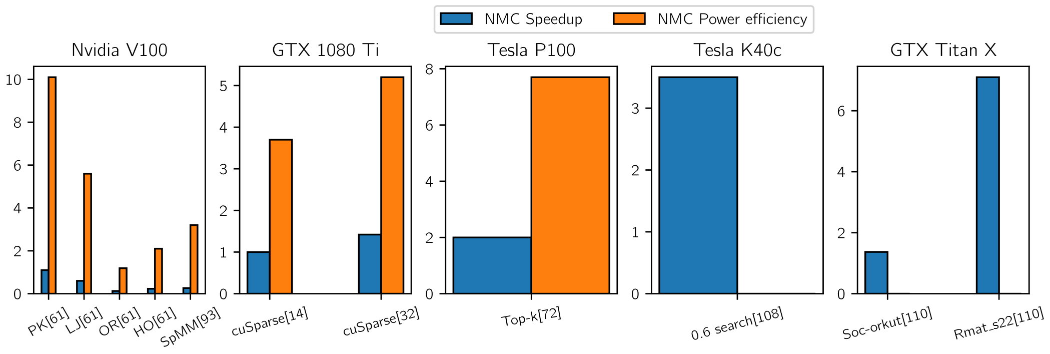

Fig. 7. Speedup and power efficiency values of NMC compared to a GPU solution.

From the analysis performance of FPGA-based NMC systems, we observe that application-specific accelerators involve both optimizations on the host-side and the hardware. In the following sections, we will focus on HBM-based architectures and introduce previous work in these two aspects, respectively. Table 5 classifies HBM-based designs according to the techniques presented in the next two sections.

9 HARDWARE DESIGN OF FPGA-HBM: BANDWIDTH-EFFICIENT ARCHITECTURE

9.1 Processing Unit Design

Most FPGA-HBM architectures offer several low-profile processing elements that operate independently. Each processing element operates on a partition of the input data. Designing complex processing units and heavily optimizing them is not the main focus of such architectures. Rather, the goal is to optimize memory paths to and across the processing elements.

9.1.1 Scale-out Processing Units. Highest bandwidth can be reached only if the physical memory channels are accessed in parallel. Additionally, how the data is accessed is incredibly important for HBM performance. If there is only one copy of the data, then it creates a bottleneck when accessing the HBM from multiple cores in parallel, removing the HBM bandwidth advantage. To efficiently exploit the HBM, data replication is needed such that each processing element can access its own physical memory channel without reverting to the crossbar [48].

It is not trivial to achieve high computing parallelism, because the parallel implementation of some applications is not able to guarantee execution locality. ScalaBFS [61] divides the graph data into several subgraphs, and stores each subgraph in an HBM PC to ensure locality of accessing. Additionally, ScalaBFS implements multiple computing cores to provide high computing parallelism. In Microrec [40], embedding tables are distributed to the HBM banks such that each bank contains one or a few tables, and up to 32 tables can be looked up concurrently.

Sgherzi et al. [88] use multiple SpMV computing units to process partitions of the input matrix, where computing units process equal numbers of rows. Every computing unit operates to produce a part of the output vector, and partial results are collected and duplicated across HBM channels to be available in the next iteration. The vector is replicated five times for each computing unit, so that inefficient global access across HBM is prevented and parallel computation is enabled. Similarly, Parravicini et al. [72] assign each computing unit to compute a partition of the input matrix. Each core uses a single HBM channel to read the input matrix and store its output at the end of the computation. As processing elements read 512 bits every clock cycle through their HBM memory ports and memory transactions are realised in continuous maximum length AXI4 bursts, the FPGA can theoretically achieve the maximum bandwidth offered by HBM, by coalescing transactions with packets containing multiple non-zero entries, and without expensive distributed memory controllers.

In FleetRec [41], the embedded tables lookup process is highly concurrent. The number of tables stored in every memory channel (DDR and HBM) is balanced so that the lookup workload is equal across channels. The embedding vectors are read in parallel and concatenated before they are sent to the computation module. As a result of the high embedding lookup throughput due to the high retrieval concurrency, the computation workload is partitioned to multiple FPGA nodes.

Nero's [92] design integrates several processing elements that leverage data-level parallelism. Each processing element owns a separate HBM PC; hence, the design uses as many HBM ports as the number of processing elements. Such parallelism allows the exploitation of the high HBM bandwidth as each processing element reads its data from an independent port. Yang et al. [108] provide a design that can run with a maximum of 16 processing elements. Increasing hash table copies as many as possible has a negative effect on throughput because of the increased overhead of communication across processing elements.

9.1.2 Dataflow Programming. The HLS dataflow programming style enables exploiting a workload's task-level parallelism. Most FPGA-HBM application-specific accelerators implement C/C++ functions operating in parallel and communicating through streaming FIFOs. This method enables kernels to provide high throughput while maintaining small BRAM consumption.

The architecture proposed by Kara et al. [48] scans the input data N times (where N is the number of epochs), and distributes it to the processing elements, which execute following the dataflow paradigm. MiroRec [40] provides a pipelined accelerator where several data items are processed by the accelerator in parallel in multiple stages. The embedding lookup and the computation stages are pipelined. Every DNN processing module further implements three pipeline stages: feature broadcasting, computation, and result gathering. Inter-module communication is achieved through FIFOs built by BRAMs or registers. Nero [92] leverages the algorithm parallelism via pipelining. All the hardware modules work in a dataflow fashion that enables task-level parallelism. The accelerator in Graphlily [32] employs FIFOs to decouple the hardware modules and allow efficient dataflow operation. Each processing element is a three-level pipeline: fetching a temporary result from the output buffer, updating the result value, and finally writing it back to the output buffer. The processing elements perform data forwarding to resolve **read-after-write (RAW)** dependencies.

9.1.3 Processing Elements Clusters. Scale-out FPGA-HBM designs involve multiple parallel elements operating on portions of the data. Often, several processing elements require access to the same data. For example, in the case of SpMV, each processing element needs random accesses to the same input dense vector. Replicating the input vector for each processing element is not scalable to many HBM PCs for a design with numerous processing elements. To mitigate this issue, the processing elements can be grouped into multiple clusters, with a data loader module and a result collection module. Each cluster is assigned one HBM channel. The data loader reads in the data from the external memory and provides each cluster a copy of the data, which will be shared inside that cluster. The collection unit concatenates the partial results from the clusters to produce the final result and writes it back to the off-chip memory.

HiSparse [14] uses the processing element clusters design to compute SpMV where the dense vector is shared between all the processing elements of a given cluster and the shared data is banked to improve throughput. The input vector is therefore replicated for every cluster. Graphlily [32] similarly implements multiple processing element clusters, each assigned its own HBM channel. Each processing element cluster integrates a matrix reader, a vector reader, and an array of processing elements, each operating on one input stream. This processing element cluster design is similar to a SIMD computation on CPUs in that both execute vectorized calculations. The main difference is that the processing element cluster needs significantly less synchronization. When a lane in a SIMD module finalizes operating on a row of the matrix and writes back the output, the rest of the lanes stall. The processing elements in Graphlily, however, synchronize once when the processing of the whole matrix is finished. Sextans [93] implements 8 processing engine groups to calculate Sparse-Matrix Dense-Matrix multiplication. Each group consists of 8 processing elements. To stream in the partitioned sparse matrix from HBM to each processing element group, 8 read modules are implemented. A separate module is deployed to stream the dense matrix in a given window size from HBM and propagate the partial data to the eight processing element groups. Sgherzi et al. [88] use five computational units for SpMV. The hardened AXI switch controls the number of processing units, which fully use the 32 AXI master channels to HBM. Each SpMV compute unit calculates a portion of the output vector. Partial results are replicated five times for each compute unit across HBM channels to use them in the following iteration.

