Journal of Systems Architecture 129 (2022) 102561

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives

Biagio Peccerillo [a,*], Mirco Mannino [a], Andrea Mondelli [b], Sandro Bartolini [a]

[a] Department of Information Engineering and Mathematics, University of Siena, Italy
[b] Huawei Technologies Research & Development (UK) Ltd, Cambridge, UK

ARTICLE INFO

Keywords:
Accelerators
Domain-Specific Architectures
Survey
Taxonomy
Classification
Data-parallel
Machine Learning
PIM
CGRA
Open challenges
Future research directions

ABSTRACT

In recent years, the limits of the multicore approach emerged in the so-called "dark silicon" issue and diminishing returns of an ever-increasing core count. Hardware manufacturers, out of necessity, switched their focus to accelerators, a new paradigm that pursues specialization and heterogeneity over generality and homogeneity. They are special-purpose hardware structures separated from the CPU with aspects that exhibit a high degree of variability. We define a taxonomy based on fourteen of these aspects, grouped in four macro-categories: general aspects, host coupling, architecture, and software aspects. According to it, we categorize around 100 accelerators of the last decade from both industry and academia, and critically analyze emerging trends. We complete our discussion with throughput and efficiency figures. Then, we discuss some prominent open challenges that accelerators are facing, analyzing state-of-the-art solutions, and suggesting prospective research directions for the future.

1. Introduction

Around fifteen years ago, hardware manufacturers had to abandon the path set by Dennard scaling thirty years before due to growing leak currents, power dissipation, and wire delays [1,2]. Further improvements to the single-core architecture became increasingly harder to achieve, to the point that they were forced to turn their attention to new design paradigms.

The first broadly adopted solution was the so-called "multicore", which involves the coexistence of multiple cores on the same die [3]. Switching to a multicore design helped computer architects to continue taking advantage of transistor shrinking and their ever-increasing count per unit area, but the limits of this solution manifested soon:

- the difficulty in dissipating thermal power made the use of all the available cores together at full speed inconvenient [4,5];
- the diminishing returns of an increased core count depicted by Amdahl's law [6] sets an upper limit to the number of cores that can be fruitfully employed in most *practical* applications.

These factors together made clear that the multicore approach alone is not sufficient to satisfy the worldwide growing demand for computing power and performance.

A similar scenario emerged in HPC, in embedded and, in general, in energy-constrained contexts, due to the difficulty to keep on improving performance per Watt metrics of general-purpose processors [7]: the support of a wide range of workloads requires significant investment of micro-architecture/die-area/cache/energy and, due to the interaction between the effects of the end of Dennard scaling and wire-delay issues, decreases the potential computational density of general-purpose chips [4]. The limit becomes operations per die, and due to the unavoidable power-envelope constraints, it translates into limited performance per Watt.

A possible way to overcome these limits is to relax the general-purpose trait, so as to leverage the possibility to customize the hardware to perform specific tasks. For instance, this happened in the GPU domain, which steadily grew in the last decade: some non-functional structures (e.g., big caches) are reduced and the number of functional ones, directly involved in computations, are increased accordingly. This enabled an on-chip HW parallelism that nowadays surpasses that of general-purpose processors by as much as 2 orders of magnitude (e.g., thousands of cores in high-end GPUs vs few tens of cores). Today, many hardware manufacturers are following this path and adopting

* Corresponding author.
E-mail address: peccerillo@diism.unisi.it (B. Peccerillo).

https://doi.org/10.1016/j.sysarc.2022.102561
Received 9 April 2022; Accepted 12 May 2022
Available online 24 May 2022
1383-7621/© 2022 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

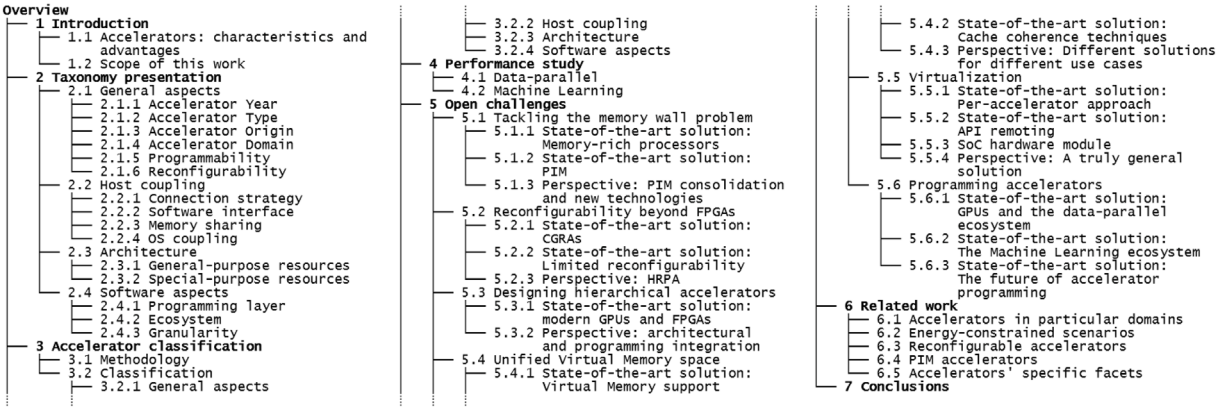


Fig. 1. Overview of the survey.

these design principles, investing in what are called Domain-Specific Architectures (DSAs) or *accelerators* [8].

1.1. Accelerators: characteristics and advantages

An accelerator can be defined as "a separate architectural substructure [...] that is architected using a different set of objectives than the base processor, where these objectives are derived from the needs of a special class of applications" [9]. It is not meant as a replacement for the base processor: a CPU in the system is still necessary to execute O.S. tasks and orchestrate execution on the accelerator itself. In principle, the CPU is sufficient to execute also the workloads demanded to the accelerator, generally called the "accelerated workloads" [10]. The reason why the latter is preferred lays in the promised improvement of non-functional requirements such as compute throughput and/or low latency and/or energy efficiency. This, above all, is the main reason why accelerators are such a hot topic today.

Accelerators are not new in the computing landscape. Historically, they have been used for various tasks, but their adoption remained limited. Even the most promising accelerator became obsolete in a few years and was surpassed by the growing performance of general-purpose processors, thus making the investment "not worth it" [9,11]. This is no longer true: on the contrary, they are regarded as the most promising driving force of computer architecture [4], the only viable solution to satisfy the expanding computing needs of both private users and companies. Summarizing the words by Hennessy and Patterson [4] and Dally et al. [12], the real value of accelerators lies in:

- More efficient and specialized forms of parallelism;
- More effective use of the memory hierarchy, with high-bandwidth, low-cost, low-energy local/optimized memories;
- Data specialization, to do in one cycle what would need several cycles on a CPU;
- Reduced fetch and decode overhead, due to hardware specialization;
- Support for Domain Specific Languages (DSLs), to leverage the underlying architecture more efficiently.

Since accelerators are specialized hardware, finely tuned to the specific needs of a class of applications, they are not bound to the design constraints that are a property of general-purpose hardware. Computer architects may explore countless techniques that would not pay off in the *general* case, but could fit the *special* case, providing measurable advantages.

For instance, it is well-known that CPUs evolved to leverage an ever-increasing instruction-level parallelism (ILP), resorting to the replication of execution-units (superscalar paradigm) and pipelines with a growing number of stages. With them, area- and power-hungry features such as out-of-order execution and complex branch prediction became fundamental to minimize performance penalties associated with pipeline stalls and flushes [3]. Not to tell the enormous amount of caching structures that nowadays general-purpose processors employ, trying to exploit locality and keeping the cores steadily crunching instructions in the average cases. On the other hand, GPUs, the most widespread accelerators in the market, aim at compute-intensive workloads where many arithmetic instructions are performed on big sets of data, with high degrees of data-parallelism. Under the load of these workloads, the die-area and complexity investments in sophisticated mechanisms like out-of-order execution and branch prediction pay off far less than dedicating such area to additional units directly involved in the computations. Moreover, such a simpler design drastically reduces the energy per operation compared to general-purpose architectures [13].

1.2. Scope of this work

Thanks to their special-purpose nature, accelerators can be designed more freely than general-purpose processors. By relaxing the *generality* trait, original, even *exotic* design choices can be exploited as long as they prove advantageous in the reduced domain for which the accelerator is intended. These choices are highly dependent on the characteristics of the accelerated work, its algorithmic features, the required form factor, etc. This variety affects not only architecture, but also the software ecosystems of languages, compilers, libraries, and frameworks that revolve around accelerators.

Due to this intrinsic heterogeneity, it is not easy to grasp the plethora of aspects that characterize accelerators: the domain in which they operate, their hardware components, whether they are programmable or not, the granularity at which they operate, the characteristics and level of maturity of their software ecosystem, the way they are connected to the host system, their relation with the host O.S., etc. These aspects are often intertwined, to the point that a comprehensive analysis of accelerators should discuss them together, from a holistic point of view.

This survey discusses accelerators in such spirit, considering several aspects that characterize them. First, we provide a multi-level taxonomy, organized in four macro-categories: general aspects, host coupling, architecture, and software aspects, and according to it we classify numerous accelerators from the last decade. The value of this classification is twofold: on the one hand, we help the readers to get synthetic information about each accelerator quickly and position it in this multi-dimensional design space; on the other hand, we give them a comprehensive overview of the *accelerator world* as a whole and highlight emerging trends. We enrich the discussion with throughput and efficiency figures for data-parallel and ML-oriented accelerators, and then discuss our findings in relation to the classification done. Finally, we discuss some open challenges that accelerators are facing,

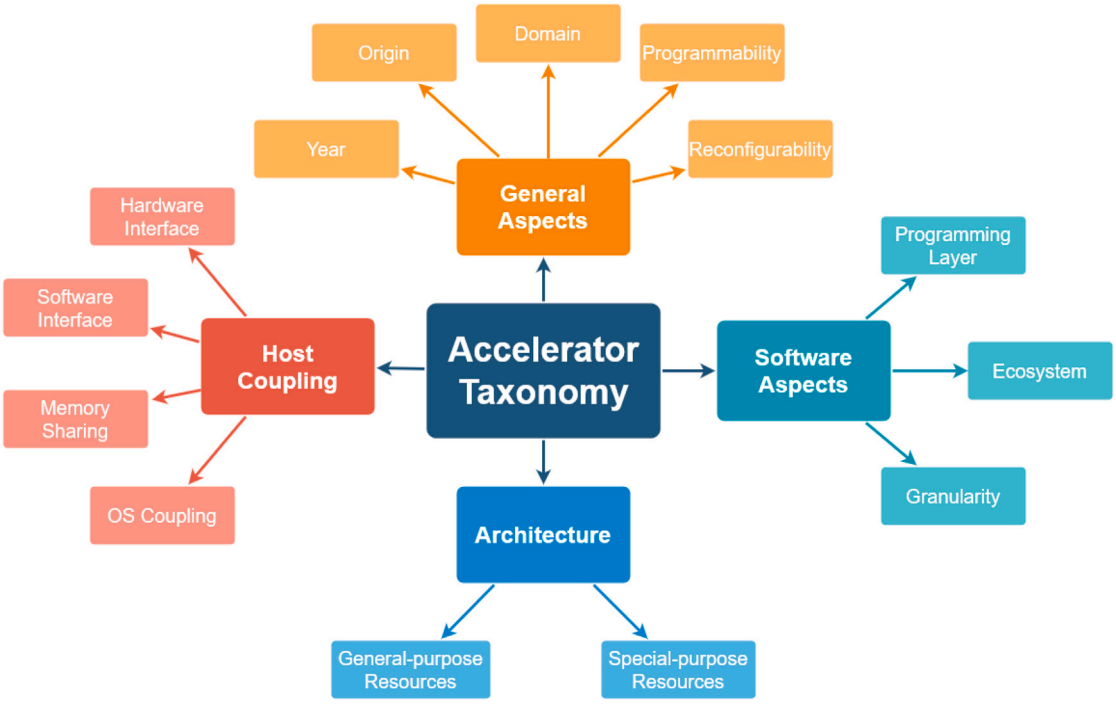


Fig. 2. A visual representation of the proposed taxonomy, with the classification criteria grouped in four macro-categories.

analyze some proposed state-of-the-art solutions, and suggest promising research directions as future perspectives.

In summary, the main contributions of this work are:

- A complete, multi-level taxonomy of accelerators organized in four macro-categories: general aspects, host coupling, architecture, and software aspects.
- A categorization of several accelerators from industry and academia and a comprehensive discussion of emerging trends.
- A comparison of throughput and efficiency figures of many data-parallel and ML-oriented accelerator models.
- An in-depth analysis of open challenges faced by accelerators as a whole, state-of-the-art solutions to them, and prospective directions for future research.

The remainder of this paper is organized as follows: in Section 2, we present our taxonomy, in Section 3, we use it to do a thorough classification of several accelerators and discuss the trends that emerge from it. In Section 4, we discuss throughput and efficiency figures. In Section 5, we discuss challenges, state-of-the-art solutions, and prospective research directions. In Section 6, we present some studies that share our view and focus on accelerators in general. We conclude in Section 7. Table 1 lists acronyms and abbreviations used throughout the paper. Fig. 1 provides an overview of the paper.

2. Taxonomy presentation

In order to describe accelerators comprehensively, we propose fourteen criteria and group them into four macro-categories. Fig. 2 gives a graphic representation of them.

In this section, we thoroughly discuss the classification criteria and present the possible values for each of them. They are descriptive of the variety of accelerators we have found and categorized, but they are not intended as a complete coverage of the design space. Unavoidably, some possibilities are not listed because they are not representative of the accelerators we took into consideration. Others have been omitted on purpose because they are too specific of a single model and would lead to a fragmentation of the relevant information. In the following, we acknowledge some exclusions for the sake of completeness.



Fig. 3. The General Aspects macro-category.

2.1. General aspects

This macro-category, shown in Fig. 3, includes aspects that describe an accelerator with a very high level of abstraction and permit to quickly contextualize it.

2.1.1. Accelerator year

The accelerator's launch year, where available, otherwise its announcement year. For academic projects, it is usually the year of publication of the first paper describing the accelerator. In this work, we restrict the scope to the last decade: 2013–2022.

Table 1
Acronyms and abbreviations used throughout the paper.

AI	Artificial Intelligence
ADC	Analog-to-Digital Converter
ANN	Artificial Neural Network
APU	Application Processing Unit
AR	Augmented Reality
ASIC	Application-Specific Integrated Circuit
BNN	Binarized Neural Network
CCIX	Cache Coherent Interconnect for Accelerators
CGRA	Coarse-Grained Reconfigurable Architecture
CNN	Convolutional Neural Network
CXL	Compute Express Link
DAC	Digital-to-Analog Converter
DDK	Device Development Kit
DDR	Double Data Rate
DFG	Data-Flow Graph
DIMM	Dual Inline Memory Module
DLRM	Deep Learning Recommendation Model
DNN	Deep Neural Network
DSA	Domain-Specific Architecture
DSL	Domain-Specific Language
DSP	Digital Signal Processing
eDRAM	Embedded DRAM
FLOPS	Floating-Point Operations per Second
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GAN	Generative Adversarial Network
GEMM	GEneral Matrix–Matrix multiplication
GPGPU	General Purpose computing on GPU
GPU	Graphics Processing Unit
HBM	High-Bandwidth Memory
HDL	Hardware Description Language
HLS	High-Level Synthesis
HMC	Hybrid Memory Cube
HPC	High-Performance Computing
IDE	Integrated Development Environment
IoT	Internet of Things
IOMMU	I/O Memory Management Unit
IOTLB	I/O Translation Lookaside Buffer
JIT	Just-In-Time
k-NN	k-Nearest Neighbors
LBM	Lattice-Boltzmann Method
LLC	Last-Level Cache
LPDDR	Low-Power Double Data Rate
LSTM	Long Short-Term Memory
MAC	Multiply-Accumulate
MCDRAM	Multi-Channel DRAM
MIMD	Multiple Instruction Multiple Data
ML	Machine Learning
MLP	Multi-Layer Perceptron
MMIO	Memory-Mapped I/O
NFA	Nondeterministic Finite Automata
NIC	Network Interface Controller
NNP	Neural Network Processor
NoC	Network-on-Chip
NPU	Neural Processing Unit
NRE	Non-Recurring Engineering
OAM	Open Accelerator Module
OCP	Open Compute Project
OPS	Operations per Second
PE	Processing Engine
PIM	Processing-In-Memory
PoD	Point of Delivery
RNN	Recurrent Neural Network
RTL	Register Transfer Level
SFU	Special Function Unit
SIMD	Single Instruction Multiple Data
SPE	Synergistic Processing Element
STT-MRAM	Spin-Transfer Torque Magnetic RAM
SVM	Support Vector Machine
TDP	Thermal Design Power
TLB	Translation Lookaside Buffer
TPU	Tensor Processing Unit
UVM	Unified Virtual Memory
VLIW	Very-Long Instruction Word
VM	Virtual Machine
WGA	Whole Genome Alignment

(*continued on next page*)

Table 1 (*continued*).

2.1.2. Accelerator type

It synthetically specifies the accelerator's *type*, intended as its broadest classification from an architectural point of view. We identify the following types:

ASIC chip designed to perform a specific application. As such, they offer the most efficient solution, but it comes at the cost of no reconfigurability, no programmability, and high NRE costs.

FPGA reconfigurable accelerators with a large number of logic blocks (up to millions [14]), memory cells, and specialized blocks (e.g., DSP, I/O, etc.) interconnected. Both blocks and interconnect allow a fine-grain reconfigurability that supports a high variety of applications, making them suitable for general-purpose computing [15,16]. Reconfiguration time is generally high, in the order of ms-s [17,18], as it involves the writing of configuration data into a non-volatile memory (generally an off-chip Flash memory), which is later used to feed on-chip memory cells (generally SRAM cells) when the FPGA is powered on [17,19].

Spatial spatial accelerators feature an array of relatively simple ALU-like PEs that can communicate directly through NoCs, buses or ad-hoc inter-PE connections [20]. These PEs can form a chain and exchange data each other in a dataflow fashion. Each PE can have its own control logic and memory [21].

CGRA special case of spatial accelerators, organized as arrays of reconfigurable PEs connected through a reconfigurable interconnect. They offer a *coarse-grain* reconfigurability, which makes them *domain-specific* flexible. Their reconfiguration time is generally faster than FPGAs', usually in the order of ns-μs, making them eligible for temporal computations as well as spatial computations. They support both configuration-driven and dataflow-driven execution [18].

Systolic in systolic accelerators, the main computation is performed by a 1D or 2D array of arithmetic units controlled in lockstep. Input data enter the array in the first level, each arithmetic unit applies a fixed function to its input and passes its output to arithmetic units in the following level, considered downstream. Each level produces a partial result which flows through the array in a wave-like fashion until the last level, which produces the total result as output [7]. Systolic accelerators are also a subset of spatial ones, with two main restrictions: ALUs are not programmable, but perform a *fixed function* on the input, and data flow in a *fixed direction* during elaboration (i.e., downstream).

Vector accelerators where the computation is performed by vector processors, each with one or more vector cores. Each vector core has vector registers that can hold tens or hundreds of primitive elements (e.g., 32-bit floating point values) and apply single instructions to all the elements in the vector register. They rely heavily on pipelining and have the ability to load/store from/to the memory both contiguous and strided data [7].

Manycore accelerators featuring many hundreds or even thousands of independent physical cores, sometimes grouped in tens of multi-processors. The manycore paradigm can be regarded as the natural evolution of the multi-core paradigm, with an even higher degree of parallelism that is particularly suited for data-parallel workloads. For the cores, a simpler design is preferred (in-order, no branch prediction, etc.) to pack them more densely in the chip area and leverage as much purely computational resources as possible.

GPU GPUs are the most notable instances of manycore accelerators, but they have also similarities with vector accelerators. For their importance in today's market and for their specific architectural features, we employ a distinct category for them within the *manycore* umbrella. They feature up to tens of multi-processors made up of simple, in-order cores that exploit massive thread-level parallelism [22]. These are similar to independent MIMD cores, with each multi-processor acting as a multi-threaded vector core. The main difference with *classic* vector cores lays, in fact, in the massive employment of multi-threading rather than deep pipelines, with threads mapped onto physical lanes [7].

PIM this type of accelerator brings computation closer to memory, in order to reduce the energy needed to move data and increase memory bandwidth. The term refers to accelerators providing both *near-memory* and *in-memory* computing. In the former case, compute engines are placed near to classic DRAM or SRAM cells, which retain their role unaltered. In the latter, memory cells are re-purposed to perform computing logic on stored data, also with the help of emerging memory technologies such as ReRAM and STT-MRAM that expose desirable characteristics in this sense [23].

A note is needed for the class of Spatial Accelerators. Although it is a super-set of FPGAs, CGRAs, and Systolic Accelerators, in this paper, we label as Spatial Accelerators only those that cannot be described as any of the three. This way, we always prefer the most specific definition available, using FPGA, CGRA, or Systolic where fitting and reserving Spatial only to those accelerators with interconnected non-reconfigurable PEs with flexible, often programmable data flow direction. We adopt a categorization similar to that expressed in [24].

We use the same approach for Manycore and GPUs: although the latter is a special case of the former, we use the GPU terms where appropriate and Manycore for accelerators that are not GPUs.

2.1.3. Accelerator origin

Whether the accelerator originates from Industry or Academia. Industrial accelerators are commercial products. They can be made available on the market to end-users by being sold individually or as computing resources in data-centers. Academic accelerators, conversely, are usually prototypes described in scientific journals and conference proceedings.

2.1.4. Accelerator domain

It refers to the class of applications that can be executed on the accelerator. Thus, by specifying that an accelerator operates in/pertains to a particular domain, we intend that said accelerator can execute (or *accelerate*) applications pertaining to that domain. We identify eighteen domains. These are not mutually exclusive, as there are overlaps. In this sense, "Data-parallel", "Dataflow", and "General-purpose" are *broad domains*, and the others are *narrow domains*, as highlighted in Tables 5. The former are *computation paradigms*, and as such they are more ample. They may include several applications from the latter, which can be also described as *application areas*. Consequently, accelerators capable of targeting broad domains present a high flexibility, and are certainly able to execute workloads from one or more narrow ones. For such accelerators, we specify the broad domain to which they belong, and also some narrow ones that include the most notable applications executed on that accelerator.

Broad domains, or computation paradigms:

Data-parallel computation paradigm in which problems can be expressed in terms of collections of elements that can be processed concurrently, by applying the same stream of instructions to each data element [25].

Dataflow computation paradigm in which problems are expressed in terms of processing nodes and data flowing between them, usually by the means of a *DataFlow Graph* (DFG) [26–28]. The operations are applied to the operands as soon as they are available on the processing node.

General-purpose execution of a program with no particular structure, constraints, or application domain.

Narrow domains, or application areas:

ML training execution of a machine learning model with the aim of establishing the value of the model parameters [21].

ML inference execution of a machine learning model whose parameters are already established in a previous training phase [21].

AR real time direct or indirect view of the physical real world environment that has been enhanced by the addition of virtual computer-generated information [29].

Computer Vision field of the AI that aims to build autonomous systems that could perform some tasks which the human visual system can perform, like image/video analysis and interpretation [30].

Cryptography field of computer science that aims to transform messages in order to make the message unintelligible to all but the intended receiver of the message [31].

Data (De)Compression field of computer science that studies algorithms that aim to decrease the number of bytes used to represent data, while preserving (lossless) or reducing (lossy) their quality.

Database Manipulation manipulation of information stored in a database system, intended as an organized collection of structured information, typically stored electronically in a computer system [32].

Genome Sequence Alignment alignment of sequences of nucleotide bases, read from a DNA molecule with a "genome sequencing" operation [33].

Graph Processing manipulation of data organized in a graph structure.

Graphics field of computer science that aims to communicate visually through a computer display and its interaction devices [34].

Multimedia processing of streams of data of multiple content forms, such as audio, video, speech, text, etc.

NFA computation of finite state automata in which more than one transition is possible from one state [35].

Network Flow Encryption set of operations used to encrypt data flowing in input or output through the network card [36].

Pattern Matching set of operations used to test an expression in order to check the presence/absence of a given pattern.

Search Acceleration set of operations used to deliver relevant web documents to users with consistently low response times [37].

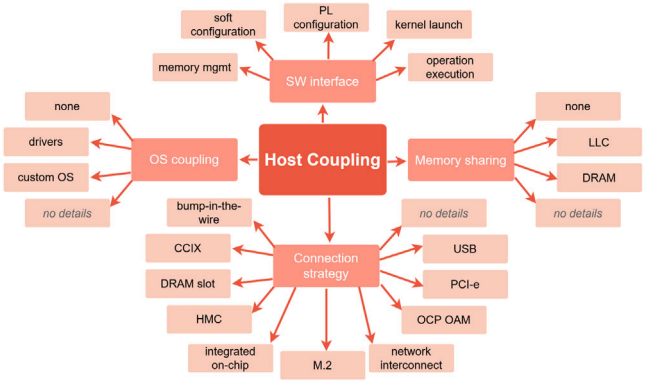


Fig. 4. The Host Coupling macro-category.

The presence of a "General-purpose" domain seems in contrast with the *special-purpose* nature of accelerators. However, it takes into account those devices that are used as accelerators, i.e., connected to a host system where a general-purpose processor executes an O.S. and regulates execution on the device, but have no constraints on the kind of applications supported. For instance, it is the case of FPGAs, as they have the ability to model even a general-purpose CPU [15,16].

Other domains exist besides those listed above. Some examples can be found in [38–45]. However, in this work we aim to pragmatically focus and cover the vast majority of domains as to allow discussing a solid classification of existing accelerators and their features. The remaining accelerator domains could reasonably fit in some listed domains in any case or very close to them.

2.1.5. Programmability

Every accelerator gives the possibility to perform some computations on user-provided input data to produce output data and return them back to the user. To what extent these computations can be customized characterizes the accelerator's *programmability*: if users are limited to trigger a predefined operation on an input stream, we say that the accelerator is not programmable. Conversely, we say that an accelerator is programmable if it has the ability to interpret instructions, usually provided as binary code, and perform different operations based on them.

2.1.6. Reconfigurability

An accelerator is classified as reconfigurable if there exists a software-based procedure to alter the logic functions performed by its hardware. E.g., FPGAs and CGRAs, where software techniques can be used to change the operations performed by the logic elements and the interconnection between them. Reconfigurable accelerators are always equipped with various blocks of *programmable logic*, (re)configured by users after manufacturing. Reconfiguration is generally achieved by writing a bit-stream of configuration data into a configuration memory, which is then used to feed several on-chip memory cells that are responsible for regulating the behavior of single components.

2.2. Host coupling

In this category, shown in Fig. 4, we put all those aspects that describe how the accelerator is connected to the rest of the system, the so-called *host*.

2.2.1. Connection strategy

It is the way the accelerator is physically connected to the host. We identify the following possibilities, spanning from the tighter and close-range couplings up to the more loose and distant ones:

bump-in-the-wire special placement between two existing components, achieved by connecting input/output wires of the components to those of the accelerator.

CCIX cache-coherent interconnect expressed as a set of specifications defined by the CCIX Consortium. It provides a non-proprietary standard to ensure chip-to-chip communication between different devices in heterogeneous systems, with high bandwidth and low latency [46,47]. It enables the expansion of the OS paged memory with the memory of the accelerator, creating a cache-coherent virtual memory space.

DRAM slot standard motherboard slot where RAM modules are inserted. Accelerators using this connection are designed as memory DIMMs with integrated in-memory computing units.

HMC high-performance single-package containing four or eight DRAM dies stacked together with a computing logic layer [48,49]. It can be connected to the host in various ways, e.g., DIMM slots or Intel QuickPath Interconnect [50].

integrated on-chip the accelerator is part of a bigger SoC that includes at least a general-purpose CPU.

M.2 flexible specification for external devices that supports PCIe, SATA, and USB buses.

OCP OAM flexible high-speed interconnect based on a mezzanine form factor, with a focus on interoperability between different components [51].

PCI-e high-speed expansion bus, the most common motherboard interface for external devices.

network interconnect standard physical connection for network devices. Accelerators using this connection are complex devices, generally designed as PoDs.

USB most common industry standard to connect external peripherals.

no details no further details about the accelerator connection are disclosed.

2.2.2. Software interface

It describes the set of operations that can be performed in code in order to interact with the accelerator. The perspective is that of the application developer: in order to write *accelerated* code, they need to perform some operations to setup and start execution on the accelerator. We identify the following possible operations:

Memory management accelerator memory allocation/freeing and data movement from/to the host memory to/from the accelerator memory.

Soft configuration software configuration of the accelerated task. It may involve a parameter writing that tunes a pre-defined operation or the loading into the accelerator of a binary representing the operation.

Programmable Logic configuration software-based procedure to configure the programmable logic of a reconfigurable device, generally expressed in RTL/HDL.

Kernel launch invocation and execution of a user-defined *kernel* function on the accelerator.

Operation execution single operation execution, generally achieved through a command packet sending.

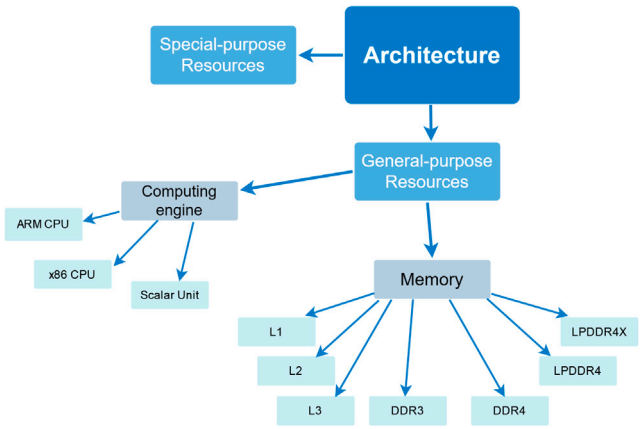


Fig. 5. The Architecture macro-category.

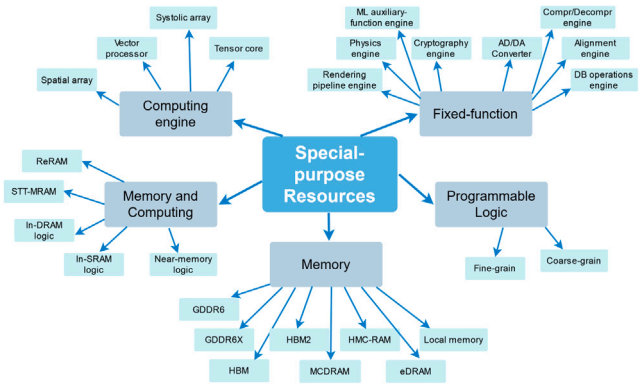


Fig. 6. The Special-purpose Resources category, from Architecture macro-category.

2.2.3. Memory sharing

It specifies whether the accelerator shares any level of the memory hierarchy with the host, and can access that memory level with no intervention from the CPU. It refers to contexts in which the accelerator acts as a node accessing the memory on-par with the CPU, and it may even be involved in coherence protocols. We identify the following possibilities:

none accelerator and host do not share any level of the memory hierarchy;

LLC accelerator has direct access to the host's LLC;

DRAM accelerator has direct access to the host's DRAM;

no details no further details about the memory sharing are disclosed.

2.2.4. OS coupling

Some accelerators need to be recognized by the Operating System to properly work, while others can be simply connected to the system and work transparently. In the first case, OS responsibilities may vary, as it may be responsible for providing to the accelerator handles to particular resources, or signaling the accelerator in response of specific events, or even managing its internal state between context switches. Overall, we identify the following possible OS coupling strategies:

none the accelerator can interact with the system transparently, no OS awareness is needed;

drivers user-space and/or kernel-space drivers need to be installed;

custom OS the accelerator needs a custom OS installed on the host in order to work properly;

no details no further details about the OS coupling are disclosed.

2.3. Architecture

As specified previously, the accelerators' *Architecture* can offer a remarkable diversity. To better highlight analogies and differences with general-purpose systems, we divide the hardware components in two categories, one for the analogies and one for the differences. Fig. 5 shows this macro-category, and Fig. 6 highlights the Special-purpose Resources.

2.3.1. General-purpose resources

They are hardware components that can be normally found in a general-purpose system. We group them into Memories and Computing engines. We identify the following:

Memory:

- L1, L2, L3: cache levels;
- DDR3, DDR4: DRAM memories;
- LPDDR4, LPDDR4X: low-power DRAM memories.

Computing engine:

- ARM CPU: ARM-based general-purpose processor;
- x86 CPU: x86-based general-purpose processor;
- Scalar unit: simple general-purpose processor, usually lightweight (e.g., in-order, single-issue) with unspecified ISA.

Other memory resources exist, but they have been replaced by more recent technologies (e.g., DDR2, LPDDR2, LPDDR3) or are not used yet (e.g., DDR5). In both cases, they are not present in the accelerators we classified. As for computing engines, we distinguish between ARM, x86 and other/unspecified ISAs because the first two are notable examples that characterize today's general-purpose systems. We put in the general *scalar unit* category other possibilities, like RISC-V, MIPS, etc.

2.3.2. Special-purpose resources

The accelerators' peculiar hardware components are put under this category. It includes those resources that have been designed to explicitly fit the needs of the accelerated applications, following principles that are not those adopted for general-purpose components. We group them further into Memories, Computing engines, Fixed-function components, and Memory & Computing engines to indicate those components that provide both memory and computing functions. We identify the following:

Memory:

- GDDR6, GDDR6X graphics-oriented DRAM technologies;
- HBM, HBM2, MCDRAM, HMC-RAM: high speed, 3D-stacked DRAM memory technologies with through-silicon vias and micro-bumps [48,49,52,53];
- eDRAM: embedded DRAM, integrated on the same die of the accelerator;
- Local memory: on-chip SRAM memory; both addressable, with content explicitly managed software-side (*scratchpad*), and non-addressable, transparently used to store partial results, configuration parameters, or instructions.

Computing engine:

- Spatial array: 1D or 2D array of interconnected PEs/arithmetic units that process data and exchange them independently of each other;

- Vector processor: processor that executes single instructions on arrays of data, which includes both state-of-the-art vector processors and SIMD processors [7];
- Systolic array: 1D or 2D array of PEs/arithmetic units that process data in lockstep and pass them downstream;
- Tensor core: single core with a 2D or 3D array of arithmetic units, generally specialized in MAC operations.

Fixed-function:

- Rendering pipeline engine: pipeline including all the required operations to render 3D models to 2D screen;
- Physics engine: computational unit specialized in physical behavior emulation;
- ML auxiliary-functions engine: computational unit specialized in auxiliary ML operations (e.g., activation functions);
- Cryptography engine: computational unit specialized in cryptographic operations (e.g., hashing, cipher/decipher);
- AD/DA converter: analog-to-digital/digital-to-analog converter;
- Compr./Decompr. engine: computational unit specialized in data compression and decompression;
- Alignment engine: computational unit specialized in genomics-related alignment operations;
- DB Operations engine: computational unit specialized in common database manipulation operations.

Programmable Logic:

- Fine-grain: permits general-purpose flexibility, with highly customizable hardware and interconnections between components. It is typical of FPGAs.
- Coarse-grain: the overall structure is fixed and dictated by domain-specific flexibility. It is typical of CGRAs, that present a 2D array of reconfigurable PEs with a reconfigurable interconnect, usually less complex than FPGAs' interconnect [18].

Memory & Computing:

- ReRAM: non-volatile memory that exploits the change of its resistance to store data [54];
- STT-MRAM: non-volatile memory that uses magnetic tunnel junctions whose magnetic layers can change value by spin-polarized currents [55];
- In-DRAM logic: computing units placed within DRAM memory;
- in-SRAM logic: computing units placed within SRAM memory;
- Near memory logic: computing units placed close to, but not within, memory.

Also in this case, we omit some memory resources that have been replaced by more recent technologies, like GDDR5 and previous iterations, because they are not used in the categorized accelerators. We merge software-controlled local memories, often called *scratchpads*, and buffer-like memories into the single category "local memory": although it is interesting to distinguish between the two, we usually have not enough information to tell how a local memory is managed. As for computing engines/fixed-function, we omit some possibilities like specialized controllers and other IP blocks that perform very specific functions.

2.4. Software aspects

The *Software Aspects* macro-category, shown in Fig. 7, summarizes the software characteristics of the accelerators.

2.4.1. Programming layer

It specifies the tools programmers can use to program, configure, or communicate with the accelerator. These are representative of the abstraction level which programmers can leverage and, consequently, how productively they can write accelerated code. We identify the following possibilities, sorted by increasing level of abstraction:

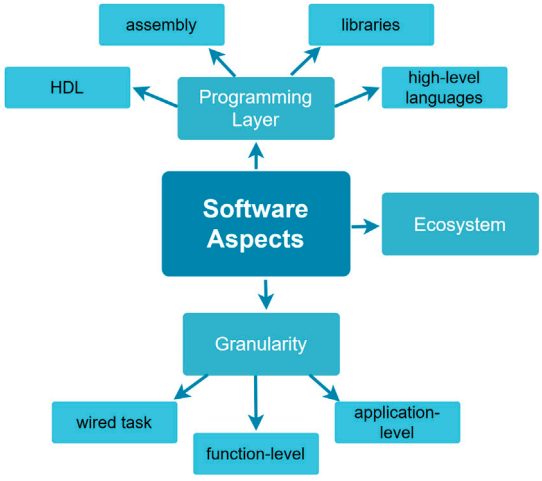


Fig. 7. The Software Aspects macro-category.

HDL used to describe digital circuits, it can be employed to (re)configure the programmable logic of *reconfigurable* accelerators, as previously defined;

assembly accelerator's equivalent to general-purpose processor's assembly language;

libraries collections of functions and classes that invoke accelerated code under the hood, they can be used in high-level source-code to gain access to the accelerator;

high-level languages accelerator's equivalent to general-purpose processor's programming languages, they permit generating accelerator code by compiling/interpreting high-level source code.

2.4.2. Ecosystem

This category is closely related to the previous one. It lists the *particular* frameworks, compilers, libraries, etc. available to programmers and end-users that intend to leverage the accelerator. There are both open-source (e.g., [56–64]) and vendor-provided examples (e.g., [65–73]).

2.4.3. Granularity

It describes the kind of task that is demanded to the accelerator. It is tightly related to the level of flexibility supported. We identify three of them, sorted by increasing complexity:

Wired task single operation wired into the accelerator hardware. Generally, it can be "activated" with a command packet.

Function-level user-defined function, as intended in the context of C, C++ and similar high-level programming languages. It is not necessarily invoked as-is, but can be part of a more complex execution flow: e.g., it may be the activation function of a neural network's neurons, a function performed by each node of a graph on its input data, etc.

Application-level a full application can be executed on the accelerator.

3. Accelerator classification

In this section, we classify several accelerators according to the criteria presented in the previous section.

Table 2
List of references of accelerator families used to fill the classification tables. For each, the models considered for performance figures are shown.

Accelerator family	Models	References
AMD Radeon RX 5000 Series	Radeon RX 5300, 5600 XT, 5700 XT	[74–77]
AMD Radeon RX 6000 Series	Radeon RX 6800 XT, 6900 XT	[78–80]
ARM Mali Bifrost		[81–85]
ARM Mali Valhall		[81,86,87]
Intel FPGA 10 Series	Intel Stratix 10NX[a]	[14,88–97]
Intel FPGA F-Series		[98–100]
Intel FPGA V Series		[101–104]
Intel Graphics Technology (Gen11)	UHD Graphics G1, Iris Plus Graphics G4, G7	[105–108]
Intel Graphics Technology (Xe-LP)	UHD Graphics 730, 770, Iris Xe Graphics G7	[109–112]
Intel Nervana NNP-I 1100, 1300	NNP-I 1100, 1300	[113–117]
Intel Nervana NNP-T 1300, 1400	NNP-T 1300, 1400	[115,118–121]
Intel Xeon Phi Knights Landing	Xeon Phi KL7210, KL7250, KL7290	[122–126]
Intel Xeon Phi Knights Mill	Xeon Phi KM7235, KM7285, KM7295	[125,127–129]
NEC SX-Aurora TSUBASA Vector Engine Type10	Type 10A, 10B, 10C	[130–133]
NEC SX-Aurora TSUBASA Vector Engine Type20	Type 20A, 20B	[130,131,133,134]
NVIDIA GeForce RTX 20xx	GeForce RTX 2060, 2060T,[b] 2080, 2080T,[b] Titan, TitanT[b]	[135–137]
NVIDIA GeForce RTX 30xx	GeForce RTX 3060, 3060T,[b] 3070Ti, 3070TiT,[b] 3090, 3090T[b]	[138–141]
Xilinx FPGA 7 Series		[142–146]
Xilinx FPGA Ultrascale+ Series		[147–151]
Xilinx Versal ACAP		[152–154]

[a]AI-optimized blocks.
[b]Tensor cores.

3.1. Methodology

Many accelerators have been proposed during the years, probably starting with the Intel 8087 Math Coprocessor [289], announced in 1980 for PCs. A treatise comprehending all the accelerators proposed in both industry and academia during more than 40 years is outside the scope of our work. We are interested in depicting the current situation of the accelerator world, in order to provide a useful guide to professionals with different backgrounds, as stated in the Introduction. Also, we are interested in identifying trends and determining successful features and open challenges to complete the "big picture". In order to do so, we also need to take into consideration *recent* accelerators. We analyze accelerators proposed in the last 10 years (i.e., not before 2013), knowing that some fundamental ones that had a big impact in the computing world are not included, like the aforementioned 8087 (1980), the Cell B.E. SPE [290] (2005), or the GeForce 256 (1999), that was dubbed "the world's first GPU" [291].

Due to the level of detail we adopt in this paper, we must necessarily analyze accelerator *models* or, when enough traits are in common, *families*. For instance, we cannot categorize "GPUs", as the term is too broad for our purposes. On the opposite side, a categorization that distinguishes between single models is often not necessary, as we can safely describe the family they belong to, highlighting differences and factorizing analogies. Thus, we categorize e.g., GeForce RTX 20xx [135], AMD Radeon 5000 Series [74], ARM Mali Bifrost [85], and so on, which are families of accelerators, including models with many traits in common. A distinction between single models is done mainly in performance-oriented discussions. Moreover, for accelerators belonging to the same "evolution line", e.g., GeForce GTX 10xx, GeForce RTX 20xx, and GeForce RTX 30xx, we pick the last two generations, also when more than two have been proposed in the last decade. Lastly, since some accelerator families feature tens of models with few differences, we take into consideration at most three different models, chosen in order to cover a wide spectrum – e.g., entry-level, middle-tier, and high-end/enthusiast.

FPGAs are treated as a special case. They are highly-flexible reconfigurable accelerators, able to accelerate tasks from different domains and with different characteristics, as a significant body of work testifies – e.g., [292–299]. In many research works, they are used to implement accelerator proposals as prototypes, or *soft* accelerators. In this survey, we discuss them as accelerators *per se*, rather than considering the single accelerator instances that have been obtained as *particular* configurations. For instance, in [293], Zhu et al. describe an accelerator for sparse CNNs implemented on a Xilinx ZCU102 board, which includes an Ultrascale FPGA [147]. For our purposes, we describe the Ultrascale FPGA reconfigurable accelerator, rather than the sparse CNN accelerator prototyped on it.