Yang et al. [108] take a different approach where they divide the HBM PCs into multiple groups, and each processing element is connected to a group of PCs. The number of PCs that is assigned to each PE is an architecture parameter that can be decided at design time. The full hash table is stored for each group of PCs. Table entries are equally divided across the PCs of each group.

9.1.4 Simple Synchronization Constructs. Most architectures we surveyed do not implement explicit synchronization barriers for the processing elements. Simple FIFOs are used to maintain loose synchronization. The FIFO depth is an architecture-dependent parameter that decides the highest number of ahead/delay cycles of asynchronization allowed between processing units. If the asynchronization cycles are greater than the FIFO depth, then there is one processing element that has consumed all elements of its FIFO while another processing element's FIFO is full. At the producer side, the sending is stalled. The former processing element stays idle, because the connected FIFO is empty. After the latter element consumes at least one element in the connected FIFO, the processing resumes.

Sextans [93] uses FIFO to obtain loose synchronization that is implicitly maintained in the broadcasting where one Read module is the producer, and the processing units are the consumers. Yang et al. [108] ensure that their processing elements maintain the AXI in-order response, hence the ordering between their request queue and response queue for the same HBM channel is kept synchronized. chronized. The arbitration module is a simple FIFO. In MicroRec's inference [40], the input feature vectors are broadcasted to all processing elements, and the partial results are collected when the layer computation is finished and sent to the next layer. The authors observe that this broadcasting and gathering implementation for the interconnection across processing elements has less latency than other interconnect designs, such as systolic arrays [99]. The data forwarding design in the systolic arrays causes a higher inference latency, especially for designs with many processing elements.

Kara et al. [48] interface several processing elements to the HBM. A central control unit asynchronously starts/stops and monitors the processing elements. The control unit communicates with the CPU through a register read/write interface enabling asynchronous control by software. Hence, each processing element can be started/stopped individually in parallel. Synchronization among them can be implemented in software.

9.1.5 Efficient Pipelining. The irregularity in the computations and access patterns of some applications lead to bank conflicts and carried data dependencies, which reduce compute occupancy. Bank conflicts and carried dependencies present difficulties to HLS compilers, which usually employ static scheduling to produce a conservative pipeline with high performance.

HiSparse [14] makes the observation that using the conventional HLS implicit control logic for their shuffle unit, which manages data replication and banking for several processing elements, is suboptimal. The static scheduling of HLS is unable to produce appropriate arbitration logic as the compile-time analysis assumes the worst-case traffic pattern. Therefore the generated HLS schedule processes input lanes sequentially, with an **initiation interval (II)** equal to the number of input lanes. HiSparse solves bank conflicts through data reordering. Re-sending logic is implemented in HLS by iteration-level modeling, which selects the computations the datapath should execute per iteration. If the datapath needs information from past iterations, then special data structures are leveraged to hold and forward the information. To fully pipeline the processing elements, load-store forwarding is used to dynamically resolve RAW dependencies.

9.2 Multi-level Memory Optimizations

Inefficiently accessing memory and fetching huge amounts of random data from off-chip memory significantly reduces FPGA-HBM architectures performance. Multi-level memory optimizations with HBM are required for efficient accessing and streaming.

9.2.1 Trade-off between On-chip and Off-chip Memory. An important design goal for FPGA designs is to minimize off-chip memory accesses, given limited on-chip memory capacity. The size of on-chip memory, i.e., BRAMs and URAMs, is always limited. For example, the RAM size on the

Alveo U280 is 41 MB. In many of the works we studied [40, 41, 61, 72, 109, 112], not all the data fits in the FPGA's on-chip memory, which means that some data is read from external memory. Since external memory access is quite expensive, the aim is to decrease external memory accesses. This is achieved by leveraging precious on-chip memory resources to store frequently modified data. Careful selection must be performed, to decide where data should be placed. Modern FPGAs now enjoy increasingly-growing on-chip memory capacity, which can store larger amounts of data.

Sextans [93] divides the random memory read and write by an access window, so random memory accesses are kept to fast on-chip memory. The read-only dense matrix is stored in BRAMs to limit random read from on-chip memory. A scratchpad memory is leveraged to maintain the random accumulation operations on-chip using the URAM. FleetRec [41] aims to prevent the random DRAM accesses from retrieving embedding vectors on many tables, which cause severe underutilization of HBM bandwidth and reduced performance. The random accesses of retrieving vectors from different tables is sub-optimal to the hardware, because it forces the row buffer of the DRAM bank to be charged and discharged continuously. Since a given vector is short (usually between 4 and 64 elements), the DRAM bandwidth utilization is very low. Hence, as many small tables as possible are stored in SRAM, since that allows low-latency and high-concurrency vector retrieval. Nero [92] divides on-chip memory to avoid the stalling of the pipelined design, since the on-chip BRAM/URAM has only two read/write ports.

9.2.2 Streaming Access. To exploit the high bandwidth of HBM, architectures must possess a high degree of parallelism to effectively use multiple channels, and maintain streaming accesses to hide latencies. In Sextans [93], the three matrices in SpMMs are partitioned according to the processing window size and the partitioned matrix fragments are stored in HBM. Individual element accessing is not issued to HBM, but rather accessing a matrix fragment. This allows efficient streaming accessing. Similarly, Yang et al. [108] read data batches instead of one slot at a time to improve HBM efficiency and collision handling overhead. FleetRec [41] performs single-millisecond-level inference on small and medium models using medium batch sizes (e.g., 256), while the CPU needs around 10 ms. Using a medium batch size of 256 already allows FleetRec to achieve close-to-peak throughput (around 85% of the peak performance), while the CPU only achieves around half of the throughput reached with batch sizes of 1,024.

9.2.3 External Memory Burst Access. To enable serving random memory accesses, a crossbar is needed to globally access all the memory channels of HBM. Taking the Alveo U280 as an example to study, there are two types of crossbar offered by Xilinx.

- A Full Xilinx AXI Interconnect. The full Xilinx AXI Interconnect IP [102] only supports up to 16×16 AXI interconnect, which is exceedingly inefficient considering that the U280 has 32 HBM PCs.
- Xilinx's Switch Network. Liu et al. [61] analyse the performance of the Xilinx's built-in switch network when there exist memory accesses that are global across HBM channels. Shuhai [33] is used to measure the performance of each AXI channel by reading data from adjacent HBM channels. They use a data width of 256 bits, an outstanding buffer size of 256, and a burst length of 64. The pressure on the switch network grows with increasing the number of HBM channels each AXI channel reads from. This has a severe impact on the achievable throughput of each AXI channel. Crossing 32 HBM channels achieves less than 0.5 GB/s, more than 20 times less than the bandwidth when no crossing occurs. Hence, Xilinx's built-in switch network cannot handle globally random accesses. Such accesses cause significantly low performance for the system. For the design proposed by Liu et al. [61], Xilinx crossbars cannot reach the memory bandwidth needs of the BFS engine. Sgherzi

et al. [88] and Choi et al. [9] also observe that the hardened AXI switch in the Alveo U280 provides very low performance if several AXI master channel access the same HBM bank. Only 32 AXI master channels are available, and small memory transactions (32 bits) have the same performance as larger transactions, preventing bandwidth sharing.