The vastness of the topic here discussed and the level of detail we want to consider for each accelerator, necessarily sets some limitations of this work. First, despite our research effort, there could be some accelerators missing from our study. In order to mitigate the impact of these omissions, we categorize several accelerators from many domains, so to have a good coverage of the whole spectrum and increase the probability that no fundamental aspects are overlooked. Second, during the review phase of our work and after its publication, other accelerators will likely be proposed, both as commercial models or as research proposals. We mitigate the impact of their absence from our analysis by identifying trends that will likely continue in the near future, and shape the accelerators to come. Third, for some accelerators, not all the aspects that we deem relevant are defined/disclosed, and thus we cannot categorize them fully according to our taxonomy. To handle these cases, we provide a "no details" value for the missing aspects.

In the following, Tables 5, 6, 7 and 8 present the accelerator classification in accordance to the discussed taxonomy. Each table covers one macro-category. Accelerator rows with the same values are grouped to highlight similarities.

Tables 2 and 3 list the references related to the classified accelerators and the considered models. By concentrating all of them in two specific tables, we avoid cluttering the classification tables with references that would hamper readability.

3.2. Classification

In the following, we classify more than 100 accelerators and families of accelerators. Table 5 presents their general aspects, Table 6 presents their host coupling details, Table 7 presents their architecture, and Table 8 presents their software aspects. We comment the trends that emerge from the tables.

3.2.1. General aspects

Table 5 shows an overall picture of the accelerator landscape of the last decade.

Type. Accelerator designers have fully embraced the PIM approach, deciding to tackle the memory wall problem [300] by moving computation closer to the memory, sometimes even augmenting the memory itself with processing capabilities, as we show in Table 7. These are

Table 3
List of references of accelerators used to fill the classification tables.

Accelerator	References
AHA371/372	[155,156]
AHA374/378	[157–159]
AHA604/605	[160]
Apple A13 Bionic Neural Engine	[161–163]
Apple A14 Bionic Neural Engine	[164–166]
Baidu Kunlun K200	[167,168]
BioSEAL	[169]
Cambricon-X	[170]
CASCADE	[171]
Cerebras WSE	[172–174]
Coral USB Accelerator	[175–178]
CSRAM	[179]
DaDianNao chip[a]	[180,181]
Darwin	[33]
Darwin-WGA	[182]
DianNao	[181,183]
DIMA Inference Processor	[184]
DIMA-CNN	[185]
DRISA	[186]
DUAL	[187]
Eyeriss	[20,188]
Eyeriss v2	[189]
FlexFlow	[190]
FloatPIM	[191]
FPSA	[192]
GenCache	[193]
Google Pixel Visual Core	[7,194,195]
Google TPU v2	[7,196–199]
Google TPU v3	[7,196–199]
Graphcore IPU-M2000	[200–202]
GraphH	[203]
Graphicionado	[204]
GraphP	[50]
GraphQ	[205]
GraphR	[206]
GRIM-Filter	[207]
GroqCard	[208–210]
Hailo-8	[211]
Heterogeneous-PIM	[212]
HReA	[213]
HRL	[214]
Huawei Atlas 200 AI	[215–218]
Huawei Atlas 300I	[216–219]
Huawei Atlas 300T	[216–218,220]
Huawei Kirin 9000x NPU	[218,221–223]
Huawei Kirin 990x NPU	[218,223–225]
IMP	[226]
Intel Habana Labs HL-100	[227]
Intel Habana Labs HL-20x	[228]
ISAAC	[229]
LerGAN	[230]
Micron Automata Processor	[231–235]
Microsoft Project Catapult	[36,236,237]
Mixed-signal	[238]
Multibit	[239]
NAND-Net	[240]
NDA	[241]
NEST	[242]
Neural Cache	[243]
Neurocube	[244]
NonVolCIM	[245]
NP-CGRA	[246]
Origami	[247,248]
PipeLayer	[249]
Plasticine	[250]
PLiM	[251]
PracticalNDP	[252]
PRIME	[253]
PROMISE	[254]
PuDianNao	[181,255]
PUMA	[256]
PX-CGRA	[257]
Q100	[258,259]
Qualcomm Hexagon 698 DSP	[260–262]
Qualcomm Hexagon 780 DSP	[262–264]
RADAR	[265]
RAPID	[266]
REMUS	[267]
RobustInMem	[268]
Samsung Exynos 9825 NPU	[269,270]
Samsung Exynos 990 NPU	[271]
Sandwich-RAM	[272]
ShiDianNao	[181,273]
Softbrain	[10]
SparseReRAM	[274]
Spin-transfer	[275]
TensorNode	[276]
Tesla FSD Computer NPU	[277,278]
Tesseract	[279]
TETRIS	[280]
Time	[281]
Untether TsunAImi	[282–284]
UPMEM PIM	[285,286]
X-CGRA	[287]
YodaNN	[288]

[a]Single chip.

(*continued on next page*)

usually experimental proposals coming from academia, with only Micron Automata Processor [231,232] and UPMEM PIM [285] coming from industry. The table also highlights their versatility, as they have been employed for ML (both inference and training), Cryptography, Genomics, Graph processing, Data-parallel, NFA, and Pattern matching. We will discuss the PIM type further in Section 5.1.

After PIM, the most represented type is the Manycore accelerator. Unlike PIM, the manycore approach is well-established and has been successfully applied over the years: it is perceived as the natural evolution of the multi-core approach, with an even more extreme commitment to parallelism. As such, it can count many industrial accelerators, usually dedicated to data-parallel and ML workloads, like Huawei's products based on the DaVinci architecture [218] or Intel's Nervana [113,118] and Habana [227,228] products.

Its most prominent declination, i.e., the GPU, is arguably the most common type of accelerator: they are no longer limited to graphics, as they are now fully-fledged data-parallel accelerators. Despite the few entries in the table that reflect the market dominance of only three producers (AMD, ARM, and NVIDIA), GPUs have a prominent role in mobile, desktop, data-center, and HPC market segments.

Around one third of the categorized accelerators are designed as arrays of ALUs/PEs interconnected. Systolic accelerators are the most abundant, and they are employed to target ML workloads, both inference and training. Designers can rely on them to accelerate NNs, which constitute the majority of ML workloads accelerated today (as shown in Table 4), because there is a topological similarity that can be exploited by mapping NN's layers on levels of the systolic array, neurons on PEs, and connections between layers on connections between levels. Moreover, they are particularly suitable to map matrix–matrix multiplications [196], and thus convolutions, which are the heart of CNNs, the most common and most performance-demanding NN type.

CGRAs are gaining increasing attention from the community. Their interesting characteristics (i.e., high efficiency, fast reconfigurability, support for both spatial and temporal computations) make them a good fit for various workloads, especially in the dataflow area. HReA, a CGRA accelerator for the 13-Dwarfs [213], proves that CGRAs, despite their *domain-specific* flexibility, can still be flexible enough to accelerate a high variety of real-world tasks. However, they still present some limitations and challenges that relegate them to academic proposals, as the absence of industrial CGRAs in Table 5 highlights. We will deepen the discussion on these promising architectural solutions in Section 5.2.

Table 4
Supported workloads in detail for accelerators in the ML domain.

Accelerator	ML workload
Apple A13, A14 Bionic Neural Engine	DNN
Baidu Kunlun K200	DNN
Cambricon-X	DNN (Sparse NNs, CNN)
CASCADE	CNN, RNN
Cerebras WSE	CNN
Coral USB Accelerator	CNN, RNN, MLP
CSRAM	CNN
DaDianNao chip	CNN
DianNao	CNN
DIMA Inference Processor	SVM, Template matching, k-NN, Matched filter
DIMA-CNN	CNN
DRISA	CNN, RNN
DUAL	Unsupervised ML
Eyeriss v1, v2	CNN
FlexFlow	CNN
FloatPIM	CNN
FPSA	ANN
Google Pixel Visual Core	CNN
Google TPU v2, v3	CNN, RNN, MLP
Graphcore IPU-M2000	CNN
GroqCard	CNN
Hailo-8	DNN
Heterogeneous-PIM	CNN, GAN
HRL	DNN
Huawei Atlas 200 AI	DNN
Huawei Atlas 300I, 300I Pro	DNN
Huawei Atlas 300T	DNN
Huawei Kirin 900x, 9000x NPU	DNN
Intel Habana Labs HL-100	DNN
Intel Habana Labs HL-20x	DNN
Intel Nervana NNP-I 1100, 1300	DNN
Intel Nervana NNP-T 1300, 1400	DNN
ISAAC	CNN
LerGAN	GAN
Microsoft Project Catapult	DNN
Mixed-signal	BNN
Multibit	CNN
NAND-Net	CNN, BNN
Neural Cache	CNN
Neurocube	CNN
NonVolCIM	CNN
NP-CGRA	CNN
NVIDIA GeForce RTX 20xx, 30xx	DNN (CNN, MLP, RNN), ML (LinReg, Lasso, ElasticNet, k-NN)
Origami	CNN
PipeLayer	CNN
Plasticine	k-means
PracticalNDP	CNN, MLP
PRIME	CNN, MLP
PROMISE	SVM, k-NN, DNN, LinReg, Template matching
PuDianNao	k-NN, k-means, DNN, SVM, LinReg, Naive Bayes, Classification tree
PUMA	MLP, LSTM, CNN, Lin/Log-Reg, SVM
Qualcomm Hexagon 698, 780 DSP	DNN
RobustInMem	SVM
Samsung Exynos 9825, 990 NPU	CNN
Sandwich-RAM	CNN
ShiDianNao	CNN
SparseReRAM	CNN
Spin-transfer	k-means, k-NN, MLP, SVM
Tesla Full Self Driving Computer NPU	DNN
TensorNode	CNN, RNN, MLP
TETRIS	CNN
Time	CNN, MLP
Untether TsunAImi	DNN (Vision, MLP, Recommendation)
UPMEM PIM	DLRM, CNN, k-means, Decision tree
Xilinx Versal ACAP	CNN
YodaNN	BNN

Spatial accelerators are used for inference tasks in all the cases, and training in some cases. Some models are also suited for generic dataflow workloads, as spatial accelerators, like CGRAs, offer a natural way to map DFGs: PEs can implement the processing associated to nodes, and the path on the interconnection can serve as the arc between them. In a couple of accelerators, i.e., Neurocube [244] and TETRIS [280], a spatial array of PEs has been implemented in the logic die of an HMC stack [48].

All FPGA accelerators in the table come from industry. FPGA market is similar to GPU market: few manufacturers, Intel and Xilinx in this case, are responsible for several products that span a high variety of form factors, from chips integrated into development boards to PCI-e cards. They have the ability to accelerate any task, thanks to a fine-grain reconfigurability that can mimic even a general-purpose processor [15,16].

Table 5
Accelerators' general aspects.

					Domain																				
					Broad			Narrow																	
Accelerator		Year	Type	Origin	Data-parallel	Dataflow	General-purpose	ML Inference	ML Training	AR	Computer Vision	Cryptography	Data (De)Compression	Database Manipulation	Genome Seq. Alignment	Graph Processing	Graphics	Multimedia	NFA	Network Flow Encryption	Pattern Matching	Search Acceleration	Programmability	Reconfigurability
AHA	371, 372 374, 378	2014 2015	ASIC	Ind									×										N	N
AHA 604, 605		2016	ASIC	Ind								×	×										N	N
AMD Radeon RX	5000 Series 6000 Series	2019 2020	GPU	Ind	×												×	×					Y	N
ARM Mali	Bifrost Valhall	2016 2019	GPU	Ind	×												×	×					Y	N
Apple Bionic Neural Engine	A13 A14	2019 2020	-[a]	Ind				×															Y	N
Baidu Kunlun K200		2021	Manycore	Ind	×			×	×														Y	N
BioSEAL		2020	PIM	Acad											×								Y	N
Cambricon-X		2016	Manycore Systolic[b]	Acad				×															Y	N
CASCADE		2019	PIM Systolic[c]	Acad				×															Y	N
Cerebras WSE		2019	Spatial	Ind		×		×	×														Y	N
Coral USB Accelerator		2019	Systolic	Ind				×															Y	N
CSRAM		2018	PIM	Acad				×															N	N
DaDianNao chip		2014	Manycore Systolic[b]	Acad				×	×														Y	N
Darwin		2018	ASIC	Acad											×								N	N
Darwin-WGA		2019	ASIC	Acad											×								N	N
DianNao		2014	Systolic	Acad				×															Y	N
DIMA Inference Processor		2018	PIM	Acad				×															Y	N
DIMA-CNN		2018	PIM	Acad				×															N	N
DRISA		2017	PIM	Acad	×			×															N	Y
DUAL		2020	PIM	Acad				×															Y	N
Eyeriss	v1 v2	2016 2019	Spatial	Acad		×		×															Y	N
FlexFlow		2017	Spatial	Acad		×		×	×														Y	N
FloatPIM		2019	PIM	Acad				×	×														Y	N
FPSA		2019	PIM	Acad				×															N	Y
GenCache		2019	PIM	Acad											×								N	N
Google Pixel Visual Core		2017	Manycore	Ind				×			×							×					Y	N
Google TPU	v2 v3	2017 2018	Systolic	Ind				×	×														Y	N
Graphcore IPU-M2000		2020	Manycore	Ind				×	×							×							Y	N

(*continued on next page*)

Table 5 (*continued*).

Accelerator		Year	Type	Origin	Domain: Broad: Data-parallel	Broad: Dataflow	Broad: General-purpose	Narrow: ML Inference	Narrow: ML Training	Narrow: AR	Narrow: Computer Vision	Narrow: Cryptography	Narrow: Data (De)Compression	Narrow: Database Manipulation	Narrow: Genome Seq. Alignment	Narrow: Graph Processing	Narrow: Graphics	Narrow: Multimedia	Narrow: NFA	Narrow: Network Flow Encryption	Narrow: Pattern Matching	Narrow: Search Acceleration	Programmability	Reconfigurability
GraphH		2019	PIM	Acad												×						Y	N
Graphicionado		2016	Manycore	Acad												×						Y	N
GraphP		2018	PIM	Acad												×						Y	N
GraphQ		2019	PIM	Acad												×						Y	N
GraphR		2018	PIM	Acad												×						Y	N
GRIM-Filter		2018	PIM	Acad											×							N	N
GroqCard		2020	Spatial	Ind				×	×													Y	N
Hailo-8		2019	Spatial	Ind				×														Y	N
Heterogeneous-PIM		2018	PIM	Acad				×	×													Y	N
HReA		2018	CGRA	Acad			×															N	Y
HRL		2016	PIM CGRA[d]	Acad	×			×								×						N	Y
Huawei Atlas 200 AI		2019	Manycore	Ind				×										×				Y	N
Huawei Atlas 300I, 300I Pro		2019	Manycore	Ind				×										×				Y	N
Huawei Atlas 300T		2019	Manycore	Ind					×									×				Y	N
Huawei Kirin	990x NPU	2019	Manycore	Ind				×														Y	N
	9000x NPU	2020																					
IMP		2018	PIM	Acad	×																	Y	N
Intel FPGA 10 Series		2013	FPGA	Ind			×															N	Y
Intel FPGA F-Series		2019	FPGA	Ind			×															N	Y
Intel FPGA V Series		2013	FPGA	Ind			×															N	Y
Intel Graphics Technology	Gen11	2019	GPU	Ind	×												×					Y	N
	Xe-LP	2020																					
Intel Habana Labs HL-100		2019	Manycore	Ind	×[e]			×														Y	N
Intel Habana Labs HL-20x		2019	Manycore	Ind	×[e]				×													Y	N
Intel Nervana NNP-I 1100, 1300		2019	Manycore	Ind				×														Y	N
Intel Nervana NNP-T 1300, 1400		2019	Manycore	Ind					×													Y	N
Intel Xeon Phi Knights	Landing	2016	Manycore	Ind	×																	Y	N
	Mill	2017																					
ISAAC		2016	PIM	Acad				×														Y	N
LerGAN		2018	PIM	Acad				×	×													Y	N
Micron Automata Processor		2014	PIM	Ind															×		×	N	Y
Microsoft Project Catapult		2015	FPGA	Ind				×												×		N	Y
Mixed-signal		2018	PIM	Acad				×														N	N
Multibit		2019	PIM	Acad				×														N	N
NAND-Net		2019	PIM	Acad				×														N	N
NDA		2015	PIM CGRA[f]	Acad	×																	N	Y
NEC SX-Aurora TSUBASA V.E.	Type10	2018	Vector	Ind	×																	Y	N
	Type20	2020																					
NEST		2020	PIM	Acad											×							N	N
Neural Cache		2018	PIM	Acad				×														N	N

(*continued on next page*)

Table 5 (*continued*).

					Domain																				
					Broad			Narrow																	
Accelerator		Year	Type	Origin	Data-parallel	Dataflow	General-purpose	ML Inference	ML Training	AR	Computer Vision	Cryptography	Data (De)Compression	Database Manipulation	Genome Seq. Alignment	Graph Processing	Graphics	Multimedia	NFA	Network Flow Encryption	Pattern Matching	Search Acceleration	Programmability	Reconfigurability
Neurocube		2016	PIM Spatial[g]	Acad				×	×														Y	N
NonVolCIM		2018	PIM	Acad				×															N	N
NP-CGRA		2021	CGRA	Acad				×															N	Y
NVIDIA GeForce	RTX 20xx RTX 30xx	2018 2020	GPU	Ind	×			×	×								×	×					Y	N
Origami		2015	Systolic	Acad				×															N	N
PipeLayer		2017	PIM	Acad				×	×														Y	N
Plasticine		2017	CGRA	Acad	×			×								×							N	Y
PLiM		2016	PIM	Acad	×							×											Y	N
PracticalNDP		2015	PIM	Acad				×	×							×							Y	N
PRIME		2016	PIM	Acad				×															Y	N
PROMISE		2018	PIM	Acad				×															Y	N
PuDianNao		2015	Systolic	Acad				×	×														Y	N
PUMA		2019	PIM	Acad				×															Y	N
PX-CGRA		2018	CGRA	Acad		×												×					N	Y
Q100		2014	ASIC	Acad										×									N	N
Qualcomm Hexagon	698 DSP 780 DSP	2020 2021	Vector	Ind				×		×	×							×					Y	N
RADAR		2018	PIM	Acad											×								N	N
RAPID		2019	PIM	Acad											×								N	N
REMUS		2013	CGRA	Acad														×					N	Y
RobustInMem		2018	PIM	Acad				×	×														N	N
Samsung Exynos	9825 NPU 990 NPU	2019 2020	Systolic	Ind				×															Y	N
Sandwich-RAM		2019	PIM	Acad				×															N	N
ShiDianNao		2015	Systolic	Acad				×															Y	N
Softbrain		2017	CGRA	Acad		×																	N	Y
SparseReRAM		2019	PIM	Acad				×															Y	N
Spin-transfer		2017	PIM	Acad				×															Y	N
Tesla Full Self Driving Computer NPU		2019	Systolic	Ind				×															Y	N
TensorNode		2019	PIM	Acad				×															Y	N

(*continued on next page*)

Table 5 (*continued*).

				Domain																			
				Broad			Narrow																
Accelerator	Year	Type	Origin	Data-parallel	Dataflow	General-purpose	ML Inference	ML Training	AR	Computer Vision	Cryptography	Data (De)Compression	Database Manipulation	Genome Seq. Alignment	Graph Processing	Graphics	Multimedia	NFA	Network Flow Encryption	Pattern Matching	Search Acceleration	Programmability	Reconfigurability
Tesseract	2015	PIM	Acad												×							Y	N
TETRIS	2017	PIM Spatial[g]	Acad				×															Y	N
Time	2018	PIM	Acad				×	×														Y	N
Untether TsunAImi	2021	Spatial	Ind				×															Y	N
UPMEM PIM	2020	PIM	Ind	×				×			×							×				Y	N
X-CGRA	2020	CGRA	Acad		×												×					N	Y
Xilinx FPGA 7 Series	2017	FPGA	Ind			×																N	Y
Xilinx FPGA Ultrascale+ Series	2015	FPGA	Ind			×																N	Y
Xilinx Versal ACAP	2019	FPGA	Ind			×	×	×			×											Y	Y
YodaNN	2018	Systolic	Acad				×															N	N

[a]No details available

[b]Many PEs, each being a systolic array of arithmetic units.

[c]Systolic array of cascaded ReRAM MAC arrays.

[d]CGRA implemented in the logic die into an HMC stack.

[e]Fully programmable Tensor Processing Cores.

[f]CGRA integrated near conventional DRAM.

[g]Spatial array of PEs implemented in the logic die of an HMC stack.

Although some concepts related to vector computing are present in other types of accelerators (e.g., GPUs), the only examples of *classic* vector accelerators are Qualcomm Hexagon DSPs [262] and NEC SX-Aurora TSUBASA Vector Engines [130]. The former are integrated in mobile SoCs and operate in inference, AR, and Computer Vision domains, while the latter are shipped as PCI-e cards and can accelerate any data-parallel task.

Accelerators labeled as ASIC are non-programmable, non-reconfigurable, and are tailored on targeted workloads. Q100, in particular, is conceived as 11 tiles aggregated in a single ASIC, each performing a particular Database-related function [258].

Domain. As for accelerators' domains, Table 5 testifies the "Cambrian explosion" of ML-oriented proposals. Most accelerators in the table belong to this category, with inference being around three times more represented than training. This gap may be due to the fact that training, with respect to inference, has generally higher requirements in terms of storage capacity and arithmetic precision. More storage is needed because intermediate results need to be preserved; while higher precision is needed because of the higher requirements of gradient-based optimization techniques, needed to calculate the optimal weight values [21].

ML is a multifaceted domain that comprehends a vast body of algorithmic solutions. The important role that AI and ML occupy in today's society and the ever-growing interest of the research community in this field requires an increased level of detail. Thus, we highlight the class of ML workloads that are commonly targeted by each accelerator in Table 4. DNNs, especially CNNs, are the most abundant. They have two main characteristics that make them particularly amenable for acceleration:

- they are employed in several AI applications like speech recognition, image recognition, self-driving cars, cancer detection, and gaming [21], thus a DNN accelerator, despite being a *specialized* device, is still versatile enough to accelerate many real-world applications;
- they have a high computational complexity, which makes techniques like hardware acceleration an important strategy to improve energy efficiency and throughput and expand the opportunities for DNN deployment [21].

The second most represented domain is Data-parallel (around 16% of cases), which includes diverse real-world applications that span various fields: linear algebra (GEMM, convolution, triangular factorization), graphics and imaging (rendering, filtering), physical modeling and simulation (weather forecasting, molecular dynamics, LBM), finance (Black–Scholes, option pricing), Monte Carlo simulations, etc.

These applications are characterized by reduced control logic and the employment of the same processing steps on a multitude of elements in the managed dataset. Therefore, they are a good match for dedicated hardware accelerators: control logic can be minimized in favor of computing units (e.g., ALUs, FPUs, simple PEs), that are replicated in pursue of massive parallelism.

With ever-increasing datasets that emerge from such a diverse range of applications, the prominence of energy efficiency and throughput grows accordingly, making hardware acceleration even more important. Table 5 shows that various accelerators target this domain: all the GPUs; some Manycore accelerators like Baidu Kunlun K200 [168], Intel Habana Labs [227,228], and Intel Xeon Phi [125,126]; some PIM proposals like DRISA [186], PLiM [251], and UPMEM PIM [285]; two PIM/CGRA accelerators; the NEC SX-Aurora TSUBASA vector accelerator [130]; and Plasticine [250], a CGRA.

Multimedia can count on various accelerator proposals. Image/video filtering is a task usually performed by GPUs, but there are also specific proposals that target this domain. For instance, the Huawei Atlas accelerators [215,219,220], Qualcomm Hexagon DSPs [262], or three CGRAs (PX-CGRA [257], X-CGRA [287], REMUS [267]).

The Genome Sequence Alignment domain is populated mainly by PIM accelerators. Its applications are characterized by large datasets that involve hundreds of millions of DNA base pairs. With the PIM design, the processing can happen where data are stored, avoiding a massive data transfer altogether that would limit bandwidth and energy efficiency.

Table 5 shows that some accelerators are capable of accelerating tasks even in the general-purpose domain. As specified before, FPGAs are able to do that thanks to their fine-grain reconfigurability in both control logic and interconnect. CGRAs, which are *coarse-grain* reconfigurable, usually target more specific domains, but this is not the case of HReA [213]. It has been designed with a "hybrid-grained datapath" [213] that supports data-intensive applications as well as control intensive ones, like dynamic programming, backtracking, and FSM.

Dataflow is a computation paradigm that puts data at the center and expresses a program in terms of the operations performed on them, highlighting the concurrency between operations. Since its first proposal as a form of program representation by Dennis and Misunas [27], a "data-flow processor" was proposed, arguing that the classic von Neumann design would not be a good fit. In this processor, the memory is organized in *instruction cells* equipped with three registers that hold an instruction and two operands.

As soon as the operands are available, the content of the registers is forwarded to an *operation unit* that applies the operation indicated by the instruction and forwards the result to a destination cell. Today's Dataflow accelerators from Table 5 adopt solutions inspired by the data-flow processor, but more advanced: four of them are Spatial architectures (Cerebras WSE [172], Eyeriss [20], Eyeriss v2 [189], and FlexFlow [190]), and three are CGRAs (PX-CGRA [257], Softbrain [10], X-CGRA [287]). The instruction cells have been replaced by PEs that include the capabilities of the operation units as well, the interconnection network allows a direct forward of the result to the next PE, and the instruction-driven operation has been replaced by a reconfigurable solution in the CGRAs.

Different types of accelerator target Graph processing: Spatial, Manycore, CGRA, and PIM. The latter is the preferred approach in seven out of the ten considered cases. Typical graph processing tasks that would benefit from acceleration, like big-data analytics and PageRank, are defined by large-scale graphs, poor locality, and random access patterns [50,203], with memory being the bottleneck for both performance and efficiency. Therefore, the PIM approach becomes an amenable solution: it has the capacity to avoid data movements altogether, keeping only those related to communication between processing elements.

Programmability/Reconfigurability. The most common combination inherent to accelerators' programmability and reconfigurability is "programmable, non-reconfigurable". Most accelerators, despite being domain-specific, still offer flexibility in the form of programmability, i.e., the ability to perform operations according to user-provided instructions. However, around one fifth of the accelerators are *inflexible*. Generally speaking, an inflexible design may be a good choice in terms of throughput and energy efficiency, but it may limit an accelerator's lifetime and applicability. In our opinion, all the non-programmable, non-reconfigurable accelerators analyzed so far are rather protected against this risk: those in the Genome Sequence Alignment domain accelerate a task that offers little to no degrees of freedom, while those in the Data (De)Compression domain accelerate well-established algorithms that have not changed in years (e.g., zlib and gzip algorithms).

The many tiles of the mentioned Q100 [258] act as different functional units that offer a good coverage of the possible workloads. Finally, as Table 4 highlights, those dedicated to ML inference are all dedicated to BNN/CNN acceleration (except RobustInMem [268]) and retain the possibility to employ user-defined weights, which is sufficient to support a high number of different scenarios and use cases.

Table 6
Accelerators' host coupling.

Accelerator	Connection											SW interface					Memory sharing				OS coupling			
	bump-in-the-wire	CCIX	DRAM slot	HMC	integrated on-chip	M.2	network interconnect	OCP OAM	PCI-e	USB	no details	memory mgmt	soft configuration	PL configuration	kernel launch	op execution	none	LLC	DRAM	no details	none	drivers	custom OS	no details
AHA371, AHA372, AHA374, AHA378									×							×	×					×		
AHA604, AHA605									×							×	×					×		
AMD Radeon RX 5000 Series, 6000 Series									×			×			×		×					×		
ARM Mali Bifrost, Valhall					×							×			×					×		×		
Apple A13 Bionic Neural Engine, A14 Bionic Neural Engine					×								×			×		×				×		
Baidu Kunlun K200									×			×	×		×	×	×					×		
BioSEAL											×	×				×	×							×
Cambricon-X											×		×			×				×				×
CASCADE											×	×	×			×	×							×
Cerebras WSE							×								×		×					×		
Coral USB Accelerator										×			×			×	×					×		
CSRAM											×	×				×				×				×
DaDianNao chip											×	×	×			×				×				×
Darwin											×					×	×							×
Darwin-WGA											×		×			×	×							×
DianNao											×	×	×			×				×				×
DIMA Inference Processor											×	×	×			×				×				×
DIMA-CNN											×	×	×			×				×				×
DRISA			×[a]						×[a]			×			×					×		×		
DUAL											×	×	×			×	×							×
Eyeriss									×				×			×				×				×
Eyeriss v2											×		×			×				×				×
FlexFlow											×					×				×				×
FloatPIM											×	×	×			×	×							×
FPSA											×	×	×	×		×	×							×
GenCache											×					×	×							×
Google Pixel Visual Core					×								×		×	×			×			×		
Google TPU v2, v3							×						×			×	×					×		
Graphcore IPU-M2000							×		×			×			×		×					×		
GraphH				×								×	×		×				×			×		
Graphicionado											×	×			×		×							×
GraphP				×								×			×				×			×		
GraphQ				×								×			×				×			×		
GraphR											×	×				×	×							×
GRIM-Filter												×				×	×							×
GroqCard									×				×			×	×					×		
Hailo-8						×			×				×			×	×					×		
Heterogeneous-PIM				×								×	×		×	×			×					×
HReA					×									×		×			×					×
HRL				×										×		×				×		×		
Huawei Atlas 200 AI									×	×		×	×		×	×	×					×		
Huawei Atlas 300I, 300I Pro, 300T									×			×	×		×	×	×					×		
Huawei Kirin 990x NPU, 9000x NPU					×							×	×		×	×		×				×		

(*continued on next page*)

Table 6 (*continued*).

Accelerator	Connection											SW interface					Memory sharing				OS coupling			
	bump-in-the-wire	CCIX	DRAM slot	HMC	integrated on-chip	M.2	network interconnect	OCP OAM	PCI-e	USB	no details	memory mgmt	soft configuration	PL configuration	kernel launch	op execution	none	LLC	DRAM	no details	none	drivers	custom OS	no details
IMP			×									×			×				×					×
Intel FPGA V Series, 10 Series, F-Series					×				×			×		×		×	×					×		
Intel Graphics Technology Gen11, Xe-LP					×							×			×			×				×		
Intel Habana Labs HL-100									×			×			×		×					×		
Intel Habana Labs HL-20x						×			×			×			×		×					×		
Intel Nervana NNP-I 1100, 1300						×			×				×			×	×					×		
Intel Nervana NNP-T 1300, 1400								×	×			×			×		×					×		
Intel Xeon Phi Knights Landing, Knights Mill									×			×			×		×					×		
ISAAC											×	×	×			×	×							×
LerGAN											×	×	×		×	×	×					×		
Micron Automata Processor									×			×		×		×	×							×
Microsoft Project Catapult	×								×			×		×		×	×					×		
Mixed-signal	×											×	×			×				×				×
Multibit											×	×	×			×				×				×
NAND-Net											×	×	×			×				×				×
NDA			×									×		×		×			×					×
NEC SX-Aurora TSUBASA Vector Engine Type10, Type20									×			×			×		×						×	
NEST											×					×				×				×
Neural Cache					×							×				×								×
Neurocube				×								×	×			×			×					×
NonVolCIM											×	×	×			×				×				×
NP-CGRA											×			×		×				×				×
NVIDIA GeForce RTX 20xx, 30xx									×			×	×		×	×	×					×		
Origami					×											×	×				×			
PipeLayer											×	×	×			×	×							×
Plasticine											×	×			×				×					×
PLiM			×									×	×		×				×					×
PracticalNDP				×								×	×		×				×			×		
PRIME											×	×	×		×	×				×		×		
PROMISE											×	×			×	×				×	×[b]			
PuDianNao											×	×	×			×				×				×
PUMA											×	×	×		×	×	×							×
PX-CGRA											×			×		×	×							×
Q100											×					×				×				×
Qualcomm Hexagon 698 DSP, 780 DSP					×							×			×			×				×		
RADAR											×					×				×				×
RAPID											×		×			×				×				×

(*continued on next page*)

Table 6 (*continued*).

	Connection											SW interface					Memory sharing				OS coupling			
Accelerator	bump-in-the-wire	CCIX	DRAM slot	HMC	integrated on-chip	M.2	network interconnect	OCP OAM	PCI-e	USB	no details	memory mgmt	soft configuration	PL configuration	kernel launch	op execution	none	LLC	DRAM	no details	none	drivers	custom OS	no details
REMUS					×							×		×		×	×							×
RobustInMem	×											×	×			×				×				×
Samsung Exynos 9825 NPU					×								×			×			×			×		
Samsung Exynos 990 NPU					×								×			×			×			×		
Sandwich-RAM											×	×	×			×				×				×
ShiDianNao	×											×				×	×							×
Softbrain					×							×		×		×		×[c]	×[c]		×[b]			
SparseReRAM											×	×	×			×	×							×
Spin-transfer					×							×	×			×	×				×[b]			
TensorNode			×									×				×			×			×[d]		
Tesla Full Self Driving Computer NPU					×							×				×			×					×
Tesseract				×								×			×				×			×		
TETRIS				×								×	×			×			×					×
Time											×	×	×			×	×							×
Untether TsunAImi									×			×			×		×					×		
UPMEM PIM			×									×			×				×			×		
X-CGRA											×			×		×	×							×
Xilinx FPGA 7 Series, Ultrascale+ Series					×				×			×		×		×	×					×		
Xilinx Versal ACAP		×							×			×		×	×	×	×					×		
YodaNN					×								×			×				×				×

[a]Either PCI-e or DRAM slot.

[b]CPU ISA extended, no OS intervention necessary.

[c]Either LLC or DRAM.

[d]NVIDIA drivers and runtime extended to perform operations on TensorNode.

The non-programmable, reconfigurable accelerators are all FPGAs or CGRAs except two PIM accelerators: FPSA [192] and Micron Automata Processor [231]. Both of them have reconfigurable routing architectures that connect ReRAM-crossbar-based PEs in one case and State Transition Elements in another. The only programmable, reconfigurable accelerator is Xilinx Versal ACAP [152], which is an FPGA with many specialized reconfigurable blocks (DSP, AI, Crypto, etc.) and two integrated processors.

3.2.2. Host coupling

Table 6 shows how the accelerators are connected to the rest of the (*host*) system. Unfortunately, many accelerators, mainly from Academia, are not described at this level of detail. In scientific papers, this aspect is often overlooked as authors usually concentrate on the architectural details and the achieved *metrics* in terms of power, efficiency, throughput, and area.

Connection strategy. Connection strategy describes their physical connection to the host. Between the accelerators that give this information, about 36% of them use PCI-e as their connection strategy. In desktop and server computers, PCI-e is the most common way to connect external peripherals like graphic cards, network cards, etc. Its widespread use, testified by the presence of ad-hoc slots on every desktop/server motherboard, makes it a "safe choice" that ensures maximum compatibility with existing systems. Two solutions related to PCI-e appear in the table: CCIX, which augments PCI-e with a unified, coherent virtual space, and M.2, which uses PCI-e data transfer lanes on the motherboard.

Right after PCI-e, on-die integration is the most common solution (about 26%). It is particularly suitable for mobile systems, that rely on complex SoCs that already integrate a CPU and a variety of accelerators like GPUs and NPUs [161,164,221,224,260,263]. We expect this strategy to become more common in the future, as SoCs have some interesting characteristics (e.g., low latency, high integration, high efficiency) that will likely lead them to expand their presence also in desktop and server systems, as the advent of Apple's M1 may anticipate [301].

A surprising aspect that emerges from Table 6 is that the third most abundant connection strategy is HMC, albeit considering the omissions due to lack of information. It is used by PIM accelerators that implement their logic in the logic die of an HMC stack and adopt the connections provided by the HMC technology, like DIMM slot and Intel QuickPath Interconnect [50]. After HMC comes DRAM slot, which is another solution used by PIM accelerators. In this case, these are designed as DRAM cards augmented with acceleration logic.

Bump-in-the-wire is a solution that works for accelerators performing a fixed filtering task: ShiDianNao [273], Mixed-signal [238], and RobustInMem [268] are designed to be connected directly to a sensor and filter data coming from it, while Microsoft Project Catapult [36] can be placed between the NIC and the top-of-rack switch. Network interconnect is used by industrial accelerators designed to integrate easily with a data-center infrastructure (i.e., Cerebras WSE [172], Google TPUs [197], and Graphcore IPU-M2000 [201]). They are shipped in PoDs that are connected to the network and controlled by hosts as network nodes.

SW interface. SW interface describes the operations an application programmer must perform in order to interact with the accelerator. It is another category often not covered in academic proposals. Unlike the connection strategy, it is possible to make assumptions based on other characteristics of the accelerators, such as their programmability, their reconfigurability, the nature of the accelerated workloads, the architectural components, etc. Therefore, in this category, rather than adding a *no details* column, we make assumptions based on these aspects.

Op execution is the most common host-side action to interact with the accelerator. Its abundance depends on the fact that it is the only solution to trigger execution on non-programmable accelerators, but can also be the preferred interface for programmable accelerators: in fact, as long as accelerator-side control flow is not involved, an approach based on library calls wrapping single operations is preferred to the development of a fully-fledged programming language and its necessary tools – a compiler, above all.

Another common action is the explicit memory management on the accelerator. This can be performed at low-level, e.g., with explicit LOAD and STORE instructions as in the accelerators of the DianNao family [181], or by invoking memory management API functions like cudaMalloc, cudaFree, and cudaMemcpy calls in the CUDA C++ extension for NVIDIA GPUs [65].

Soft configuration is a widespread solution to manage accelerators in contexts where little flexibility is needed. This does not reflect necessarily the capability of the accelerator, as highly flexible, even Turing-complete accelerators can provide a soft configuration procedure as a convenient interface. For instance, many accelerators recur to it to perform ML inference tasks: a soft configuration procedure takes care of loading a model into the accelerator, and a corresponding *op execution* runs the model on the input.

Kernel launch causes the execution of a *function* on the accelerator. The kernel is always expressed in a high-level programming language and is compiled with an accelerator-specific compiler. Therefore, accelerators supporting this operation offer a high degree of flexibility. A kernel launch operation does not necessarily execute the kernel function as-is or as the executable body of a multitude of threads (à-la data-parallel model), but can be part of a more complex structure. For instance, in Graph processing accelerators like Graphicionado [204], GraphP [50], and GraphQ [205], the kernel function is executed as a node function in a user-defined graph.

PL configuration concerns the act of reconfiguring the programmable logic of a reconfigurable accelerator. FPGAs and CGRAs support this procedure, although with different languages and time scales. Generally, FPGAs achieve this by the means of RTL/HDL, and CGRAs could support domain-specific lightweight possibilities that allow temporal computations.

Memory sharing. Memory sharing describes whether a level of the memory hierarchy is shared between the host and the accelerator. PIM accelerators often realize some form of memory sharing because their logic is implemented *in-* or *near-* host memory that can also be used as *classic* memory in non-accelerated, CPU-driven tasks. Apart from Softbrain [10], that can be implemented to share LLC or DRAM with the host, all the other PIM accelerators sharing a memory level share DRAM. In particular, it happens for the many accelerators implemented in the logic die of HMC stacks. This is not the only possible solution: many PIM accelerators with in-SRAM logic, as Table 7 shows, could be implemented in the LLC or even into inner caches. Unfortunately, the majority of papers describing them report no details on their possible use as enhancements of existing caches.

Another group of accelerators that share memory with the host on some level are those integrated on-die. From a mere form factor perspective, these accelerators have access to a limited die area, do not have room for memory apart from small local memories or caches, and thus need to be connected to the host memory. In some cases, it is LLC (e.g., Apple Bionic Neural Engines [161,164], Huawei Kirins [221,224]), in others it is DRAM (e.g., Google Pixel Visual Core [194], Samsung Exynos [270,271]). We will examine in depth the implications of memory sharing on coherence and virtual memory in Section 5.

OS coupling. Drivers are chosen in 41 out of 101 cases for OS coupling – albeit 54 of these do not provide this detail. For many accelerators, they are the de-facto standard way to interact with the OS to perform various tasks, such as managing accelerators' resources via MMIO, orchestrating concurrent accesses, taking care of virtual-to-physical memory mapping, and so on. In some cases, the accelerator can work independently of the OS, as the few cases with *none* OS

coupling testify. Origami [247] achieves this by interposing a controlling FPGA between it and the host: an OS coupling between host and FPGA is still necessary, but the accelerator can operate transparently. The other three solutions, i.e., PROMISE [254], Softbrain [10], and Spin-transfer [275], adopt ad-hoc ISA extensions that allow the CPU to communicate directly with the accelerator. On the other side of the spectrum, the NEC SX-Aurora TSUBASA Vector Engine accelerators [130] move many responsibilities into the OS, requiring even a custom Linux OS equipped with special services and modules. This is used to realize what the authors call the "reverse offloading" mechanism: when an executable is launched, the code is transferred to the accelerator, a process is allocated there and a corresponding pseudo-process is allocated on the host. This pseudo-process is responsible for: allocating virtual memory pages, asking a service to allocate physical pages on the accelerator, executing system calls/handling exceptions originated on the accelerator, and sending back the results [132].

3.2.3. Architecture

Accelerator descriptions, both in industrial datasheets and academic papers, usually reserve to the architecture presentation a prominent role. Arguably, this aspect more than others highlights the peculiarity of an accelerator. As for our categorization, Table 7 shows the architectural details of the analyzed accelerators. We have been able to find relevant information for all the accelerators except Apple A13 and A14 Bionic Neural Engines [161,164]. Table 7 does not list *all* their architectural components, but rather the *main* ones. We selected them so to avoid risking a dilution of the pertinent information and clutter the Table with hardware components too specific for single accelerators.

General-purpose resources. Caches are fundamental components in general-purpose processors. They help to increase data bandwidth and reduce latency dramatically, avoiding costly accesses to the main memory. However, they are not as common in accelerators: about 23% of the listed ones have an L1 cache, 15% an L2 cache, and only 4% an L3 cache. Most of the time, they are replaced by local memories.

Accelerators that feature a DRAM technology also found in general-purpose systems (DDR3, DDR4, LPDDR4, LPDDR4X) are even fewer than those with cache memories: only about 9%. This is due to many factors: integrated accelerators do not need their DRAM, as they usually connect to a shared LLC or directly to the system DRAM; the variety of existing memory technologies allow designers to select solutions that are a better fit for the accelerator requirements (e.g., GDDRs have higher bandwidth and larger buses optimized for graphics workloads, HBMs have higher bandwidth and efficiency); some PIM accelerators are designed around a different memory technology and cannot adapt to DRAM (e.g., ReRAM, HMC). However, there are also PIM accelerators that, instead, augment the standard DRAM with near-data processing: Micron Automata Processor [231] and NDA [241] augment DDR3, NEST [242] and UPMEM PIM [285] augment DDR4.