Enabling external memory burst access to several HBM PCs in the HLS programming environment is non-trivial. This is the case when multiple AXI masters access multiple PCs in many-to-many fashion [9]. Vivado HLS provides a specific coding style to allow access to multiple PCs. This coding style depends on the concept of a Vivado HLS bundle, which corresponds to an AXI master. The advantage of this HLS coding style is that it is easy to code and area-efficient. However, it creates two problems.

The first issue is that when accessing multiple PCs from one AXI master, data from different AXI masters typically share the lateral connections, which decreases the resultant bandwidth. As an example, Choi et al. [9] analyze a 16×16 unicast interface, where 16 AXI masters (M0–M15) access 16 HBM PCs (PC0–PC15) in round robin fashion. In that case, the lateral connections become the bottleneck. AXI master M0 needs to cross three lateral connections of unit switches to reach PC12–PC15. Multiple crossings considerably decreases the overall memory bandwidth. The effective write bandwidth of a given PC is reduced to as low as 7.5 GB/s. The decline in the effective bandwidth increases as more AXI masters compete with each other to access a given PC.

The second issue is that loops containing write variables whose values differ in the various iterations imply to Vivado HLS to set the AXI burst length to one for each write operation. This also negatively affects the HBM maximum bandwidth. As an example, Choi et al. [9] analyze the bucket sort application, where processing elements distribute the keys to output HBM PCs based on its value (each PC corresponds to a bucket). Since each AXI master transfers data to any PC via the built-in crossbar, there is a one-to-one connection between a processing element and an AXI master. The built-in AXI crossbar is used to distribute the keys to the processing elements. Vivado direct access results in only 59 GB/s among 16 PCs as it is possible that two consecutive keys from input PC will be sent to different buckets (output PC). Existing HLS tools are not capable of automatically buffering the data for burst AXI access to each HBM PC. The burst value in case of different output PCs is calculated as one, which severely harms the obtained bandwidth.

To mitigate burst access problems, Choi et al. [9] present an HLS-based buffering scheme that allows sharing a single FIFO among multiple virtual channels in HLS. This increases the utilization of BRAM depth as the FIFOs for many different PCs. Additionally, the authors found that bandwidth benefits by integrating AXI burst buffers and custom crossbar levels. For some applications such as bucket sort, a full custom crossbar outperforms the built-in crossbar if the result from several processing elements is written into a single memory space.

9.3 Frequency Optimization

Modern FPGA architectures incorporate several chip dies to provide more on-chip resources for developers to implement large designs. However, the interconnections between dies extend across the silicon interposer, which results in considerable latencies. Additionally, not all dies have direct connections to the HBM interface. The Alveo U280 has a heterogeneous architecture, where all the HBM channels connect to a single die. The Alveo U250, however, has a homogeneous architecture, where four DDR channels each connects to one die. On U280, all HBM ports reside at SLR0. Therefore, the AXI interfaces from the processing cores present in other SLRs must be routed across SLRs to reach the HBM. A hardware module's access to HBM would incur a long cross-die delay when the module is placed on a die far from the HBM interface, raising challenges to timing closure. Since FPGA-HBM architectures are inherently scale-out designs, they are more prone to be constrained on resource utilization. This leads to the usage of multiple SLRs.

One simple method of building an accelerator is writing one OpenCL kernel to iterate over all data partitions. This can be referred to as a monolithic accelerator. The monolithic accelerator design does not take into account the physical layout of the device. Therefore, it is challenging to find an optimal place-and-route solution. Without manual floorplanning, the placement tool will attempt to fit all logic onto the same chip die to decrease expensive die boundary crossings, leading to a highly condensed placement and routing congestion. Manual floorplanning is not always effective to improve the place-and-route results and improve the timing. The data and control signals generated by HLS frequently include combinational logic, which cannot be pipelined to amortize the die-boundary-crossing timing penalty.

Hisparse [14] takes the split-kernel approach to address this problem. Split-kernel means splitting the accelerator into several OpenCL kernels to reduce die-boundary crossings and allow the pipelining of inter-die connections. The logic to be placed on a single chip die is limited to one kernel. Different kernels communicate through pipelined connections to hide the latency of die-boundary-crossing. The AXI standard is used for kernel communication as it is common for FPGA design tools. The split-kernel approach is appropriate for the heterogeneous architecture of the FPGA-HBM hardware. Registers are added to paths that cross die boundaries to pipeline the remote connection. For the same count of clusters and the same buffer size, the Hisparse split-kernel SpMV obtains 237 MHz, significantly higher than the monolithic SpMV, which achieves 117 MHz. We note that the implementation of HiSparse on Xilinx Alveo U280 uses 18 HBM channels, offering a memory bandwidth of 258 GB/s. Further scaling to more than 18 HBM channels causes routability issues.

Kara et al. [48] deal with this challenge to some degree by first constraining one processing core to be placed in only one SLR, and second adding AXI-interconnects with internal buffering in SLRs in between the compute engine and the HBM. For example, for a core in SLR2, two AXI-interconnect modules in SLR1 and SLR0 are added to ease routing. In Graphlily [32], the large output buffer results in a high-fanout logic structure. The technique proposed in Reference [26] is used to decrease wire delays by implementing a pipelined multi-level tree structure. This optimization increases the frequency from 145 to 165 MHz. A more general solution to this issue is hardened overlays on top of the FPGA fabric that route signals in wide busses rather than bit-wise routing. More advancements on the device level are needed to make the FPGA-HBM development easier.

9.4 Data Movement

In the presence of a host CPU, the data to be consumed is initially stored in the main memory of the CPU. How to bring this data to the FPGA to be used by the processing elements that are also connected to the HBM is one of the problems that the NMC architecture should solve. One solution can be to connect each processing element both to the DMA communicating with the CPU and to one port on HBM. This however creates a bandwidth imbalance and requires arbitration when accessing the DMA by multiple processing elements, complicating system architecture and leading to difficult routing. Another alternative is presented in Reference [48], where two dedicated data movers move the data between the CPU memory and the HBM. The data movers occupy 2 of the 16 ports that the HBM-shim exposes. The remaining ports are usable by the computing elements, consuming and producing data via the HBM.

PCIe protocols incur increasing latency, do not support arbitrary lane organization to optimize bandwidth, and do not offer coherence. Supporting coherence adds the overhead of explicit copies, locks, and critical sections. Although IBM's CAPI coherent I/O interface supports coherence, it is still dependent on PCIe for data movement. OpenCAPI [95], however, ensures high bandwidth and provides a low latency coherent interface to memories and accelerators. As such, OpenCAPI provides an opportunity to integrate NMC processing units in a systematic and scalable way.