Table 7 lists general-purpose computing engines directly involved in the accelerated computations as the main work-force. We exclude those involved in orchestration/control of other resources. We make this distinction because almost every accelerator uses one or more controllers to manage its resources (e.g., memory units, systolic arrays, etc.), but only 10% of them employ general-purpose cores as their principal *computing engines*. Therefore, we concentrate our analysis on the last case.

Xilinx Versal ACAP [152] accelerators have two ARM dual-cores on-board: Cortex-A72 and Cortex-R5F. They may have a role in accelerating diverse tasks, but can also run an O.S. to make the Versal ACAP an autonomous board, rather than an accelerator. Two ML-oriented (both inference and training) PIM accelerators use ARM cores as computing engines: Heterogeneous-PIM [212] and PracticalNDP [252]. They integrate a simple general-purpose in-order ARM core (Cortex-A7 in PracticalNDP, Cortex-A9 in Heterogeneous-PIM) into the logic die of an HMC stack to realize near-data processing. This solution allows them to provide the highest flexibility, as the integrated processors are able of performing *any* calculation on the data stored in each vault. Tesseract [279] is a Graph processing accelerator designed according to the same principle. The authors choose a generic "single-issue, in-order" core similar to a Cortex-A5 as their computing engine. GraphH [203], GraphP [50], and GraphQ [205] are Graph processing accelerators inspired by Tesseract. GraphH chooses to integrate an ARM Cortex-A5 with FPU, but without cache; while the other two are less specific à-la Tesseract. General-purpose scalar units are also present in another PIM accelerator, UPMEM PIM [285,286] that integrates a scalar unit (DPU) into a standard DDR4 DIMM, and in the Manycore accelerator Baidu Kunlun K200, as part of their XPU-clusters [168], that feature an ALU for basic instructions and an SFU for log, exp, sqrt, etc.

Two accelerators only feature x86 processors, both Manycore proposals from Intel: Intel Xeon Phi [125,126], for Data-parallel workloads, and Intel Nervana NNP-I [113], for ML inference. In both cases, the presence of x86 cores is a plus from the programmability standpoint, as familiar software stacks for desktop development can be adapted to write accelerated code. Interestingly, both projects have been abandoned. Xeon Phi presumably because of the competition with GPUs, as both families of accelerators target the same workloads, but GPUs are generally cheaper and already widespread in desktop and data-center market segments. The promise of a common ISA between data-parallel accelerator and general-purpose cores was not enough to steer the enormous momentum of the GPU ecosystem. Nervana was abandoned when Intel decided to acquire Habana Labs to switch to a unified architecture for inference and training [302].

Special-purpose resources. The role cache memories play in general-purpose processors is usually played by local memories, that, conversely, are present in 86% of cases. Under the *local memory* name, we list on-chip SRAM memories implicitly managed by the execution flow (e.g., to store intermediate results) or explicitly managed by the programmer. Unfortunately, the lack of details concerning the software aspects for many accelerators does not allow us to make a clear distinction for them. Local memories are a simple but effective solution to tackle wire-delay issues and limit transfers from the main memory through data reuse, improving efficiency, latency, and also performance [12,303]. These memories are usually small, but this is not a limiting factor, as load and store cycles can be effectively integrated in streaming and pipelining mechanisms with no stalls, overlapping computation and data transfers.

With respect to cache memories, local memories allow for a higher flexibility, since they can be managed according to the policies that most apply to the accelerated task, in both execution-driven and code-driven cases. In particular, the first kind is the most appropriate choice for accelerated tasks with a regular memory access and reuse pattern,[1] while the second kind can be employed even in irregular cases. We can see in Table 7 that only 20% of accelerators that feature a cache memory do not rely also on a local memory: GraphP [50], Intel Xeon Phis [125,126], NEC SX-Aurora TSUBASAs [130], Qualcomm Hexagon DSPs [262], and Tesseract [279].

The table also highlights that 3D-stacked memories are a mature technology that can be employed in various cases — about one fifth of the total. JEDEC's HBM [53], Micron's HMC [48,49], and Intel's MCDRAM [52] (which is based on HMC), are all high-bandwidth memory technologies designed as a "memory cube", i.e., a stack of memory dies, with one logic die on the bottom, with through-silicon vias and micro-bumps. These technologies are employed in PIM accelerators, with the computing logic implemented in the logic die, and to serve as high-performance on-board memories for PCI-e based accelerators.

About half of the accelerators rely on special-purpose compute resources. These are roughly equally represented across accelerators,

[1] Nowatzki et al. claim that most "acceleratable algorithms" share this characteristic [10].

Table 7
Architectural aspects.

	General-purpose resources										Special-purpose resources																												
	Memory							Comp. engine			Memory									Comp. engine				Fixed-function							PL		Mem+comp.						
Accelerator	L1	L2	L3	DDR3	DDR4	LPDDR4	LPDDR4X	ARM CPU	x86 CPU	Scalar unit	GDDR6	GDDR6X	HBM	HBM2	MCDRAM	HMC-RAM	eDRAM	Local memory	Spatial array	Vector processor	Systolic array	Tensor core	Rendering pipeline engine	Physics engine	ML auxiliary-functions engine	Cryptography engine	AD/DA Converter	Compr./Decompr. engine	Alignment engine	DB Operations engine	Fine-grain	Coarse-grain	ReRAM	STT-MRAM	In-DRAM logic	In-SRAM logic	Near-memory logic
AHA371, AHA372																											×										
AHA374, AHA378																											×										
AHA604, AHA605																										×		×									
AMD Radeon RX 5000 Series	×	×									×							×		×			×	×													
AMD Radeon RX 6000 Series	×	×									×							×		×			×	×													
ARM Mali Bifrost	×	×																×		×			×														
ARM Mali Valhall	×	×																×		×			×														
Apple A13/A14 Bionic Neural Engine[a]	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
BioSEAL																		×															×				
Baidu Kunlun K200										×			×					×				×[b]															
Cambricon-X																		×			×				×												
CASCADE																		×			×[c]				×		×						×				
Cerebras WSE																		×				×			×												
Coral USB Accelerator																		×			×				×												
CSRAM																		×									×										
DaDianNao chip																×	×	×			×				×												
Darwin						×												×										×									
Darwin-WGA																		×										×									
DianNao																		×			×				×												
DIMA Inference Processor																		×							×												
DIMA-CNN																		×							×		×										
DRISA																		×													×			×			
DUAL																		×							×		×						×				
Eyeriss																		×	×						×												
Eyeriss v2																		×	×						×												
FlexFlow																		×	×						×												
FloatPIM																		×							×		×						×				
FPSA																		×							×		×				×		×				
GenCache	×																	×										×							×	
Google Pixel Visual Core																		×				×														

(*continued on next page*)

Table 7 (*continued*).

	General-purpose resources										Special-purpose resources																										
	Memory							Comp. engine			Memory								Comp. engine				Fixed-function								PL		Mem+comp.				
Accelerator	L1	L2	L3	DDR3	DDR4	LPDDR4	LPDDR4X	ARM CPU	x86 CPU	Scalar unit	GDDR6	GDDR6X	HBM	HBM2	MCDRAM	HMC-RAM	eDRAM	Local memory	Spatial array	Vector processor	Systolic array	Tensor core	Rendering pipeline engine	Physics engine	ML auxiliary-functions engine	Cryptography engine	AD/DA Converter	Compr./Decompr. engine	Alignment engine	DB Operations engine	Fine-grain	Coarse-grain	ReRAM	STT-MRAM	In-DRAM logic	In-SRAM logic	Near-memory logic
Google TPU v2, v3													×					×			×				×												
Graphcore IPU-M2000					×													×		×																	
GraphH								×								×		×																			×
Graphicionado										×							×																				
GraphP	×									×						×																					×
GraphQ	×									×						×		×																			×
GraphR																		×									×						×				
GRIM-Filter														×				×											×								×
GroqCard	×																	×	×						×												
Hailo-8																		×	×																		
Heterogeneous-PIM	×							×								×		×							×												×
HReA	×																	×	×													×					×
HRL																×		×												×	×					×
Huawei Atlas 200 AI, 300I, 300I Pro	×	×					×											×				× [d]															
Huawei Atlas 300T	×	×			×									×				×				× [d]															
Huawei Kirin 9000x, 990x NPU	×	×																×				× [d]															
IMP																		×									×						×				
Intel FPGA 10 Series														×				×													×						
Intel FPGA F-Series														×				×													×						
Intel FPGA V Series																		×													×						
Intel Graph. Tech Gen11, Xe-LP	×	×	×															×		×			×														
Intel Habana Labs HL-100					×													×		×																	
Intel Habana Labs HL-20x														×				×		×					×												
Intel Nervana NNP-I 1100, 1300	×	×	×				×		×									×		×		×			×												
Intel Nervana NN 1300, 1400														×				×				×			×												
Intel Xeon Phi Knights Landing, Mill	×	×			×				×						×					×																	

(*continued on next page*)

Table 7 (*continued*).

	General-purpose resources										Special-purpose resources																												
	Memory							Comp. engine			Memory								Comp. engine				Fixed-function								PL		Mem+comp.					
Accelerator	L1	L2	L3	DDR3	DDR4	LPDDR4	LPDDR4X	ARM CPU	x86 CPU	Scalar unit	GDDR6	GDDR6X	HBM	HBM2	MCDRAM	HMC-RAM	eDRAM	Local memory	Spatial array	Vector processor	Systolic array	Tensor core	Rendering pipeline engine	Physics engine	ML auxiliary-functions engine	Cryptography engine	AD/DA Converter	Compr./Decompr. engine	Alignment engine	DB Operations engine	Fine-grain	Coarse-grain	ReRAM	STT-MRAM	In-DRAM logic	In-SRAM logic	Near-memory logic
ISAAC																	×	×							×		×						×				
LerGAN																		×							×		×						×				
Micron Automata Proc.				×																											×				×		
Microsoft Project Catapult				×																										×							
Mixed-signal																		×									×								×		
Multibit																		×														×					
NAND-Net																		×			×														×	×		
NDA				×														×													×						×
NEC SX-Aurora TSUBASA V.E. Type10, Type20	×	×	×											×						×																	
NEST					×													×								×											×
Neural Cache																		×			×				×											×	
Neurocube	×															×		×				×			×												×
NonVolCIM																		×							×								×				
NP-CGRA																		×	×													×					
NVIDIA GeForce RTX 20xx	×	×									×			×				×		×		×	×	×													
NVIDIA GeForce RTX 30xx	×	×										×		×				×		×		×	×	×													
Origami																		×			×																
PipeLayer																		×							×		×						×				
Plasticine	×																	×													×						
PLiM																		×															×				
PracticalNDP	×	×						×								×		×															×				×
PRIME																		×							×		×						×				
PROMISE																		×									×									×	
PuDianNao																		×			×				×												
PUMA																		×		×					×		×						×				
PX-CGRA																		×	×													×					
Q100																														×							
Qualcomm Hex. 698, 780 DSP	×	×																		×																	
RADAR																		×								×		×				×				
RAPID																		×								×		×				×				
REMUS																		×	×								×					×					
RobustInMem																		×			×						×								×	

(*continued on next page*)

Table 7 (*continued*).

	General-purpose resources										Special-purpose resources																												
	Memory							Comp. engine			Memory							Comp. engine				Fixed-function								PL		Mem+comp.							
Accelerator	L1	L2	L3	DDR3	DDR4	LPDDR4	LPDDR4X	ARM CPU	x86 CPU	Scalar unit	GDDR6	GDDR6X	HBM	HBM2	MCDRAM	HMC-RAM	eDRAM	Local memory	Spatial array	Vector processor	Systolic array	Tensor core	Rendering pipeline engine	Physics engine	ML auxiliary-functions engine	Cryptography engine	AD/DA Converter	Compr./Decompr. engine	Alignment engine	DB Operations engine	Fine-grain	Coarse-grain	ReRAM	STT-MRAM	In-DRAM logic	In-SRAM logic	Near-memory logic
Samsung Exynos 9825, 990 NPU																		×			×				×													
Sandwich-RAM																		×							×											×		
ShiDianNao																		×			×				×													
Softbrain																		×	×													×						
SparseReRAM																		×							×		×						×					
Spin-transfer																		×																×				
TensorNode																		×				×																×
Tesla Full Self Driving Computer NPU																		×			×				×													
Tesseract	×	×	×					×								×																						×
TETRIS																×		×	×						×													×
Time																		×							×		×						×					
Untether TsunAImi																		×	×																			
UPMEM PIM					×					×																												×
X-CGRA																		×	×													×						
Xilinx FPGA 7 Series																		×									×				×							
Xilinx FPGA Ultrascale+ Series																		×									×				×							
Xilinx Versal ACAP								×										×	×							×	×				×							
YodaNN																		×			×																	

[a]No architectural details available.

[b]Cores with tensor and vector units.

[c]Systolic array of ReRAM arrays.

[d]Cores with tensor, vector, and scalar units.

but systolic arrays and vector processors are the most abundant. We already discussed the role of systolic arrays in ML accelerators. As for vector processors, they can be found in three variations: classic vector processors with deep pipelines and variable number of lanes (e.g., NEC SX-Aurora TSUBASA [130]), SIMD-like vector processors with a fixed number of lanes (e.g., Baidu Kunlun K200 [168], Qualcomm Hexagon DSPs [262], Intel Habana Labs [227,228]), massively multi-threaded vector cores (e.g., AMD [74], NVIDIA [135,138], Intel [105,109] GPUs). Vector processors provide a level of parallelism in the order of tens to hundreds of primitive elements. Some accelerator designs like Manycore and GPU replicate these vector cores to achieve an even higher level of parallelism. They are also employed in PUMA [256], a PIM accelerator that integrates a ReRAM crossbar and a SIMD unit to support various ML workloads.

According to our classification, spatial arrays can be found in Spatial accelerators (programmable PEs) and CGRAs (reconfigurable PEs). However, there are three notable examples: TETRIS [280], a PIM accelerator which integrates an NN engine organized as a spatial array of PEs in each vault of an HMC stack, Xilinx Versal ACAP [152], an FPGA augmented with two general-purpose cores and the so-called Intelligent Engines, arrays of interconnected VLIW and SIMD engines, and GroqCard [208]. The last has an original architecture that the authors call Tensor Streaming Processor (TSP): the accelerator's core is a heterogeneous spatial array, with its elements tiled "per role". A tile is dedicated to instruction decode and dispatch, and flow Northward towards the functional tiles [210].

The last special resource, tensor cores, are employed in various accelerators as basic building blocks in more complex structures, as well as ALUs, simple PEs, scalar, and vector units. They are present in Manycore accelerators (e.g., Baidu Kunlun K200 [168], Intel Nervana [113,118], DaVinci-powered Huawei accelerators [217,218,224]), GPUs (Turing and Ampere architectures from NVIDIA [135,138]), Spatial accelerators (Cerebras WSE [172]), and PIM accelerators (Neurocube [244], TensorNode [276]).

In today's accelerators, Programmable Logic is not limited any longer to FPGAs. Depending on the case, it may play a *primary* or an *ancillary* role. It plays a primary role in FPGAs (fine-grain PL) and in CGRAs (coarse-grain PL), where it is responsible for the computing elements and the interconnection between them. It plays an ancillary role in: DRISA [186], a PIM accelerator with reconfigurable Boolean logic operations performed in the memory cells; and the aforementioned FPSA [192] and Micron Automata Processor [231].

The abundance of PIM accelerators in Table 5 implies an abundance of "Memory and Computing" components in Table 7 that detail the way processing-in-memory is achieved. A surprising datum that emerges is that the most adopted solution to PIM is ReRAM, that is present on about 39% of PIM accelerators. Although it is an emerging technology, there are many researchers proposing its use in accelerators. All of them are academic proposals, as there are still challenges related to ReRAM employment, as we will analyze in Section 5. After ReRAM, the most common approach is the *near-memory* one, as achieved by about 31% of PIM accelerators. 60% of these rely on HMC and implement their acceleration logic in the logic die. About one fifth of PIM accelerators propose in-SRAM logic, that could be used to augment local memories and caches with computation capabilities. Cache augmentation, in particular, may even invest general-purpose processors, leading to a major leap away from von Neumann. Finally, there are only four accelerators that propose the use of in-DRAM logic and one that uses STT-MRAM.

Finally, fixed-function components are employed in more than 60% of accelerators, generally to aid programmable and reconfigurable components in task acceleration. Since the majority of accelerators are dedicated to ML workloads, the most common fixed-function components are dedicated to auxiliary ML-related functions, such as activation functions (e.g., sigmoid, tanh, ReLU). Since these functions are invoked many times in ML workloads, it is reasonable to provide them as hardware blocks even in programmable accelerators that could calculate them as sequences of arithmetic instructions.

The second most prominent fixed-function blocks are ADC/DAC converters. CSRAM [179], an in-SRAM accelerator, and ReRAM-based ones are equipped with them because they perform multiplications in the analog domain. Some ML-oriented in-SRAM accelerators perform mixed-signal multiplications between weights and data (DIMA-CNN [185], Mixed-signal [238], PROMISE [254], and RobustInMem [268]). Finally, Xilinx FPGAs [142,147,152] feature them to convert output signals into analog ones or input signals into digital ones.

3.2.4. Software aspects

Table 8 summarizes the software aspects of the identified accelerators. As for Host coupling, there are various accelerators from Academia that do not disclose this information. We make assumptions based on other characteristics of the accelerators.

Programming Layer. In this work, we confirm that "libraries are a universal 'programming model' for all kinds of accelerators" [304]. Independently of the particular approach chosen by manufacturers, it is always possible to wrap the CPU-accelerator communication logic into library calls for the most common programming languages. Moreover, there can be various layers of libraries that offer an increasing level of abstraction, with wrappers of driver functions or even inline assembly with ad-hoc instructions [10,254,275] at the bottom.

Around 41% of the accelerators in the table can be programmed with an ad-hoc high-level language. For accelerators with a high flexibility, like those that operate in Data-parallel, Dataflow, or General-purpose domains, this is the design choice that most of all gives programmer's productivity. However, it is also the most complex, as it needs at least a compiler/interpreter and a debugger. For this reason, many designers prefer simpler solutions such as library wrappers, that generally give an adequate level of abstraction for accelerators with limited flexibility.

On the opposite side of the spectrum, there are HDL and assembly. The former is used as a tool to configure the programmable logic of reconfigurable accelerators, mostly FPGAs' *fine-grain* logic. Apart from HRL, which adopts a design flow similar to FPGAs [214], most CGRAs choose lightweight procedures to reconfigure their programmable logic. Thanks to this and their shorter reconfiguration time, the reconfiguration process can be performed at runtime and included in a workflow that makes CGRAs eligible for temporal computations.

Assembly is supported in a few cases (about 9%). Most of them are Data-parallel accelerators: AMD and NVIDIA GPUs [74,135,138] (albeit as *virtual ISA* that is later translated to the target hardware ISA, generally not disclosed by vendors), NEC SX-Aurora TSUBASA [130], Baidu Kunlun K200 [168], Intel Xeon Phis [125,126], and PLiM [251]. There are also two accelerators for ML inference (i.e., PROMISE [254] and Spin-transfer [275]) and one for Dataflow (i.e., Softbrain [10]).

Ecosystem. The Ecosystem category best highlights the accelerators' heterogeneity: a plethora of frameworks, compilers, languages, and libraries populate the table. This variety involves mainly industrial accelerators, as for half of the total – all from Academia – we do not have any information. Overall, TensorFlow [56], a ML-oriented library in Python, is the most represented element, as it is supported by 14 accelerators. It is the de-facto standard for ML, albeit other solutions like PyTorch [305], ONNX [306], PaddlePaddle [307], and TensorFlow Lite [308] are also common. It is not even limited to ML workloads, as the authors of IMP [226] chose it as the programming framework for their Data-parallel PIM accelerator, observing that "TensorFlow's programming semantics is a perfect marriage of data-flow and vector-processing that can be applied to more general applications" [226].

The authors of Heterogeneous-PIM [212] do a complementary choice: they extend the OpenCL [57,58,309] programming model, which is intended for data-parallel tasks, to express NN training operations for their accelerator. Table 8 shows that OpenCL is supported across very different accelerators, including GPUs, Manycores, the

Table 8
Accelerators' software aspects [310–354].

	Programming layer					Granularity		
Accelerator	HDL	assembly	libraries	high-level languages	Ecosystem	wired task	function-level	application-level
AHA371, AHA372, AHA374, AHA378			×		gzip [310], ZLIB [311], Java API, Hadoop [312]	×		
AHA604, AHA605			×		gzip [310], ZLIB [311], OpenSSL [313]	×		
AMD Radeon RX 5000 Series, 6000 Series		×	×	×	OpenCL [57,58,309], ROCm [314], Sycl [59], Kokkos [355], OpenMP4+ [60], OpenACC [315], SkelCL [356], StarPU [316], Halide [317], OmpSs [318], OpenGL [357], Vulkan [63], DirectX [358], Metal [319]		×	
ARM Mali Bifrost, Valhall			×	×	OpenCL [57,58,309], Sycl [59], OpenGL [357], Vulkan [63]		×	
Apple A13 Bionic Neural Engine, A14 Bionic Neural Engine			×		TensorFlow Lite [308], PyTorch [305], Core ML [320]	×		
Baidu Kunlun K200		×	×	×	XTCL, XTDK [167], TensorFlow [56], PaddlePaddle [307], PyTorch [305]		×	
BioSEAL			×	×		×		
Cambricon-X			×		Caffe [64]	×		
CASCADE			×			×		
Cerebras WSE			×	×	Cerebras SDK [321], TensorFlow [56], PyTorch [305]		×	
Coral USB Accelerator			×		TensorFlow Lite [308], PyCoral [73]	×		
CSRAM			×			×		
DaDianNao chip			×			×		
Darwin			×			×		
Darwin-WGA			×			×		
DianNao			×			×		
DIMA Inference Processor			×			×		
DIMA-CNN			×			×		
DRISA			×	×			×	
DUAL			×			×		
Eyeriss			×		Caffe [64]	×		
Eyeriss v2			×			×		
FlexFlow			×			×		
FloatPIM			×			×		
FPSA			×	×	NN compiler [322]		×	
GenCache			×			×		
Google Pixel Visual Core			×	×	Halide [317], TensorFlow [56], Android Camera API [323]		×	
Google TPU v2, v3			×		TensorFlow [56], scikit-learn [324], XGBoost [325], Keras [359]		×	
Graphcore IPU-M2000			×		Poplar SDK [326], TensorFlow [56], ONNX [306], PaddlePaddle [307], PyTorch [305], Keras [359]		×	
GraphH			×	×			×	
Graphicionado			×	×	GraphMath [327]		×	
GraphP			×	×			×	
GraphQ			×	×			×	
GraphR			×			×		
GRIM-Filter			×			×		
GroqCard			×		TensorFlow [56]	×		
Hailo-8			×		TensorFlow [56], ONNX [306], AI SDK [328]	×		
Heterogeneous-PIM			×	×	OpenCL [57,58,309]		×	
HReA			×				×	
HRL	×		×		Verilog [61]		×	
Huawei Atlas 200 AI, 300I, 300I Pro, 300T			×	×	CANN [329], MindSpore [330], TensorFlow [56], PyTorch [305], PaddlePaddle [307], MindX SDK [331]		×	
Huawei Kirin 990x NPU, 9000x NPU			×	×	HiAI DDK [72], CANN [329], TensorFlow Lite [308], Android NNAPI [332], MindSpore [330], PaddlePaddle [307]		×	
IMP			×		TensorFlow [56]		×	
Intel FPGA V Series, 10 Series, F-Series	×		×	×	Intel Quartus Prime [71], DSP Builder [333], Intel HLS [334], Vivado Design Suite [335], OpenCL [57,58,309]		×	
Intel Graphics Technology Gen11, Xe-LP			×	×	OpenCL [57,58,309], Sycl [59], OpenMP4+[60], OpenACC [315], SkelCL [356], StarPU [316], Halide [317], OmpSs [318], OpenGL [357], Vulkan [63], DirectX [336]		×	
Intel Habana Labs HL-100			×	×	TensorFlow [56], PyTorch [305], ONNX [306], MXNet [337], Glow ML Compiler [338], Synapse AI Suite [227]		×	
Intel Habana Labs HL-20x			×	×	TensorFlow [56], SynapseAI Suite [228]		×	
Intel Nervana NNP-I 1100, 1300			×	×	TensorFlow [56], OpenVINO [339], PyTorch [305], nGraph [340,341], ONNX [306]		×	
Intel Nervana NNP-T 1300, 1400			×	×	TensorFlow [56], PaddlePaddle [307], PyTorch [305], nGraph [340,341]		×	

(*continued on next page*)

Table 8 (*continued*).

Accelerator	Programming layer				Ecosystem	Granularity		
	HDL	assembly	libraries	high-level languages		wired task	function-level	application-level
Intel Xeon Phi Knights Landing, Knights Mill		×	×	×	Standard x86 tools, OpenCL [57,58,309], Sycl [59], OpenMP [60], OpenACC [315], SkelCL [356], StarPU [316], Halide [317], OmpSs [318]		×	×
ISAAC			×			×		
LerGAN			×	×			×	
Micron Automata Processor			×	×	ANML [234], AP SDK, RAPID [342]		×	
Microsoft Project Catapult	×		×		Verilog [61], VHDL [62,343]	×		
Mixed-signal			×			×		
Multibit			×			×		
NAND-Net			×			×		
NDA			×				×	
NEC SX-Aurora TSUBASA Vector Engine Type10, Type20		×	×	×	VE offloading C-API, NEC SDK [344]			×
NEST			×			×		
Neural Cache			×			×		
Neurocube			×			×		
NonVolCIM			×			×		
NP-CGRA			×			×		
NVIDIA GeForce RTX 20xx, 30xx		×	×	×	CUDA [65,345], OpenCL [57,58,309], Sycl [59], PHAST [360], Kokkos [355], OpenMP4+[60], OpenACC [315], SkelCL [356], StarPU [316], Halide [317], OmpSs [318], Matlab [346], OpenGL [357], Vulkan [63], DirectX [347], Metal [319]		×	
Origami	–[a]	–[a]	–[a]	–[a]		×		
PipeLayer			×			×		
Plasticine			×	×	Delite Hardware Definition Language [348]		×	
PLiM		×	×				×	
PracticalNDP			×	×	Phoenix++[349]		×	
PRIME			×	×			×	
PROMISE		×	×	×	MXNet [337], Flux [350,351]		×	
PuDianNao			×			×		
PUMA			×		Caffe [64], PyTorch [305], MXNet [337], ONNX [306]		×	
PX-CGRA			×	×	Matlab [287]		×	
Q100			×	×	SQL	×		
Qualcomm Hexagon 698 DSP, 780 DSP			×	×	Hexagon DSP SDK [66], Android NN API [332], TensorFlow Lite [308], Qualcomm Neural Processing SDK [67], ONNX [306], PyTorch [305]		×	
RADAR			×			×		
RAPID			×			×		
REMUS			×	×			×	
RobustInMem			×			×		
Samsung Exynos 9825 NPU, 990 NPU			×		Android NN API [332], TensorFlow Lite [308], Samsung Neural SDK [68]	×		
Sandwich-RAM			×			×		
ShiDianNao			×			×		
Softbrain		×	×	×			×	
SparseReRAM			×			×		
Spin-transfer		×	×			×		
TensorNode			×			×		
Tesla Full Self Driving Computer NPU			×		NN compiler	×		
TETRIS			×			×		
Tesseract			×	×			×	
Time			×			×		
Untether TsunAImi			×		TensorFlow [56], PyTorch [305], imAIgine SDK [282]		×	
UPMEM PIM			×	×	UPMEM SDK [69]		×	
X-CGRA			×	×	Matlab [287]		×	
Xilinx FPGA 7 Series, Ultrascale+ Series	×		×	×	Vitis Unified Software Platform [70], Vivado Design Suite [352], OpenCL [57,58,309], Verilog [61,353], VHDL [62,343], Matlab [354]		×	
Xilinx Versal ACAP	×		×	×	Vitis Unified Software Platform [70], Vivado Design Suite [352], OpenCL [57,58,309], Verilog [61,353], VHDL [62,343]		×	×
YodaNN			×			×		

[a] It depends on the attached FPGA.

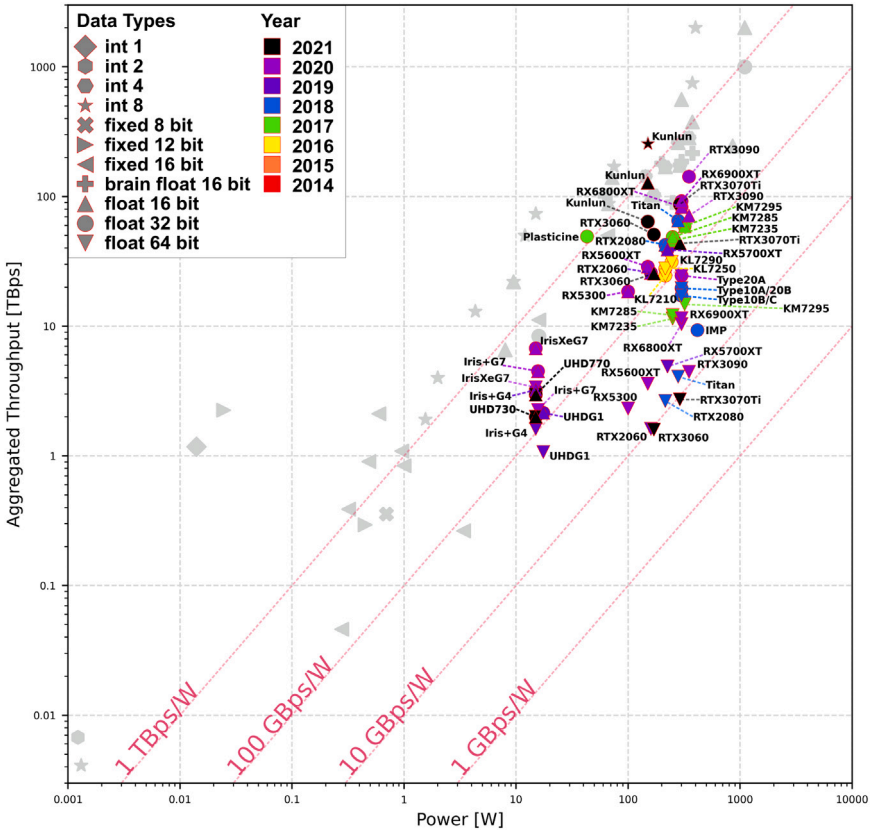


Fig. 8. Power consumption (Watt) vs aggregated throughput (TOPS) for data-parallel accelerators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

aforementioned PIM accelerator, and even FPGAs. In the last case, it is used as a high-level synthesis tool, as the kernels expressed in this language are compiled into a binary that can be used to reconfigure the programmable logic of these accelerators.

Other more traditional options are also present in the table, like Verilog [61] and VHDL [62]. Finally, the type of accelerators with the richest ecosystem is by far GPUs. Graphics-oriented APIs like OpenGL [357], Vulkan [63], and DirectX [358] are flanked by purely data-parallel solutions like the aforementioned OpenCL, but also Sycl [59] and OpenMP [60].

Granularity. The granularity category shows a strong correlation with the programming layer. Accelerators that support high-level programming languages have at least a function-level granularity. However, it is not true the opposite, as this can be achieved by the means of reconfigurable logic, but also with libraries only, as five non-reconfigurable accelerators testify. In these cases, function-level granularity is achieved by specifying a callback that is compiled JIT and executed on the accelerator, e.g., TensorFlow's custom activation functions. Conversely, reconfigurable accelerators can employ HDL and libraries, the latter mainly to write configuration data in CGRAs at runtime.

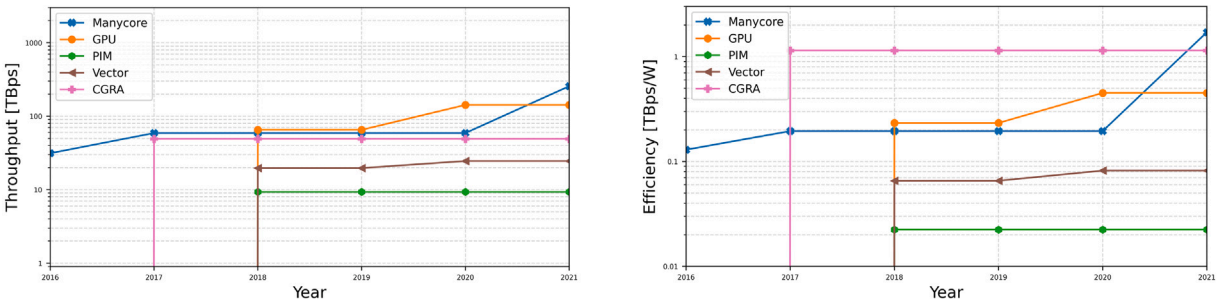
Wired task granularity is the most common, with around 52% of accelerators supporting it. It can refer to simple operations, equivalent to common instructions such as LOAD or ADD, but also to more complex ones that trigger a non-trivial computation on the accelerator. So, the remaining 48% of accelerators support at least function-level granularity. This datum highlights that accelerators, despite their domain-specific nature, are generally *flexible* rather than *inflexible* devices. This may sound surprising: accelerators, after all, pursue high throughput and efficiency, that find their maximum realization in ASICs. However, being *inflexible* may limit too much, in the majority of cases, the set of acceleratable workloads — which is hardly convenient from the users' perspectives outside of very selected and critical contexts.

Finally, three accelerators support application-level granularity: Xilinx Versal ACAP [152], accelerators in the Xeon Phi family [125,126], and NEC SX-Aurora TSUBASA Vector Engines [130]. The first two can be used in a mode that Intel calls *native mode* (opposed to *offload mode*). In this mode, they can operate as stand-alone nodes and take care also of the prerogatives commonly reserved to the host. The last one achieves application-level granularity thanks to the "reverse offloading" mechanism described before.

4. Performance study

Due to the variety and heterogeneity of the presented accelerators, we cannot define a *performance* metric or a set of metrics that are descriptive of all of them. Performance can refer to throughput, latency, memory bandwidth, efficiency, etc. Some of them are not available for every accelerator, whether because they do not apply or because they are not disclosed by vendors. However, a common metric that can be defined for basically every accelerator is *throughput* – but also in this case, there is no homogeneity in what it refers to. First, *throughput* can refer to the number of operations performed by all the units in the accelerator (ALUs, FPUs, PEs, etc.) per second or the number of "artifacts" produced *in output* per second (images per second, compressed bytes per second, etc.). We define the former *aggregated throughput*, and the latter *output throughput*. Second, even interpreting throughput the same way, it can be expressed with different units of measure: it can be given in FP32 FLOPS, FP64 FLOPS, Gbps, images per second, and so on, depending on the domain in which the accelerator operates, the data types the accelerator supports, and the *taste* of the vendor in terms of what characteristic they prefer to highlight.

Aggregated throughput metrics are generally available for accelerators in the ML inference, ML training, and Data-parallel domains. These values are expressed in FLOPS or OPS, but can be easily normalized



(a) Throughput (b) Efficiency

Fig. 9. Maximum performance achieved until each year for each type of data-parallel accelerator.

and compared each other by multiplying the metric by the number of bytes of the processed data type, obtaining a Bps (Bytes per second) value. Unfortunately, output throughput metrics, where available, are highly heterogeneous and cannot be normalized, even for accelerators in the same domain. Therefore, in this paper, we focus our analysis on aggregated throughput expressed in Bps for accelerators in the ML inference, ML training, and Data-parallel domains. We relate this performance to the power dissipated at full utilization, for which we take the Thermal Design Power (TDP) as an estimation. This way, we frame accelerators from an *efficiency* standpoint, intended as the achievable throughput per power unit or, equivalently, number of bytes processed per energy unit.

Some accelerators categorized so far are presented as families, rather than single models. Even in the same family, different models can perform very differently. Therefore, in the figures in this section we show the achieved performance of accelerator *models*. The association between families and models is displayed in Table 2.

4.1. Data-parallel

Fig. 8 shows the relation between power consumption and *normalized aggregated throughput* (in the following: throughput) in data-parallel accelerators. These include GPUs, Manycores, Vectors, one PIM, and one CGRA. We can identify two main groups of accelerators: laptop and desktop/server accelerators, with power in the range 10–50 W and 50–500 W, respectively. All accelerators can be easily assigned to one group or the other, with the only exception of Plasticine [250], at the boundary with 43 W. The figure clearly displays the strong bond that exists between power consumption and achievable throughput, with accelerators in the lower (higher) power group generally achieving lower (higher) throughput.

All the GPUs except Intel's UHD730 [110] and UHD770 [111] have higher FP16 and FP32 throughput than FP64. Generally, FP64 units are very few: for instance, on NVIDIA Ampere GPUs, each Streaming Multiprocessor features 128 FP32 cores and only 2 FP64 cores [138]. There are two complementary reasons behind this design choice. On the one hand, for most workloads commonly executed on GPUs, a 64-bit precision is not needed (e.g., graphics). On the other, the need for FP16 support is increasing: according to a significant body of work, in ML workloads, precision can be safely reduced with small accuracy losses and high gains in performance, memory bandwidth, and storage cost [21]. Many GPU families have the same FP16 and FP32 throughput (e.g., NVIDIA RTX 20xx, AMD RX5xxx, AMD RX6xxx): their FPUs support packed data, so they are able to perform one 32-bit operation or two 16-bit operations together [74,135]. NVIDIA made a different design decision in its most recent generation (30xx): FP32 throughput is twice the FP16 throughput — which means that the correspondent non-normalized FLOPS values are the same.

The ongoing investment in small data-types (and reduction in FP64 support) is evident also in the Intel Xeon Phi family. Knights Landing models [122–124] hit the market in 2016 with the same FP32 and FP64 throughput and no native support for FP16. In the following generation, which debuted in 2017, Knights Mill accelerators [127–129] had the same FP16 and FP32 throughput, and 1/4 FP64 throughput. At the time of writing, it is rumored that upcoming Intel Alchemist GPUs will skip FP64 support altogether [361].

NEC SX-Aurora TSUBASA models [130], that do not target explicitly ML models, still in 2019 have no native support for FP16 and have the same FP32 and FP64 throughput. In this case, 64-bit entries from vector registers can be treated as two packed 32-bit floats, which are processed together by double precision units.

Baidu Kunlun K200 [168] has Data-parallel capabilities, but it is advertised mostly as an ML-oriented accelerator. It follows the general tendency towards precision: maximum throughput is achieved with INT8 operations (256 TBps), followed by FP16 (128 TBps), and FP32 (64 TBps).

In Fig. 8, there are only two academic proposals: IMP [226], a PIM accelerator, and Plasticine [250], a CGRA. Considering that academic proposals are generally prototypes, it is even more remarkable that Plasticine achieves about the same FP32 throughput as an RTX3060 GPU with about 25% its power consumption – i.e., it is 4× as efficient. Since it is the only CGRA in the figure, we cannot generalize and affirm that this is due to the high efficiency of this type of accelerators compared to GPUs. However, such a claim would be in line with other studies that highlight how CGRAs generally achieve better efficiency than other architectures (except ASICs) [10,18,214,241].

Overall, Baidu Kunlun K200 [168] achieves the maximum INT8 throughput (256.00 TBps) and FP16 throughput (128.00 TBps), and NVIDIA RTX3090 [138] the maximum FP32 throughput (142.40 TBps). As for efficiency, Baidu Kunlun K200 [168] achieves the maximum INT8 efficiency (1.707 TBps/W) and FP16 efficiency (0.852 TBps/W), and Plasticine [250] the maximum FP32 efficiency (1.144 TBps/W).

Fig. 9 shows the throughput and efficiency trends in data-parallel accelerators, grouped by type. For each year, it shows the maximum performance achieved until that year. In formulas, the value V assumed by the curve associated to data-parallel accelerators of type T the year y_0 is calculated as:

$$V_T(y_0) = \max_{\substack{A \in \mathcal{A}^{DP}_{T,y} \\ y \le y_0}} Perf_A$$

where $Perf_A$ is the performance achieved by accelerator A, and $\mathcal{A}^{DP}_{T,y}$ is the set of data-parallel accelerators of type T proposed in the year y. $Perf$ may refer to throughput (Fig. 9(a)) or efficiency (Fig. 9(b)).

Fig. 9(a) shows that Manycore accelerators achieve the highest throughput for data-parallel workloads. GPUs were the type with the highest throughput from 2018 to 2020. In 2021, the introduction of Baidu Kunlun K200 [168] overturned the situation, surpassing the best GPU by 79.77% and increasing the maximum throughput achieved by Manycore accelerators by 4.34× with respect to the previous best performing one, Intel Xeon Phi KM7295 [129]. The superiority of Manycores and GPUs – which are closely related, as already explained

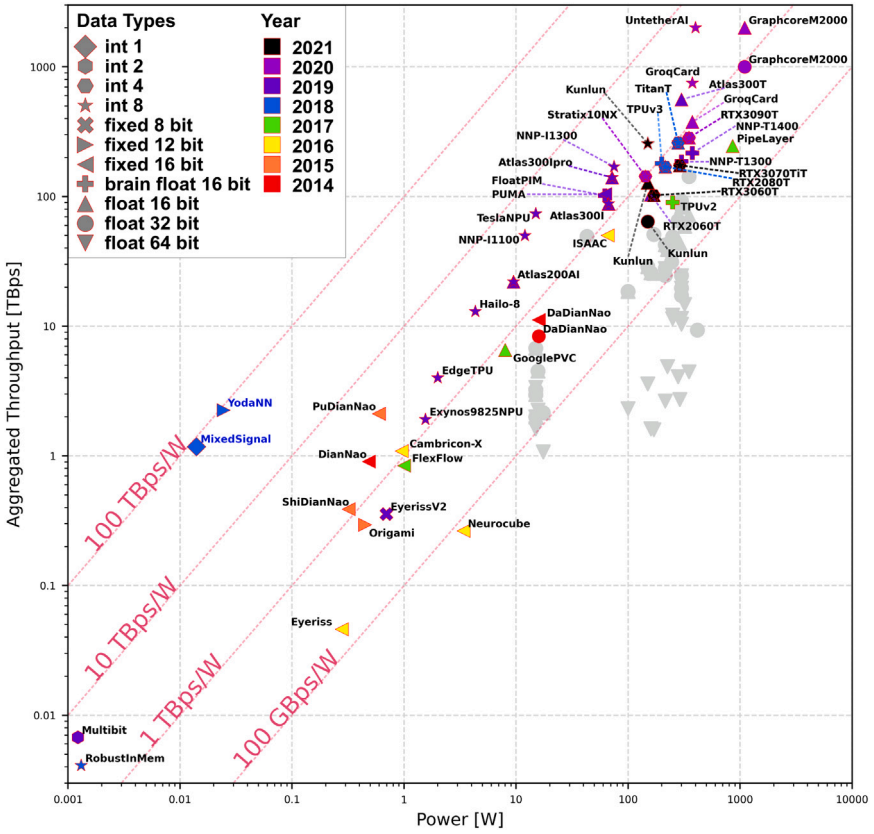


Fig. 10. Power consumption (Watt) vs aggregated throughput (TOPS) for ML accelerators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

– seems to suggest that this is the way to design high-throughput data-parallel accelerators.