We now refer to some NMC architectures that investigated the data movement challenge. Such systems exploit the **Dual-inline Memory Module (DIMM)** to integrate processing modules within DRAM memory. Although not HBM-based, these architectures provide an inspiration for tackling this issue. Contutto [96] uses IBM's DMI memory interface and integrates an FPGA between the host memory interface and the DRAM DIMMs. No modification to the host system or the DIMMs is required. The FPGA communicates with the memory through the DIMM, hence does not fully leverage the benefits of NMC. Closer coupling with the DRAM chips is still required.

Acceleration DIMM (AXDIMM) [4, 49] is a DDR4-compatible FPGA-based near-memory acceleration platform. A smart buffer chip acts as a conventional buffer chip for a high-capacity DIMM. ARM cores and FPGA fabrics are provided for memory-intensive computations. AXDIMM allows for simultaneous access to every rank within a DIMM, hence it increases the bandwidth and enhances the performance. Processing the data locally in the DIMM instead of moving it to the CPU, which additionally decreases power consumption. These properties allow AXDIMM to integrate NMC with a host via the standard memory interface. Samsung provides a use case for AxDIMM by incorporating a deep learning recommendation model with AXDIMM to demonstrate that AXDIMM can boost performance and power efficiency.

TensorDIMM [54] investigates accelerating gather-and-reduction by incorporating processing modules for every rank in the buffer chip of DRAM to leverage the internal bandwidth. The badwidth in this case is the single channel bandwidth multiplied by the number of ranks per channel. However, rank-level parallelism is not capable of reaching the maximum potential of NMC. NEST [34] accelerates k-mer counting by modifying the LRDIMM. Communication is achieved via the standard DDR channel with host support. Similar to AxDIMM and TensorDIMM, an NMC module is placed for each LRDIMM rank.

10 HARDWARE-ORIENTED HOST-SIDE OPTIMIZATION

10.1 Data Preprocessing

Data preprocessing is used to ensure that the input data is presented to the hardware through streaming accesses to amortize latency penalties. ScalaBFS [61] uses **Compressed Sparse Row (CSR)** and **Compressed Sparse Column (CSC)** to represent the graph data. The CSR contains an edge array, in addition to an offset array, which holds the offsets of the outgoing neighbor lists. The CSC differs in that its edge array holds the incoming neighbor lists. ScalaBFS employs both CSR and CSC such that the processing elements can easily access both the outgoing and incoming neighbor lists of a selected vertex.

Parravicini et al. [72] observe that the popular CSR format is not optimized for fully pipelined streaming accelerators due to the inherent data dependencies to access matrix values. Hiding the memory controller latency is challenging without hardware prefetching techniques. However, the **Coordinate (COO)** format consists of three arrays to store the two coordinates of a matrix element and the element's value. COO provides streaming burst accesses of non-zero matrix elements to fully use the bandwidth, as data-dependent memory accesses determined by the number of non-zero values of each row do not exist. However, COO incurs a large storage space, decreasing the effective operational intensity. The authors propose BS-CSR, a sparse matrix storage layout that incorporates the low memory footprint of CSR with the streaming properties of COO. As the HBM memory controllers favor memory transactions of blocks between 256 and 512 bits, each packet in BS-CSR is built as an independent CSR partition.

HiSparse [14] tailors the sparse matrix format to allow streaming accesses to each HBM channel and simultaneous access to several channels. This is achieved via partitioning, streamizing, and packing. Partitioning deals with matrices that are larger than buffer size by dividing the matrix along both the rows and columns. Streamizing assigns rows to processing elements in round robin fashion and integrates rows assigned to a given processing element into one stream, with a special marker at the end of a row to avoid indirect addressing and enable fully streaming accesses. Finally, the streams are packaged as streams of packets, where a packet consists of two elements. One stream of packets is written per HBM channel and accessed by a cluster of two processing elements, each processing one stream of elements.

Graphlily [32] aims to fully utilize the HBM streaming bandwidth by representing the sparse matrix in a tailored format that provides vectorized accesses to each HBM channel and simultaneous accesses to several HBM channels. **Cyclic packed streams of rows (CPSR)** is a sparse matrix storage format customized for HBM. CPSR provides packing of consecutive rows, and concurrent memory accesses to several HBM channels by interleaving the row packs in a cyclic manner. CPSR requires much lower storage capacity than that required by CSR.

In MicroRec [40], tables are combined using cartesian products to decrease random memory accesses.

10.2 Scheduling

Poor scheduling and workload imbalance makes parallelization, and thus exploitation of HBM bandwidth, difficult. Architectures aim to reach a balanced workload distribution among processing elements with an II = 1 pipeline. Sextans [93] employs non-zero out-of-order scheduling for SpMM computation. A non-zero element from the sparse matrix is scheduled at the earliest cycle where the row index of the scheduled non-zero has no RAW with the row index of non-zeros being processed in previous N cycles, where N is the distance of RAW dependency of a specific hardware platform. This scheduling obtains an II = 1 pipeline. It is integrated in the preprocessing of the sparse elements and provided as a host C++ wrapper.

GraphLily [32] implements a middleware to perform data transfer between the CPU host and the FPGA device, in addition to kernel scheduling. The middleware exposes the GraphBLAS interface to users for porting existing GraphBLAS programs to the hardware accelerators on the FPGA with little changes to the CPU/GPU versions. Graph algorithms are built by specifying the required modules and scheduling the execution order of the modules. A set of APIs manage data transfer between the host and the FPGA, and between kernels on the FPGA. GraphLily middleware is based on **Xilinx Runtime Library (XRT)**. The middleware leverages a simple scalar-vector add kernel in the GraphLily overlay to perform on-device data transfer by setting the scalar to zero. This is more than 2 orders of magnitudes faster than AXI Central DMA2, which XRT uses for on-device data transfer.

11 OPEN RESEARCH QUESTIONS

Based on our analysis of many existing FPGA NMC architectures, we identify the following key directions for future research, which are essential to exploit the full potential of processing near to high-performance memory.

11.1 HLS Tools Optimization

Low-level hardware description languages like Verilog require good knowledge of hardware design and hands-on experience. However, higher-level languages such as HLS can reduce programming efforts needed by FPGA programmers. Even though HLS increases programmability, high performance cannot be simply obtained without careful design of the hardware pipeline and the memory subsystem. Choi et al. [9] observe that the HLS programming environment does not provide an efficient communication architecture when multiple processing units access multiple HBM channels, leading to a severe impact on the effective bandwidth. They implement several optimizations

Table 5. Classification of HBM-based Architectures According to Design Techniques in Sections 9 and 10

Technique	Proposal
9.1.1 Scale-out Processing Units	[9, 14, 32, 40, 41, 48, 61, 72, 88, 92, 93, 108, 112]
9.1.2 Dataflow Programming	[9, 14, 32, 40, 41, 48, 61, 72, 88, 92, 93, 112]
9.1.3 Processing Element Clusters	[14, 32, 72, 93]
9.1.4 Simple Synchronization Constructs	[9, 32, 40, 41, 72, 88, 92, 93, 108]
9.1.5 Efficient Pipelining	[14, 32, 93]
9.2.1 Tradeoff between On-Chip and Off-Chip Memory	[14, 32, 40, 41, 61, 72, 92, 93, 112]
9.2.2 Streaming Access	[14, 32, 40, 41, 61, 72, 88, 92, 93, 112]
9.2.3 Optimizing External Memory Burst Access	[9, 33, 61, 88]
9.3 Frequency Optimization	[14, 32, 48, 93]
9.4 Data Movement	[48, 92]
10.1 Data Preprocessing	[14, 32, 61, 72, 93]
10.2 Scheduling	[32, 93]

targeting HLS such as the HVB and the mux-demux switch to overcome the shortcoming of current HLS HBM syntax. They also found out that for some workloads, utilizing a full custom crossbar provides the best trade-off of using on-chip memory in case the data from several processing units can be stored into a single memory space.

Shuhai [33] implements benchmarks in HDL while HBM Connect [9] implements benchmarks in HLS. Comparing performance results, HLS on the Alveo U280 has about 5X lower bandwidth when accessing data with a fixed address stride. The cause can be understood when investigating how the memory request is sent to the AXI bus. In Shuhai, a new memory request is sent as soon as the AXI bus is ready to receive a new request. In HLS, the generated AXI master has some limitations in book-keeping the outstanding requests. Since a burst access cannot be used for strided access, the number of outstanding requests becomes much larger than that of sequential access. As a result, HLS AXI master often stalls and the performance is limited to about 50–80 GB/s. In case new page in the HBM, and the bandwidth becomes equivalent to that of Shuhai. The bandwidth of the Intel Stratix 10 MX, however, decreases less with greater strides. Stratix 10 MX fixes the AXI burst length to one and instead depends on several outstanding requests even for sequential access. Therefore, the Stratix 10 MX's HLS AXI master is designed to handle more outstanding requests, and it can better retain the effective bandwidth even with short accesses.

Potential enhancements are required in HLS tools to better support developing high-performance accelerators. First, many workloads exhibit random access patterns in addition to data reuse, which requires efficient on-chip buffering. Hardware templates similar to shuffle units [14] need to be included in the HLS libraries. Second, some workloads such as sparse linear algebra operators often incorporate accumulation. Load-store forwarding should be implemented to serve as a general approach to resolving inter-iteration carried dependencies. The HLS tools can offer users some pragmas based on which the tool performs dependence resolution at compile time [14]. Additionally, when an accelerator consumes more resources than available in one die, the HLS tool should automatically switch to pipelined communication protocols and insert registers or at least warn the programmer of potential timing degradation [14].

11.2 Hardware Flexibility

An important direction is how to design a general-purpose accelerator that does not need to go through the slow FPGA development flow of synthesis, place, and route. The aim is to support execution of different input sizes directly by hardware as a general-purpose accelerator. In an ideal case, the accelerator should be called by OpenCL run-time without the need to customize the hardware and design details. A promising work in this direction is Hflex in Sextans [93]. HFlex is implemented via an iteration pointer list that resembles an instruction queue. The sparse input matrix is divided into several sub-matrices. Every sub-matrix is in turn encoded into an array of non-zeros. The arrays of all sub-matrices are stored linearly in a memory space. The iteration pointer list memorizes the starting index of each scheduled nonzero array. When computation starts, elements of the iteration pointer are considered as the loop iteration number. Therefore, the design enables the operation of an arbitrary SpMM without changes to the hardware.

11.3 Optimal Number of Processing Elements

It is unclear how to decide the optimal number of processing elements for a certain design. Given a fixed number of HBM pseudo-channels, there exists a number of processing units that can achieve the upper-bound performance. Intuitively, increasing the number of processing elements should lead to an improvement in performance. Nevertheless, there might be a break-point after which the performance will decrease with increasing processing elements, due to channel saturation. Additionally the required data replication per processing element might limit the number of processing elements due to HBM capacity. Yang et al. [108] observe that HBM can be saturated with 8 or more elements. When this happens, read heavy query distribution benefits from large number of processing elements. However, as query distribution becomes write heavy, lesser processing elements could perform better, especially when the memory efficiency is low. Inter-element communication overheads are the main reason causing the throughput drop with more elements. For instance, with 16 elements running in parallel, the design can easily oversubscribe the HBM. The optimal number of elements, however, varies across various works as it is application-specific. In ScalaBFS [61], the design reports small resource utilization but cannot scale to more than 64 processing elements due to placement and routing issues. The HBM bandwidth is constrained due to the low clock frequency of processing elements logic.

11.4 Remote Computing at the Memory Interface

Another promising direction of research is to explore enhancing the NoC-AXI RDMA bridges to perform remote computing at the memory interface, such as: scatter/gather accesses, block zero, block copy, add to memory, checksum, select, reduce, regexp, sort, and decompress. Such compute at the memory controllers should decrease the need for FPGA-spanning full HBM bandwidth read/write.

11.5 Optimizing FPGA Architecture for HBM

Increased bandwidth can strain the FPGA routing fabric, as it did in tests with several of the works we studied. Using high-bandwidth interfaces on current routing resources is challenging, and it is a complicated task to design hardware that is able to distribute wide memory response signals across the FPGA. Providing an overlay to augment the routing resources to distribute these high-bandwidth signals, optimizing multi-channel hardened memory controllers and perhaps hardened NoCs would be a promising FPGA architecture exploration. Systematic studies of various trade-offs to find the best design parameters for a future FPGA architecture is required. To stand up to the challenges of technology scaling and ever more demanding acceleration tasks, the FPGA routing architecture will have to evolve. Understanding existing routing architectures using tools such as NetCracker [75] is necessary for providing the right basis for innovation.

While the HBM architecture offers some space on the logic die to integrate a custom FPGA solution, one could consider a new HBM architecture with a larger logic die for computation to be closer to the memory. However, modern FPGAs are significantly larger than the existing HBM logic dies. Thus, this could be a very invasive approach and would require significant engineering effort and re-design.

11.6 Automatic Detection of Offload Kernels

Enhanced automatic detection of offload kernels is another promising research direction. Employing profiling and compiler techniques is beneficial to allow architectures to dynamically decide which kernels are suitable for NMC execution. The characteristics of kernels that obtain a high speedup when accelerated using FPGA-HBM architectures should be extracted, such that they serve as a guide for the offloading decision. Such work has been done before for different architectures [20, 64, 65, 69], but has not been adapted to FPGA-HBM architectures.

12 CONCLUSION

The decreasing performance and energy efficiency for modern memory-intensive workloads on conventional CPU systems, has led to a large amount of research in processing close to memory. This paradigm is enabled by FPGAs that are enhanced with 3D-stacked memories. Such FPGAs have been used in numerous works in recent years to accelerate various applications. In this article, we have analyzed and organized the literature of placing compute units close to a high-performance 3D memory on an FPGA.

According to the performance evaluation presented in Section 8, some FPGA-NMC architectures can achieve a power efficiency as high as 10× that of state-of-the-art GPUs. However, careful design is needed to leverage the full bandwidth of the HBM memory, especially in terms of controlling the access of processing elements to the HBM channels, and data replication to provide parallel computation.

By systematically reviewing the existing FPGA-NMC systems, we have identified key challenges that motivate future research directions. There is a number of open research questions in this field, many of which focus around designing a programming model for NMC architectures.

REFERENCES

[1] AWS. [n. d.]. AWS F1 instances. Retrieved from aws.amazon.com/ec2/instance-types/f1/.