On the efficiency side, Fig. 9(b) highlights a different situation: Plasticine's introduction in 2017 [250] improved the maximum efficiency of about one order of magnitude. In the period 2017–2020, two generations of GPUs were introduced in the market. The maximum efficiency doubled in the last one, but they achieved at most 0.39× Plasticine's efficiency. Since these GPUs are destined to laptops, data-centers, and supercomputers, we think that their designers took efficiency in high consideration – albeit less than throughput. In 2021, Baidu Kunlun K200 [168] was introduced and beat Plasticine's efficiency by 49%. In our opinion, the fact that it took 4 years for an industrial manycore accelerator to score a better efficiency than an academic CGRA can be read as a proof of the superiority of the CGRA approach in terms of efficiency and how promising it can potentially be.

In both figures, the PIM type achieves the worst throughput and efficiency. In our opinion, this is due to the fact that there is only one PIM, IMP [226], which may be not enough to represent its category. We will show in the next Subsection that PIM accelerators too may be highly efficient.

4.2. Machine learning

Fig. 10 shows the relation between power consumption and throughput in ML accelerators. There are GPUs, Manycores, Systolic, Spatial, PIM accelerators and one FPGA. Mixed-signal [238] and YodaNN [288] are displayed in blue because, since they target BNNs, they replace MAC units with complement units and multiplexers and their throughput is calculated based on simpler operations than multiplications. For this reason, we display them in the figure, but avoid any comparison with other accelerators that would be unfair.

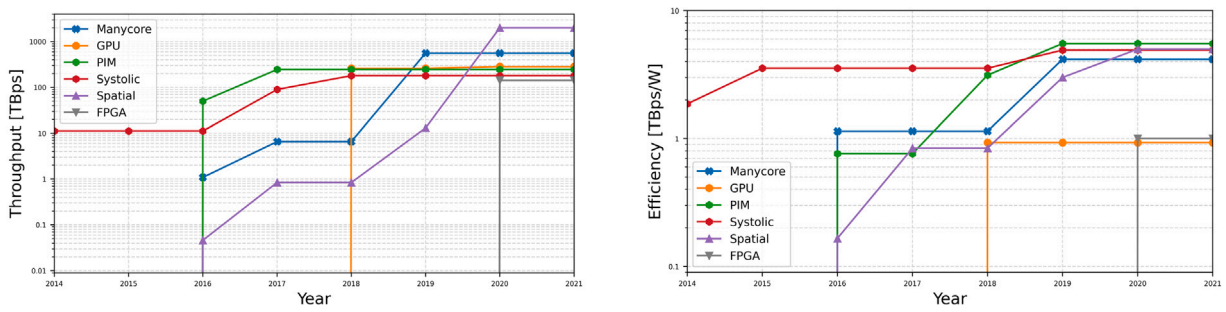
In this Figure, as much as five power-dependent groups can be identified:

- Very-low: less than 0.2 W;
- Mobile: range 0.2–10 W;
- Laptop: range 10–50 W;
- Desktop/server: range 50–500 W;
- High-end: more than 500 W.

Again, each of them can be easily assigned to a group, with some ambiguity for Atlas 200 AI [215] and Intel Nervana NNP-I 1100 [116], with 9.5 W and 12 W, respectively. It is interesting to note that laptop and desktop/server accelerators achieve higher throughput than their data-parallel counterparts, displayed in gray in the background of the figure. Even DaDianNao [180], proposed in 2014, achieves higher throughput than the most recent laptop GPUs, basically with the same power budget. This happens because ML-oriented accelerators are tailored for a narrower domain than data-parallel. Their designers can pursue specialization more aggressively, reducing flexibility and replacing control-flow units with purely functional ones.

As expected, the tendency of lowering data-type precision to seek throughput and efficiency is even higher in accelerators that explicitly target ML workloads. In this case, the native data-types supported are as low as single bits (INT1), the majority of them employ at most 16-bit data-types, and the few with 32-bit data-types (DaDianNao [180], Baidu Kunlun K200 [168], and Graphcore M2000 [201]) achieve less than 1 TBps/W efficiency. No accelerator manipulates FP64 data-types. INT8 is the preferred data-type by many recent accelerators (2018 and later) in the efficiency range 1 TBps/W–10 TBps/W. 16-bit fixed-point, conversely, has no support in accelerators proposed after 2016.

The only accelerators displayed in both Figs. 8 and 10 are Baidu Kunlun K200 [168], described before, and NVIDIA GPUs. The latter, despite being Data-parallel accelerators, are widespread also to perform ML tasks, even at data-center scale. In Fig. 10, we display the throughput achieved by their Tensor Cores [135,138], advertised as "specialized execution units designed specifically for performing the



(a) Throughput (b) Efficiency

Fig. 11. Maximum performance achieved until each year for each type of Machine Learning accelerator.

tensor/matrix operations that are the core compute function used in Deep Learning" [135]. In both 20xx and 30xx generations, they are able to perform INT4, INT8, and FP16 calculations with the same throughput. We think this is due, also in this case, to the ability of using packed data.

Intel Stratix 10 NX FPGA [362] is the only FPGA in the figures. FPGAs' achieved throughput and power consumption greatly depend on the configuration adopted, and a general discussion cannot be done regardless of it. However, this FPGA features AI-optimized compute blocks and Intel discloses all the relevant data (data-type, aggregated throughput, and efficiency).

The maximum throughput achieved are: Untether TsunAImi [282] (2.00 PBps, INT8), Graphcore IPU-M2000 [201] (2.00 PBps, FP16 and 1.00 PBps, FP32), Geforce RTX3090 Tensor Core [141] (284.00 TBps INT4), Intel Nervana NNP-T 1400 [121] (216.00 BF16), PUMA [256] (104.62 TBps, FixP16), Eyeriss V2 [189] (356 GBps, FixP8), Origami [247] (294 GBps, FixP12), Multibit [239] (7.00 GBps, INT2).

The maximum efficiencies achieved are: Multibit [239] (5.533 TBps/W, INT2), Untether TsunAImi [282] (5.00 TBps/W, INT8), PuDianNao [255] (3.544 TBps/W, FixP16), Atlas 200 AI [215] (2.316 TBps/W, FP16), FloatPIM [191] (1.637 TBps/W, BF16), Intel Stratix 10 NX FPGA [362] (1.00 TBps/W, INT4), Graphcore IPU-M2000 [201] (909 GBps/W, FP32), Origami [247] (655 GBps, FixP12), Eyeriss V2 (511 GBps/W, FixP8).

Fig. 11 shows the throughput and efficiency trends in ML accelerators, grouped by type. It is analogous to Fig. 9 for ML accelerators. The value V assumed by the curve associated to ML accelerators of type T the year y_0 is calculated as:

$$V_T(y_0) = \max_{\substack{A \in \mathcal{A}^{ML}_{T,y} \\ y \le y_0}} Perf_A$$

with $Perf_A$ with the same meaning as before and $\mathcal{A}^{ML}_{T,y}$ denoting the set of ML accelerators of type T proposed in the year y.

Fig. 11(a) shows that, as today, the best throughput is achieved by Spatial accelerators. This is true since 2020, when Untether TsunAImi [282] was proposed and improved the best throughput by 3.57×. Until the year before, the best performing accelerator was the Manycore Huawei Atlas 300T [220], but the year before the record was contended between PIMs and GPUs, with Systolic accelerators just behind. This turnover seems to suggest that, in the case of ML accelerators, there is not a neat distinction between the achievable capabilities of various types of accelerators. More than one candidate seems to be a good fit to deliver high throughput.

Fig. 11(b) shows the efficiency trend. Also in this case, there is no neat distinction: the best efficiency is achieved by PIM accelerators, that are competitive from 2018, when RobustInMem [268] was proposed. It scored an efficiency 11.77% lower than the most efficient accelerator at the time, the Systolic PuDianNao [255]. In 2019, the PIM Multibit [239] and the Systolic Tesla FSD Computer NPU [277] were proposed, with the former still being the most efficient ML accelerator, with 5.533 TBps/W. In 2020, Untether TsunAImi [282] came closer with 5.00 TBps/W. Overall, four accelerator types are close together with efficiencies in the range 4–6 TBps/W: PIM, Spatial, Systolic, Manycore.

5. Open challenges

In this Section, we discuss some open challenges that affect accelerated architectures horizontally. For each of them, we describe the problem in order to let readers grasp its characteristics and why it matters. Then, we discuss some state-of-the-art techniques adopted to address it and also some prospective research directions.

5.1. Tackling the memory wall problem

Since its formulation in 1995, the memory wall problem [300] has been dominating the debate. It predicted the widening of the performance gap between memory system and processor, and the prediction proved correct over the years. Today's processors suffer the limitations imposed by the bandwidth and latency of the memory system to the rate at which they can consume data and instructions. The performance gap has become even more dramatic with the advent of highly parallel systems, due to the consequent narrowing of the channel to the memory reserved for each computing element [363]. Moreover, also from an energy point view, energy consumption in today's systems is dominated by moving data back and forth rather than consuming them.

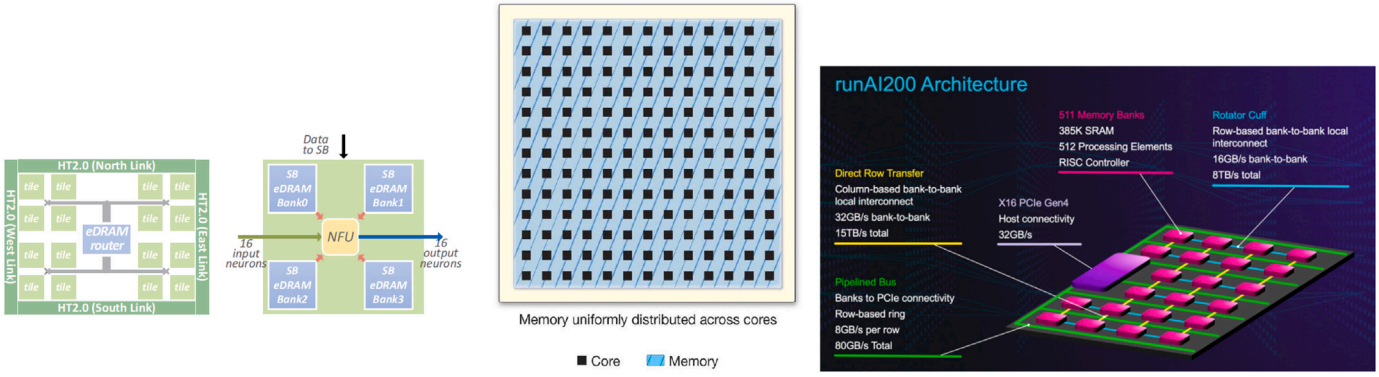
Traditional solutions concentrated on limiting the impact of the problem, addressing it by improving both the memory subsystem, e.g., speculative load, prefetching techniques, and miss rate reduction; and the processor, e.g., investing in thread-level parallelism as a way to have "work to do", so to avoid cache miss stalls. Unfortunately, not only multiprocessors are affected: every computing device that needs data or instructions from a *distant* memory resource may suffer the performance gap between its compute elements and the memory system providing operands. If a *culprit* is to be found, it must be identified in the *distance* between processing elements and memory that is dictated by the von Neumann model.

In the following, we analyze two main techniques adopted by accelerator designers to address the memory wall problem, both sharing the need to move away from the von Neumann model.

5.1.1. State-of-the-art solution: Memory-rich processors

The first approach is the so-called *memory-rich processor*, which consists in bringing as much memory on-chip as possible, closer to compute units. This same principle inspires the abundance of local memories in accelerated architectures and the presence of ever larger cache memories in general-purpose processors[2]: the integration of more

[2] At the time of writing, AMD is in the process of releasing EPYC 7003X "Milan-X" server CPUs with 768 MB L3 cache [364].



(a) DaDianNao [180] (b) Cerebras WSE [174] (c) Untether TsunAImi [282]

Fig. 12. Accelerators designed according to the *memory-rich processor* principle.

and more high-bandwidth, low-latency memory closer to compute units reduces the frequency of transfers from/to the main memory, improving efficiency, latency, and performance.

Several accelerators are designed according to this principle. For instance, the Manycores DaDianNao [180] and Graphcore IPU-M2000 [201], and the Spatial accelerators Cerebras WSE [172], GroqCard [208], and Untether TsunAImi [282]. Fig. 12 shows block diagrams of DaDianNao, Cerebras WSE, and Untether TsunAImi taken from original papers, whitepapers, and presentations. Manycore and Spatial accelerators have in common the presence on-chip of *many* PEs. In the Spatial case, these are interconnected and can exchange data directly. The *memory-rich* principle applied to these accelerators involves the addition of abundant on-chip memory that can be partitioned between PEs or not.

With memory-rich accelerators, the need to exchange data with the main memory is not eliminated. On-chip memory needs to be fed with input data to process, instructions need to reach the PEs, and output results need to be written back to the main memory. There is still the need to overlap processing and data movement, with the latter potentially limiting the performance of the accelerator.

In these architectures, achieving an efficient use of *on-chip* memory is a challenge per se. If on-chip memory is partitioned between PEs, data movements must happen explicitly (e.g., with message passing between PEs), but it could happen transparently in a shared memory design. In both cases, data movements can lead to performance penalties, especially when data accessed are physically distant from the accessing PE. More or less, these systems present the same challenges that characterize distributed systems: each PE should have its data as close as possible (inside its partition if the memory is partitioned), and data should move as little as possible. Depending on the programming model, achieving this may be the programmer's responsibility or the effect of an automatic procedure (either optimal or heuristic-based) as part of a building process. The latter is a common solution for industrial accelerators: manufacturers usually provide it as part of ad-hoc backends to popular frameworks/libraries (e.g., TensorFlow [56], PyTorch [305]). In any case, both programmer-driven or compiler-driven solutions can work if the accelerated application exposes a regular access pattern to the memory, so that either the programmer or compiler can arrange computations in a way that minimizes data transits. Conversely, applications with an intrinsically irregular data access pattern, *data-dependent* in the worst case – thus, not knowable in advance – represent a big challenge for these architectures as much as for traditional ones.

5.1.2. State-of-the-art solution: PIM

PIM is a complementary approach with respect to Memory-rich processors. It is based on bringing computation closer to the memory to reduce the energy needed to move data and increase bandwidth. It features two variations: *near-memory* computing is obtained by placing compute elements as close as possible to memory banks, being them main memory, caches, even scratchpad memories; *in-memory* computing is obtained by fusing in the same component memory and computing capabilities, either by augmenting traditional technologies with computing units or by adopting emerging technologies that naturally provide both.

The PIM idea is not new, as early proposals date back to 1970 [365]. It was discussed again in the 1990s, but "at the time, the industry moved towards increasing the off-chip memory bandwidth instead of adopting the PIM concept due to costly integration of computation units inside memory dies" [279]. That cost was drastically reduced in 2010s thanks to the emergence of 3D stacking technologies, which allow memory and computing dies to be stacked together and exchange data at high speed thanks to through silicon vias [23]. Apart from this, other two factors contribute to the PIM blooming we witness these days: the emergence of novel technologies that blur the line between memory and computation, like memristors, and the *experimentalism* hardware designers had to embrace as a byproduct and a reaction to the end of frequency scaling and, in general, the slowdown in general-purpose processors' improvement.

The goal of near-data and in-DRAM accelerators is to reduce drastically data transfers between CPU and main memory, which are the most expensive ones from both latency and energy points of view. They cannot be eliminated altogether, since CPU likely continues to be employed to program PEs and offload work to them — as it happens in the Heterogeneous-PIM [212] case, that is programmed in OpenCL and receives work by a host-operated profiling-based scheduler. Therefore, if not *data*, at least *instructions* need to reach PEs [366]. However, both DRAM and 3D-stacked technologies (i.e., HMC, HBM, HBM2) have a modular design that affects the architecture of accelerators based on them: single PEs (that can be as small as few logic ports) are local to single modules – *banks* in DRAM and *vaults* in 3D-stacked memories. For this reason, also PIM accelerators are similar to distributed systems: each PE's processing should be limited to data contained in the same module as much as possible, as communication across different modules can be expensive and have a big impact on performance and efficiency. The same considerations about programming models for memory-rich accelerators apply also in this case, and, unfortunately, also the observation that applications with irregular or data-dependent access patterns are not a good fit for these accelerators.

There are some specific challenges related to the technologies used to achieve PIM. Memristors, which are the heart of ReRAM-based in-memory accelerators, suffer from various issues: low precision [21, 206], ADC/DAC overhead [21], durability issues related to write operations [265].

Traditional technologies present their challenges as well. In-SRAM computing is performed by activating two or more SRAM word-lines

and sensing on the pair of bit-lines [23]. Logical operations between word-lines can be implemented easily, but more complex ones like arithmetic operations are *bit-serial* instead of *bit-parallel*, which increases the number of cycles needed to perform an operation. An alternative is to rely on mixed-signal computing (e.g., PROMISE [254], Mixed-signal [238]), that mixes digital and analog signals, but it works well only for reduced precision, needs costly ADCs, and requires modifications that decrease the memory density [23]. In-DRAM computing is also achieved by activating more word-lines and reading the outcome in the bit-line charges. This technique usually requires circuit modification, supports limited operations (AND and OR), is sensitive to charge variance, and is destructive of the word contents [23]. However, in-DRAM computing has proved *historically* difficult because the integrated computing logic must be implemented with the DRAM process, which produces slower processing elements with respect to CPUs. This, unfortunately, is a problem common to all near-memory and in-memory proposals, as no process is as optimized for speed as CPU process. A reduced speed in computing logic may risk nullifying the performance gains of an even drastically reduced data movement, and this is even more true when complex operations that require many cycles are involved.

5.1.3. Perspective: PIM consolidation and new technologies

Memory-rich and PIM accelerators, both near-memory and in-memory, are already well-represented in the accelerator world. We argue that their presence in both market and academia will steadily increase in the next years as technological scaling will keep on exacerbating the memory wall problem, and this processing paradigm appears to be one of the main broad directions for stepping away from von Neumann architectures. However, these solutions are currently in their infancy, and need further research to reach maturity from both technological and software perspectives.

The biggest technological challenge concerns the difficulty for PIM accelerators to exploit fast processing elements, which is determined by the different processes adopted for memory technologies and processors. Since the former is optimized for memory density and not for speed, it is a poor choice for computing logic. As PIM proposals become more and more widespread, we can expect novel techniques to be explored solely for this purpose, possibly looking for a trade-off between the two that may pursue higher computing speed at the expense of memory density, positioning “in the middle” between the two. In the case of 3D-stacked memories like HMC or HBM, it would be sufficient that this speed-oriented process is limited to the logic die.

Much research is addressing the shortcomings of specific technologies. Memristors’ low-precision issues can be mitigated by adopting the *splicing* method, which consists in employing multiple cells for different bits, later combined in high-precision numbers through shift–add operations [192]. The durability issue is due to the high voltage necessary for writes [171], and can be mitigated by lowering the voltage. In turn, this would make writes less deterministic [171], so this solution is viable only for applications that can tolerate the resulting errors. In-DRAM related issues are also being addressed at various levels [23,186,367,368]. Overall, there is much research that aims at limit the shortcomings of particular technologies, and we can expect it to continue and find increasingly effective solutions.

From a programming point of view, there is a need for sophisticated programming tools targeting both memory-rich and PIM accelerators. These should address the similarities between these architectures and distributed systems, leveraging many techniques that maturated in those contexts, but at a drastically different scale: minimizing data exchange between processing nodes, promoting data reuse at node level, allocating frequently-communicating logical nodes on neighboring physical nodes, pursuing load balancing, etc. An important difference to consider is that, with respect to nodes commonly found in distributed systems, these nodes could have severe limitations (e.g., they could be very simple PEs featuring only few logical ports) and could be unable to run even simple peripheral daemons/processes that may contribute in system management. Depending on the characteristics of the accelerator, such management could be necessarily centralized. However, even considering these differences, we think that future research should investigate strategies to map the *best practices* implemented in tools such as Slurm [369] or Kubernetes [370] on these accelerators.

Other promising research perspectives come from technologies that have the potential to eliminate the performance and energy gap between the memory system and the computing elements. In the shorter term, some breakthrough technologies are on their roadmap for integration into mainstream products, and *silicon photonics* is one of them. It promises to reduce of around one order magnitude the physical communication latency on chip, thus making units, modules, zones of chips *closer in time*. This can, from one side, reduce the magnitude of nowadays wire-delay effect, re-enabling some design choices and, on the other side, might foster also new solutions otherwise unfeasible [371].

For instance, one of the key cornerstones of HP’s The Machine [372] proposal was a notable usage of photonics to bridge distances and glue-up the machine at various levels. Silicon-photonics interposers are present in the technological roadmaps since more than 10 years and are getting closer and closer to adoption. We can expect silicon photonics to potentially be a key ingredient to enable novel architecture designs based on PIM paradigm, HBM-/HMC-inspired technologies, etc. However, photonics is intrinsically an end-to-end communication, different from the consolidated store-and-forward approach of the electronic NoC era, because it is inconvenient/impossible to perform computations (e.g., routing) in the optical domain. Therefore, specific, novel design solutions will need to be investigated for blending in the intrinsic opportunities of the technology [373].