[2] Baidu. [n. d.]. Baidu FPGA instances. Retrieved from https://cloud.baidu.com/product/fpga.html.

[3] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 105–117.

[4] Mohammad Alian, Seung Won Min, Hadi Asgharimoghaddam, Ashutosh Dhar, Dong Kai Wang, Thomas Roewer, Adam McPadden, Oliver O'Halloran, Deming Chen, Jinjun Xiong, Daehoon Kim, Wen-mei Hwu, and Nam Sung Kim. 2018. Application-transparent near-memory processing architecture with memory channel network. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 802–814. https://doi.org/10.1109/MICRO.2018.00070

[5] James A. Ang, Brian W. Barrett, Kyle B. Wheeler, and Richard C. Murphy. 2010. Introducing the graph 500. *Cray Users Group (CUG)* 19 (2010), 45–74.

[6] arm. 2020. AMBA AXI and ACE protocol specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite. Retrieved from www.arm.com.

[7] Oreoluwatomiwa O. Babarinsa and Stratos Idreos. 2015. JAFAR: Near-data processing for databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

[8] Jared Casper and Kunle Olukotun. 2014. Hardware acceleration of database operations. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'14)*. ACM, 151–160.

[9] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. 2021. HBM connect: High-performance HLS interconnect for FPGA HBM. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 116–126.

[10] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan,

Ahmad El Husseini, Tamas Juhasz, Kara Kagi, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven Reinhardt, Bita Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. 2018. Serving DNNs in real time at datacenter scale with project brainwave. *IEEE Micro* 38, 2 (2018), 8–20.

[11] Tim Davis and Yifan Hu. 2011. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.* 38 (Nov. 2011), 1. https://doi.org/10.1145/2049662.2049663

[12] Timothy A. Davis and Yifan Hu. 2011. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (Dec. 2011), 25 pages.

[13] Michael F. Deering, Stephen A. Schlapp, and Michael G. Lavelle. 1994. FBRAM: A new form of memory optimized for 3D graphics. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'94)*. ACM, 167–174.

[14] Yixiao Du, Yuwei Hu, Zhongchun Zhou, and Zhiru Zhang. 2022. High-performance sparse linear algebra on HBM-Equipped FPGAs using HLS: A case study on SpMV. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'22)*. 54–64.

[15] D. G. Elliott, W. M. Snelgrove, and M. Stumm. 1992. Computational ram: A memory-simd hybrid and its application to dsp. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 30.6.1–30.6.4.

[16] Ivan Fernandez, Ricardo Quislant, Eladio Gutiérrez, Oscar Plata, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, and Onur Mutlu. 2020. NATSA: A near-data processing accelerator for time series analysis. In *Proceedings of the IEEE 38th International Conference on Computer Design (ICCD'20)*. 120–129.

[17] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. 1–14.

[18] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. IEEE, 126–137.

[19] Kartikay Garg and Jeffrey Young. 2017. Evaluating hybrid memory cube infrastructure to support high-performance sparse algorithms. In *Proceedings of the International Symposium on Memory Systems*. ACM, 112–114.

[20] Anirban Ghose, Soumyajit Dey, Pabitra Mitra, and Mainak Chaudhuri. 2016. Divergence aware automated partitioning of OpenCL workloads. In *Proceedings of the 9th India Software Engineering Conference (ISEC'16)*. ACM, 131–135. https://doi.org/10.1145/2856636.2856639

[21] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu. 2018. Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions. Retrieved from http://arxiv.org/abs/1802.00320.

[22] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying complex workload-DRAM interactions: An experimental study. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 60 (Dec. 2019), 50 pages.

[23] M. Gokhale, B. Holmes, and K. Iobst. 1995. Processing in memory: The terasys massively parallel PIM array. *Computer* 28, 4 (1995), 23–31.

[24] Jan Gray. 2019. 2GRVI phalanx: A 1332-Core RISC-V RV64I processor ClusterArray with an HBM2 high-bandwidth memory system, and an OpenCL-like programming model, in a Xilinx VU37P FPGA (WIP report). In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis.*

[25] Bon-Geun Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)*. 153–165.

[26] Licheng Guo, Jason Lau, Yuze Chi, Jie Wang, Cody Hao Yu, Zhe Chen, Zhiru Zhang, and Jason Cong. 2020. Analysis and optimization of the implicit broadcasts in FPGA HLS to improve maximum frequency. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218718

[27] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. 982–995. https://doi.org/10.1109/ISCA45697.2020.00084

[28] Ramyad Hadidi, Bahar Asgari, Burhan Ahmad Mudassar, Saibal Mukhopadhyay, Sudhakar Yalamanchili, and Hyesoon Kim. 2017. Demystifying the characteristics of 3D-stacked memories: A case study for hybrid memory cube. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'17)*. 66–75.

[29] Ramyad Hadidi, Bahar Asgari, Jeffrey Young, Burhan Ahmad Mudassar, Kartikay Garg, Tushar Krishna, and Hyesoon Kim. 2018. Performance implications of NoCs on 3D-stacked memories: Insights from the hybrid memory cube. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'18)*. 99–108.

[30] P. Holzinger, D. Reiser, T. Hahn, and M. Reichenbach. 2021. Fast HBM access with FPGAs: Analysis, architectures, and applications. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'21)*. IEEE Computer Society, 152–159. https://doi.org/10.1109/IPDPSW52791.2021.00030

[31] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. 2016. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *Proceedings of the IEEE 34th International Conference on Computer Design (ICCD'16)*. 25–32.

[32] Yuwei Hu, Yixiao Du, Ecenur Ustun, and Zhiru Zhang. 2021. GraphLily: Accelerating graph linear algebra on HBM-Equipped FPGAs. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'21)*. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643582

[33] Hongjing Huang, Zeke Wang, Jie Zhang, Zhenhao He, Chao Wu, Jun Xiao, and Gustavo Alonso. 2021. Shuhai: A tool for benchmarking high-bandwidth memory on FPGAs. *IEEE Trans. Comput.* (2021), 1–1.

[34] Wenqin Huangfu, Krishna T. Malladi, Shuangchen Li, Peng Gu, and Yuan Xie. 2020. NEST: DIMM based near-data-processing accelerator for K-mer counting. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. 1–9.

[35] Gabriel H. Loh, Nuwan Jayasena, Mark H. Oskin, Mark Nutter, and Da Ignatowski. 2013. A processing-in-memory taxonomy and a case for studying fixed-function PIM. In *Workshop on Near-Data Processing (WoNDP)*. 1–4.

[36] Intel. 2020. Avalon interface specifications. Retrieved from https://www.intel.com/.

[37] Intel. 2020. High-bandwidth memory (HBM2) interface intel FPGA IP user guide. Retrieved from https://www.intel.com/.

[38] Zsolt István, David Sidler, and Gustavo Alonso. 2017. Caribou: Intelligent distributed storage. *Proc. VLDB Endow.* 10 (Aug. 2017), 1202–1213. https://doi.org/10.14778/3137628.3137632

[39] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Proceedings of the Symposium on VLSI Technology (VLSIT'12)*. 87–88.

[40] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B. Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. [n. d.]. MicroRec: Efficient recommendation inference by hardware and data structure solutions. Retrieved from https://arxiv.org/abs/2010.05894.

[41] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-scale recommendation inference on hybrid GPU-FPGA clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD'21)*.

[42] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12.

[43] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (high-bandwidth memory) DRAM technology and architecture. In *Proceedings of the IEEE International Memory Workshop (IMW'17)*. 1–4.

[44] Sang-Woo Jun. 2015. BlueDBM: An appliance for big data analytics. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*.

[45] Yi Kang, Wei Huang, Seung-Moon Yoo, D. Keen, Zhenzhou Ge, V. Lam, P. Pattnaik, and J. Torrellas. 1999. FlexRAM: Toward an advanced intelligent memory system. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*. 192–201.

[46] Kaan Kara, Dan Alistarh, Gustavo Alonso, Onur Mutlu, and Ce Zhang. 2017. FPGA-accelerated dense linear machine learning: A precision-convergence trade-off. In *Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. 160–167.

[47] Kaan Kara, Jana Giceva, and Gustavo Alonso. 2017. FPGA-based data partitioning. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'17)*. ACM, 433–445.

[48] Kaan Kara, Christoph Hagleitner, Dionysios Diamantopoulos, Dimitris Syrivelis, and Gustavo Alonso. 2020. High-bandwidth memory on FPGAs: A data analytics perspective. In *Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL'20)*. 1–8. ISSN: 1946–1488.

[49] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, KyungSoo Kim, Jin Jung, Ilkwon Yun, Sung Joo Park, Hyunsun Park, Joonho Song, Jeonghyeon Cho, Kyomin Sohn, Nam Sung Kim, and Hsien-Hsin S. Lee. 2022. Near-memory processing in action: Accelerating personalized recommendation with AxDIMM. *IEEE Micro* 42, 1 (2022), 116–127. https://doi.org/10.1109/MM.2021.3097700

[50] Jehandad Khan and Peter Athanas. [n. d.]. Creating custom network packet processing pipelines on HMC-enabled FPGAs. In *Proceedings of the Third Workshop on Networking and Programming Languages (NetPL)*.

[51] Jung-Sik Kim, Chi Sung Oh, Hocheol Lee, Donghyuk Lee, Hyong Ryol Hwang, Sooman Hwang, Byongwook Na, Joungwook Moon, Jin-Guk Kim, Hanna Park, Jang-Woo Ryu, Kiwon Park, Sang Kyu Kang, So-Young Kim, Hoyoung Kim, Jong-Min Bang, Hyunyoon Cho, Minsoo Jang, Cheolmin Han, Jung-Bae LeeLee, Joo Sun Choi, and Young-Hyun Jun. 2012. A 1.2 V 12.8 GB/s 2 Gb mobile wide-I/O DRAM with 4 × 128 I/Os using TSV based stacking. *IEEE J. Solid-State Circ.* 47, 1 (2012), 107–116.

[52] Peter M. Kogge. 1994. EXECUBE-A new architecture for scaleable MPPs. In *Proceedings of the International Conference on Parallel Processing*. Vol. 1, 77–84.

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

[54] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)*. 740–753.

[55] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, Jang Hwan Cho, Ki Hun Kwon, Min Jeong Kim, Jaejin Lee, Kun Woo Park, Byongtae Chung, and Sungjoo Hong. 2014. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'14)*. 432–433.

[56] Bing Li, Yu Hu, Ying Wang, Jing Ye, and Xiaowei Li. 2017. Power-utility-driven write management for MLC PCM. *ACM J. Emerg. Technol. Comput. Syst.* 13 (2017), 1–22.

[57] Bing Li and Bonan Yan. 2019. An overview of in-memory processing with emerging non-volatile memory for data-intensive applications. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'19)*. 381–386. https://doi.org/10.1145/3299874.3319452

[58] Bing Li, Bonan Yan, Chenchen Liu, and Hai Li. 2019. Build reliable and efficient neuromorphic design with memristor technology. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC'19)*. 224–229. https://doi.org/10.1145/3287624.3288744

[59] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. 288–301.

[60] Xiaoye S. Li and James W. Demmel. 2003. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.* 29, 2 (June 2003), 110–140.

[61] Chenhao Liu, Zhiyuan Shao, Zeke Wang, Kexin Li, Minkang Wu, Jiajie Chen, Xiaofei Liao, and Hai Jin. 2021. ScalaBFS: A scalable BFS accelerator on HBM-enhanced FPGAs. Retrieved from http://arxiv.org/abs/2105.11754.

[62] Zhiqiang Liu, Yong Dou, Jingfei Jiang, Qiang Wang, and Paul Chow. 2017. An FPGA-based processor for training convolutional neural networks. In *Proceedings of the International Conference on Field Programmable Technology (ICFPT'17)*. 207–210.

[63] Scott Lloyd and Maya Gokhale. 2015. In-memory data rearrangement for irregular, data-intensive computing. *Computer* 48, 8 (Aug. 2015), 18–25.

[64] Guixiang Ma, Yao Xiao, Theodore L. Willke, Nesreen K. Ahmed, Shahin Nazarian, and Paul Bogdan. 2020. A Vertex Cut Based Framework for Load Balancing and Parallelism Optimization in Multi-core Systems. Retrieved from https://arXiv:2010.04414.

[65] Alexander Matz, Mark Hummel, and Holger Fröning. 2016. Exploring LLVM infrastructure for simplified multi-GPU programming. In *MULTIPROG Workshop, Co-located with HiPEAC 2016*.

[66] Nathaniel Mcvicar, Chih-Ching Lin, and Scott Hauck. 2017. K-mer counting using bloom filters with an FPGA-attached HMC. In *Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. 203–210.

[67] Lifeng Nai, Ramyad Hadidi, He Xiao, Hyojong Kim, Jaewoong Sim, and Hyesoon Kim. 2018. CoolPIM: Thermal-aware source throttling for efficient PIM instruction offloading. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*. IEEE, Vancouver, BC, 680–689.

[68] Lifeng Nai, Yinglong Xia, Ilie G. Tanase, Hyesoon Kim, and Ching-Yung Lin. 2015. GraphBIG: Understanding graph computing in the context of industrial solutions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*. 1–12.

[69] Tomas Öhberg, August Ernstsson, and Christoph W. Kessler. 2019. Hybrid CPU–GPU execution support in the skeleton programming framework SkePU. *J. Supercomput.* 76 (2019), 5038–5056.

[70] Md Jubaer Hossain Pantho, Joel Mandebi Mbongue, Christophe Bobda, and David Andrews. 2018. Transparent acceleration of image processing kernels on FPGA-attached hybrid memory cube computers. In *Proceedings of the International Conference on Field-Programmable Technology (FPT'18)*. 342–345.

[71] Sang Phill Park, Sumeet Gupta, Niladri Mojumder, Anand Raghunathan, and Kaushik Roy. 2012. Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture. In *Proceedings of the Design Automation Conference (DAC'12)*. 492–497.

[72] Alberto Parravicini, Luca Giuseppe Cellamare, Marco Siracusa, and Marco Domenico Santambrogio. 2021. Scaling up HBM efficiency of top-K SpMV for approximate embedding similarity on FPGAs. Retrieved from http://arxiv.org/abs/2103.04808.

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

[74] J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *Proceedings of the IEEE Hot Chips 23 Symposium (HCS'11)*. 1–24.

[75] Morten B. Petersen, Stefan Nikolic, and Mirjana Stojilovic. 2021. NetCracker: A peek into the routing architecture of Xilinx 7-series FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'21)*, Lesley Shannon and Michael Adler (Eds.). ACM, 11–22. https://doi.org/10.1145/3431920.3439285

[76] PicoComputing. [n. d.]. Retrieved from https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9h7.

[77] PicoComputing. [n. d.]. Pico framework documentation. Retrieved from http://picocomputing.zendesk.com/hc/.

[78] Picocomputing. 2016. UltraScale-based SuperProcessor with hybrid memory cube. Retrieved from http://picocomputing.com/ac-510-superprocessor-module.

[79] PicoComputing. 2017. SC6-Mini. Retrieved from http://picocomputing.com/products/picocube/picomini/.

[80] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2015. A reconfigurable fabric for accelerating large-scale datacenter services. *IEEE Micro* 35, 3 (2015), 10–22.

[81] Abhi D. Rajagopala, Ron Sass, and Andrew G. Schmidt. 2019. Volcan: System integration of HLS and HMC on FPGAs. In *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig'19)*. 1–2.

[82] Daniel Rozhko, Geoffrey Elliott, Daniel Ly-Ma, Paul Chow, and Hans-Arno Jacobsen. 2017. Packet matching on FPGAs using HMC memory: Towards one million rules. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 201–206.

[83] Zhenyuan Ruan, Tong He, and Jason Cong. 2019. INSIDER: Designing in-storage computing system for emerging high-performance drive. In *Proceedings of the USENIX Annual Technical Conference*.

[84] Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso, and Sergio López-Buedo. 2019. Limago: An FPGA-based open-source 100 GbE TCP/IP stack. In *Proceedings of the 29th International Conference on Field Programmable Logic and Applications (FPL'19)*. 286–292.

[85] Paulo C. Santos, Geraldo F. Oliveira, Diego G. Tomé, Marco A. Z. Alves, Eduardo C. Almeida, and Luigi Carro. 2017. Operand size reconfiguration for big data processing in memory. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'17)*. 710–715.

[86] Juri Schmidt and Ulrich Bruning. 2015. openHMC—A configurable open-source hybrid memory cube controller. In *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig'15)*. 1–6.

[87] Juri Schmidt, Holger Fröning, and Ulrich Brüning. 2016. Exploring time and energy for complex accesses to a hybrid memory cube. In *Proceedings of the 2nd International Symposium on Memory Systems*. ACM, 142–150.

[88] Francesco Sgherzi, Alberto Parravicini, Marco Siracusa, and Marco Domenico Santambrogio. 2021. Solving large top-K graph eigenproblems with a memory and compute-optimized FPGA design. Retrieved from http://arxiv.org/abs/2103.10040. arXiv: 2103.10040.

[89] D. Sharma. 2019. Compute express link. *CXL Consortium White Paper.*

[90] Patrick Siegl, Rainer Buchty, and Mladen Berekovic. 2016. Data-centric computing frontiers: A survey on processing-in-memory. In *Proceedings of the Second International Symposium on Memory Systems (MEMSYS'16)*. ACM.

[91] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2019. Near-memory computing: Past, present, and future. *Microprocess. Microsyst.* 71 (2019).

[92] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gomez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal. 2020. NERO: A near high-bandwidth memory stencil accelerator for weather prediction modeling. In *Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL'20).* IEEE, 9–17.

[93] Linghao Song, Yuze Chi, Atefeh Sohrabizadeh, Young kyu Choi, Jason Lau, and Jason Cong. 2022. Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.*

[94] Harold S. Stone. 1970. A logic-in-memory computer. *IEEE Trans. Comput.* 19, 1 (Jan. 1970), 73–78.

[95] J. Stuecheli, W. J. Starke, J. D. Irish, L. B. Arimilli, D. Dreps, B. Blaner, C. Wollbrink, and B. Allison. 2018. IBM POWER9 opens up a new era of acceleration enablement: OpenCAPI. *IBM J. Res. Dev.* 62, 4/5 (2018), 8:1–8:8.

[96] Bharat Sukhwani, Thomas Roewer, Charles L. Haymes, Kyu-Hyoun Kim, Adam J. McPadden, Daniel M. Dreps, Dean Sanner, Jan Van Lunteren, and Sameh Asaad. 2017. ConTutto—A novel FPGA-based prototyping platform enabling innovation in the memory subsystem of a server class processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17).* 15–26.

[97] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17).* ACM, 65–74.

[98] Zeke Wang, Bingsheng He, and Wei Zhang. 2015. A study of data partitioning on OpenCL-based FPGAs. In *Proceedings of the 25th International Conference on Field Programmable Logic and Applications (FPL'15).* 1–8.

[99] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference (DAC'17).* 1–6.

[100] Louis Woods, Gustavo Alonso, and Jens Teubner. 2015. Parallelizing data processing on FPGAs with shifter lists. *ACM Trans. Reconfig. Technol. Syst.* 8, 2 (2015), 7:1–7:22. https://doi.org/10.1145/2629551

[101] Wm. A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News* 23, 1 (Mar. 1995).

[102] Xilinx. 2017. AXI interconnect v2.1. Retrieved from https://www.xilinx.com/support/documentation/ipdocumentation/axi_interconnect/v2_1/ pg059-axi-interconnect.pdf.

[103] Xilinx. 2019. Virtex UltraScale+ HBM FPGA: A revolutionary increase in memory performance. Retrieved from https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-hbm.html.

[104] Xilinx. 2020. Alveo U280 data center accelerator card user guide. Retrieved from https://www.xilinx.com/support/documentation/boards_and_kits/acceleratorcards/ug1314-u280-reconfig-accel.pdf.

[105] Xilinx. 2020. Alveo U50 data center accelerator card user guide. Retrieved from https://www.xilinx.com/support/documentation/boards_and_kits/acceleratorcards/ug1371-u50-reconfig-accel.pdf.

[106] Xilinx. 2020. AXI high-bandwidth memory controller v1.0. Retrieved from https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf.

[107] Xilinx. 2020. Vitis unified software platform. Retrieved from https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html.

[108] Yang Yang, Sanmukh R. Kuppannagari, and Viktor K. Prasanna. 2020. A high throughput parallel hash table accelerator on HBM-enabled FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'20).* IEEE, 148–153.

[109] Hanqing Zeng and Viktor Prasanna. 2020. GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 255–265.

[110] Jialiang Zhang and Jing Li. 2018. Degree-aware hybrid graph traversal on FPGA-HMC platform. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 229–238.

[111] Jialiang Zhang, Yue Zha, Nicholas Beckwith, Bangya Liu, and Jing Li. 2020. MEG: A RISCV-based system emulation infrastructure for near-data processing using FPGAs and high-bandwidth memory. *ACM Trans. Reconfig. Technol. Syst.* 13, 4 (Oct. 2020), 1–24.

[112] Yu Zhu, Zhenhao He, Wenqi Jiang, Kai Zeng, Jingren Zhou, and Gustavo Alonso. 2021. Distributed recommendation inference on FPGA clusters. In *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications (FPL'21).*

Received 7 February 2022; revised 25 April 2022; accepted 19 June 2022