From a slightly different perspective on photonics, there are attempts for computing directly in the optical domain [374]. For instance, the approach maps to light parameters (e.g., polarization, intensity, color, etc.) the state variables of data parallel elements needing, particularly but not necessarily, linear operations to be performed. Computation happens through light filtering and the results are then converted into the electronic domain. Huge efficiency and latency benefits have been reported for computations that can fit this scheme, which are within very strategic domains like convolution, etc. For these reasons, accelerators based on photonics could well be envisioned in the future and have the potential of positively addressing the memory wall and computational density problems for some classes of computations.

5.2. Reconfigurability beyond FPGAs

In the last decades, FPGAs have established themselves as the main reconfigurable device. They are characterized by up to millions of logic blocks [14], memory cells, and specialized blocks; all interconnected by a statically reconfigurable interconnect. They offer a so-called *fine-grain*, even *bit-level* [214,375], reconfigurability that make them suitable for general-purpose computing. Although this high flexibility is the main reason for FPGAs’ success over inflexible ASICs, it comes at a cost: 60% of FPGAs’ area and power are spent in the programmable interconnect, and the long combinational paths limit their operating frequency [250]. Moreover, bit-level flexibility is often not necessary, as relying on functional units that implement logical and arithmetic operations according to one of the existing technical standards (e.g., IEEE 754 [376]) is usually *good enough* for the need of an application. Taking this into account, researchers have studied alternative ways to leverage reconfigurability that are not affected by the aforementioned limitations.

5.2.1. State-of-the-art solution: CGRAs

One example is the CGRA design (e.g., HReA [213], HRL [214], NDA [241], NP-CGRA [246], Plasticine [250], PX-CGRA [257], REMUS [267], Softbrain [10], X-CGRA [287]), which embraces a *lighter* form of reconfigurability, so-called *coarse-grain*, and was first proposed in the 1990s [18]. Same as PIM, CGRAs gained attention only recently, because of the slowdown in general-purpose processors' improvement that pushes the need of exploring novel architectural solutions. They are an example of spatial architecture, designed as arrays of reconfigurable PEs, equipped with word-level functional units and usually multi-banked SRAM memory. PEs are connected through a reconfigurable interconnect less complex than that of FPGAs. With respect to these, they are less flexible (word-level vs bit-level) in both computation capabilities and interconnect, but this gives performance and efficiency gains, alongside an additional property that acts as a fundamental enabler for many use cases: the more limited spectrum of reconfigurability allows ns-µs reconfiguration time [18]. Unlike FPGAs, that have a reconfiguration time in the order of ms-s, CGRAs can be effectively used for *temporal computations*. Different configurations can alternate rapidly, leading to an execution flow that closely resembles that achieved by instruction-driven programmable architectures. As a counterpoint, the reduced flexibility means that each CGRA is only suitable in its domain, unlike both general-purpose processors and FPGAs. Taking everything into account, we think that FPGAs will keep their position as the preferred solution to implement prototypes, but the interesting characteristics of CGRAs can lead to an expansion in the market of these architectures.

At the moment, the various proposals coming from Academia discussed in this survey show that CGRAs are an active research area. However, they are almost non-existent in the Industry, although some applications exist, as briefly listed in [18]. The demonstrated value in terms of performance and efficiency will ultimately lead them to find their way into Industry, but there are still some open research questions and challenges that need to be addressed before this can happen. In [18], Liu et al. identify in programmability, productivity, and adaptability the main areas where CGRAs fall short. They present four challenges that CGRA designers need to address and give their take on possible research directions: define a productive programming model for CGRAs that is able to produce efficient code with minimal manual intervention; improve the support for speculative parallelism, that is an important source for performance; improve CGRA virtualization by taking inspiration from techniques that proved successful for FPGAs; improve the efficiency of the memory system by investigating vectorized/streaming memory accesses, dynamic customization of the access pattern, and PIM techniques. The first challenge, in particular, is the most important to see a wide adoption of CGRAs in the market, as argued also in [377]: "the challenge with CGRAs is the development of a tool flow to schedule word-level coarse-grained computations into a mesh of PEs with a computational efficiency and functional density that surpasses FPGAs. Such tools should provide a way for the user to program the machine efficiently using popular high level programming languages like C/C++ ". In Section 5.6 we argue that this is the most important aspect for accelerator adoption in general.

5.2.2. State-of-the-art solution: Limited reconfigurability

Apart from investing in CGRAs, another possible solution is to exploit reconfigurability by limiting it to a single aspect, like routing-only, as in FPSA [192], and Micron Automata Processor [231], or computation-only, as in DRISA [186]. Accelerators could take advantage of partial reconfigurability for critical aspects only, so to achieve higher performance and efficiency figures with respect to equivalent full instruction-driven solutions. FPSA [192], Micron Automata Processor [231], and DRISA [186] have no instruction-driven components at all. In these accelerators, reconfigurability is responsible for providing some *degrees of freedom* to their users, but do not innovate consistently the classic approach to reconfigurability: they limit its scope with respect to FPGAs, placing halfway between an ASIC and an FPGA.

5.2.3. Perspective: HRPA

CGRAs have the potential to gain a wide adoption in the market, but there are various challenges connected to their effective exploitation that are yet to be solved. In our opinion, between them, the difficulty of effectively and productively program them is the biggest obstacle to their adoption. However, even if CGRAs as they are will not gain wide adoption, there is one lesson that can be learned from them: *reconfigurable logic* with a *fast reconfiguration time* is a valuable architectural component, as it could be used to implement critical logic efficiently, while not hampering the support for temporal computations.

Fast reconfigurable logic could be employed in a novel architecture that would have various advantages: a Hybrid Reconfigurable/Programmable Accelerator (HRPA). Unlike accelerators like Xilinx Versal ACAP [152] where programmable (Scalar and Intelligent Engines) and reconfigurable (Adaptable Engines) are separated in a tiled architecture, HRPA would take advantage of a tighter integration of the two, e.g., with a spatial array of programmable engines interconnected by a reconfigurable interconnect.

In our vision of such an architecture, every PE would represent a processing node, and as such would be responsible for performing a *task*. Each task could be expressed as a function that processes input data and produces output data. The instruction-driven PEs would allow functions that execute an arbitrary sequence of instructions, conveniently expressed with general-purpose programming languages. Connection between PEs could be expressed with point-to-point abstractions like Golang's channels [378], and be mapped on the reconfigurable interconnect as routing paths. If reconfiguration time is in a much different time scale with respect to the instructions, a productive use of such an accelerator would be difficult to achieve. Conversely, if reconfigurability time is in the ns-µs range, as in CGRAs, such hybrid reconfigurable/programmable architecture could be controlled in a homogeneous way, with a special instruction/command packet destined to set the configuration the same way as other instructions/command packets would control programmable resources. Such an accelerator would be a good mapping for dataflow workloads and graphs with non-trivial processing steps at a node level.

HRPA would present some challenges by itself that are typical of spatial accelerators, like achieving efficient clocking/synchronization strategies or an efficient exploitation of the memory subsystem. However, it would have the potential of being easily adoptable from a programming point of view and have an efficient way to manage data routing between processing components.

5.3. Designing hierarchical accelerators

Beside the trend of integrating more and more relatively simple accelerators on heterogeneous SoCs, a complementary one is emerging. Some accelerators are evolving towards embracing the *acceleration* concept within themselves, thus forming de-facto a *fractal-like* architectural pattern at system-level. In other words, accelerators have taken momentum due to the increasing limitations and technological constraints limiting the performance/energy scaling of general-purpose units. Then, some of these specialized accelerators have started to grow modules within themselves to make some specific sub-tasks more efficient or faster and forming a self-similar "hierarchical" architecture. This trend witnesses that the same limitations occurring in general-purpose units affect also some accelerators with a relatively wide application spectrum, to which they respond embracing deeper internal specialization. Therefore, these accelerators not only target special-purpose tasks offloaded by the cores, but they feature also some specific hardware blocks, even more specialized, that push the boundaries of performance and energy efficiency. The challenges connected to these accelerators concern a fruitful cooperation between these different modules, as this heavily affects performance and efficiency of applications that take advantage of more than one module.

5.3.1. State-of-the-art solution: modern GPUs and FPGAs

Various examples can be found between complex, wide-spectrum accelerators, usually designed as PCI-e cards. For instance, high-end GPUs, which are mainly made up of tens of multi-processors featuring simple, in-order cores, optimized for graphics and data-parallel workloads. NVIDIA introduced with its Volta data-center architecture Tensor Cores [379], specialized in MAC operations that characterize common ML workloads like CNNs. With Turing [135], they introduced Ray Tracing Cores, which are used to calculate reflections, refractions, shadows, and indirect lighting in real-time. AMD took a similar path with RDNA2 architecture [78], with the introduction of Ray Accelerators, which are dedicated cores specialized in calculations of ray intersections. In both cases, Tensor Cores and Ray Tracing Cores/Ray Accelerators perform operations that could be performed also by the standard in-order cores, but are more suitable for them from a performance perspective: the same relationship that exists between accelerators and general-purpose processors.

FPGAs are undergoing a similar process. Xilinx introduced them in 1984 with few programmable LUTs and switches [380], and evolved in size and complexity with a variety of logic cells, LUTs, flip-flops, DSP slices, etc [148]. The real *hierarchical* step has been done recently, when Xilinx introduced ACAPs [152]. They are conceived as an accelerator with three main modules interconnected by a NoC: Scalar Engines, which are general-purpose ARM processors, Adaptable Engines, that are made up of programmable logic and memory cells (similar to classic FPGAs), and Intelligent Engines, that are VLIW PEs specialized in ML and DSP operations.

5.3.2. Perspective: architectural and programming integration

We expect this trend to be more and more widespread in the design of future accelerators and thus we expect this pattern to be an active research and development direction. In particular, the programming layers and the related ecosystem should address this hierarchical nature by investing in a seamless interaction between on-accelerator modules. For those cases that would normally require different programming approaches, the particular characteristics of these should be blended to achieve more natural programming models that stress the interoperability, even across different domains. Currently, there is not much attention to this issue: for instance, CUDA Tensor Cores do not interoperate with CUDA cores and should be treated with special API functions. From an architecture point of view, the design of resources shared between different modules (e.g., on-board memory resources), the relative placement of modules on the accelerator to promote interaction or isolation, and even the wiring and connections (or lack of) between different modules are all aspects that need accurate planning, so to optimize the accelerator layout, performance, and efficiency, taking into account mono-module and multi-module workloads. In general, both programming model and architecture should be designed with a *systemic* approach that promotes cooperation between modules, so users can take advantage of the peculiarities of each in complex workloads naturally, without artificially splitting their programs.

5.4. Unified virtual memory space

In the most common memory model adopted in heterogeneous systems, host processor and accelerator memory spaces are separated. The host processor manipulates virtual memory pointers and transparently leverages cache coherence protocols that move the burden of data consistency to hardware and O.S. level. Accelerators may manipulate virtual memory or even physical memory, but usually have their own memory space that interoperates with host memory through explicit, DMA-mediated copies. This poses a heavy burden on the programmer: they must address this distinction by managing *couples* of pointers, one for each memory space, copying data back and forth between the two explicitly. Apart from programmability, performance may be impacted, as programmers could fail in achieving an efficient overlapping between data movement and computation phases, introducing unnecessary serializations.

In order to solve this criticality, many researchers and organizations like the Heterogeneous System Architecture Foundation (HSA) [381] are patronizing the support for a Unified Virtual Memory (UVM) space between host processor and accelerators. This would allow application-level pointers to be passed seamlessly between host-code and accelerated-code, without the hassle of explicitly managing replication and consistency.

UVM can be achieved whether the accelerator shares a level of the memory hierarchy with the host (e.g., integrated accelerators) or not (e.g., PCI-e cards). In both cases, data must be kept coherent so that modifications performed on the CPU are visible on the accelerator, and vice versa. In the latter case, explicit copies between two different physical memory spaces are still needed, but can be demanded by the host-accelerator interfacing logic. There are already interconnect-level protocols that implement it (e.g., CCIX [46], CXL [382]), or accelerator-specific high-level APIs that hide copies under-the-hood (e.g., CUDA [65] and OpenCL [57,58,309] Unified Memory). In both cases, the coherence mechanism must involve all the levels of the memory hierarchy: a datum written on the accelerator could be cached at all levels of the CPU cache hierarchy, and a subsequent read by the CPU of its copy in the L1 should read the same value written on the accelerator. The opposite is true, as the CPU could modify a local datum and the updated value should be also read on the accelerator — which could have a complex memory hierarchy by itself. Data coherence between CPU and accelerator, due to the many memory levels involved and the variety of possible situations, is a demanding challenge, and it must be addressed to achieve a Unified Virtual Memory space.

5.4.1. State-of-the-art solution: Virtual memory support

In order to have a UVM space shared between CPU and accelerator, both of them need to support virtual memory. While it is a very common requirement for modern CPUs, the situation is not the same accelerator-side. The main cost is the hardware support of virtual-to-physical address translation [383,384], and thus specific techniques try to minimize this aspect.

For instance, in [384] (2015), Vogel et al. concentrate their attention on many-core accelerators in heterogeneous embedded SoCs (even if their approach can be extended to any accelerator with direct access to the main memory), and propose a lightweight virtual memory support for them. Starting from the consideration that a proper implementation of IOMMU in that case could be unaffordable for area and energy consumption constraints, they propose a Remapping Address Block (RAB) as a software-managed replacement for the IOMMU. It is a simplified software-only approach that uses a kernel-level driver module and a user-space runtime to remap virtual addresses to physical addresses. The miss penalty is mitigated by prefetch, and prior knowledge of the memory access pattern is used to initialize the relevant data structure. Despite its simplicity, the technique delivers a 2× speedup with respect to classic copy-based solutions [384].

Also Hao et al. target heterogeneous SoCs [383] (2017). They propose an efficient address translation support for accelerators that share the physical memory with the CPU. They observe that accelerators that support virtual memory do it by the means of IOMMUs with support for I/O Translation Lookaside Buffers (IOTLBs) and some logic to walk the page table. In these accelerators, the latency involved is very long and the performance reaches only 12.3% of the ideal address translation [383]. Their proposal is composed of three elements:

- A small (16–32 entries), private TLB to save accesses to the IOMMU;
- A level 2 TLB shared between accelerators to filter translation requests on common pages;
- An interface to offload TLB misses to the host MMU.

This scheme achieves 93.6% of the performance of an ideal address translation, but requires important additions to existing accelerators and SoC architecture.

In general, using TLBs in accelerators poses challenges, because it requires either (a) processor-specific page-walking strategies to be implemented accelerator-side, which is very complex and not portable; or (b) relying on the CPU translation only in case of accelerator TLB misses, which is faster on average but still requires complexity on the accelerator; or (c) have all translations done in the CPU, which reduces accelerator's complexity at the expense of performance penalties. It can be expected that all the intermediate shades between macro-approaches from (a) to (c) will be adopted by different accelerators. For instance, considering the typical workloads, intense usage patterns, probably finer-grain, between accelerator and unified memory could push towards the (a) extreme, while more infrequent and bulky transfers are more likely to promote the implementation of solutions close to (c), as the high translation overhead would be amortized.

A different approach can be found in [385] (2018). Haria et al. propose a scheme to keep virtual memory as-is but to eliminate the need for address translation in the majority of cases — in their words, to "devirtualize virtual memory". Their approach is organized in the following steps:

- Memory is allocated such that physical memory and virtual memory are almost always identical (identity mapping);
- Memory protection is enforced by checking application permissions for each access.

Their De-Virtualized Memory (DVM) reduces the translation overhead to less than 2%. It also gives advantages in programmability, power/performance, flexibility, and safety [385]. It must be noted that it requires small modifications to the O.S. to support identity mapping, and imposes further constraints by construction — like eager paging, that is more expensive in terms of required disk space.

5.4.2. State-of-the-art solution: Cache coherence techniques

Many works analyze the implication of cache coherent accelerators. For instance, in [303], Giri et al. discuss the SoC case, which is usually characterized by the sharing of physical memory between CPU and on-chip accelerators. In particular, they study the impact of coherence on performance in the case of accelerators designed in isolation and then integrated into the SoC. They identify in the literature three cache-coherence models:

1. non-coherent;
2. coherent with the last-level-cache;
3. fully-coherent.

Non-coherent accelerators access directly the DRAM through DMA bypassing the cache hierarchy. In order to get coherent data, the processor caches must be flushed upon accelerator access. The support for DMA bursts is necessary in this case to be efficient. LLC-coherent accelerators use the DMA to access the LLC. In this case, the requested data region must be flushed only from the private caches of the processor. This model is efficient only if LLC has a high hit rate. Fully-coherent accelerators are included with the processor's caches in a classic coherence protocol, such as MESI or MOESI. In this case, no flushing is necessary. They also identify a fourth model, which is implemented by ARM's AXI Coherence Extensions (ACE). It maintains uncached accelerators coherent: any shared access to memory can "snoop" into the caches [386].

In order to opt for one policy or another, a variety of factors must be taken into account. First, the fact that cacheability and coherence requirements may be dependent on the application needs even within the same accelerator. Considering GPUs, for instance, GPGPU's buffers and control structures may need to be coherent because they are read/written by GPU and CPU very often. Conversely, tasks like display/render need no coherence and benefit from cache&flush approaches. Because of this, different allocation primitives (e.g., CUDA [65]) or, equivalently, allocation primitives with memory flag parameters (e.g., Vulkan [63], OpenCL [57,58,309]) may be exposed to the programmer, so they can control the intended outcome. The different policies are implemented by runtime and drivers, with several implications on performance [387]. We can expect this trend to be mimicked also in other accelerator domains, as it is flexible enough to support various use-cases, requires no modifications at the hardware level, but has the drawback of increasing the programmer's responsibilities. Simultaneously, we are already witnessing proposals based on automatic and semi-automatic mechanisms that keep track or even infer the intended memory usage to adopt *selective caching* policies [388].

Second, the potential impact that the presence of a cache coherent accelerator may have on the system. Taking again GPUs into consideration, they tend to consume data at a high bandwidth. Adding naïvely a GPU to a MESI/MOESI protocol as if it were another core may seriously hamper CPU's functioning, as the GPU would fill caches with its data and wipe out whatever was there before, like cores' and other accelerators' data. This is a well-known research problem going back to the first attempt to put cores and a GPU on the same processor die: already in 2011, integrated GPUs in AMD Fusion APUs distinguished between uncacheable and cacheable memories [389], giving to programmers the opportunity to recur to uncacheable memory whenever possible. However, to properly handle those cases when caching is applied and coherence is involved, various solutions have been proposed. Some examples are cache partitioning [390–392], cache locking [392], and other modifications to the cache functioning [393]. Also in this case, we can expect these techniques to be adopted for other high-bandwidth accelerators that may suffer from the same foundational issue. We expect to be one of the top research and development priorities in the evolution of accelerated architectures.

5.4.3. Perspective: Different solutions for different use cases

As we observed, virtual-to-physical translation is the most critical design choice to implement virtual memory support on accelerators. The various possibilities (accelerator-side translation, accelerator-side translation with CPU intervention on TLB miss, CPU-side translation) have pros and cons that make the best choice dependent not only on the accelerator's characteristics, but also on the application. In the case of simple, fixed-function accelerators, the best choice can be uniquely determined, but non-trivial accelerators exposing programmable or configurable operations do not permit this. For instance, the need of efficient management of scalar and small-sized structures for configuring, controlling the accelerator activities while in operation, monitoring, and signaling could require efficient low-latency autonomous translation – e.g., in contexts where an accelerator is part of a pipeline of streamed data elaboration between itself and one/multiple cores. In such applications, also big data buffers need to be moved in and out of the accelerator for processing, so we can imagine that smart DMA engines will be refined to manage efficiently both situations: buffer copying, along with the finer-grain and fast transfer support. Therefore, the best design choice would be to investigate the coexistence of various strategies, and also automatic techniques to determine which of them would be the most convenient for a given translation task.

In general, whenever TLBs are replicated, coherence needs for their content emerge and pose related challenges. Either accelerators' TLBs expose a HW/SW interface similar to that of general-purpose cores and are *visible* to the O.S. as *nodes* of the virtual memory management, or coherence maintenance will require ad-hoc solutions at a certain level in the drivers and/or runtime. This situation pushes from one side towards a standardization of the Virtual Memory interface of accelerators and raises research opportunities on novel OS-cores-accelerator interaction patterns.

From a different angle, we can expect that DVM-inspired approaches could be blended with TLB-based ones through hybrid solutions trying

to limit each other's weaker facets, maybe also assisted by ad-hoc functionalities implemented in the cache hierarchy.

Our analysis highlights the necessity of supporting various scenarios for coherence strategies as well. Simple accelerators with straightforward needs could allow designers to opt for a neat strategy (e.g., non-coherent accelerator), but complex ones with various use cases would suggest that different strategies should be allowed to accommodate different application needs. The availability of cacheable and uncacheable buffers, which is common in GPU-oriented APIs (both graphics and data-parallel), could extend to other accelerator APIs and even in high-level programming languages to ease interoperability with accelerator-oriented libraries. So, for this aspect, in particular, we think that a general solution is difficult to achieve, and giving to programmers the control could be a viable choice to achieve high performance or efficiency in various scenarios, avoiding unnecessary cache traffic when possible, and opting for coherent buffers when necessary. In this regard, a consolidated standardized interface for managing these aspects is not available yet, but the increasing needs are pushing significantly research and development advances in this strategic direction.

5.5. Virtualization

In data-centers, different users' run their applications in virtual environments that share the same physical resources and provide encapsulation, protection, and security. According to the current trend, both the number of users and the number of applications will likely increase in the next years, as we are already witnessing the migration of many services into the cloud. Accelerators are already being deployed in data-centers [394,395] and will likely play a prominent role in the near future, as their interesting characteristics in terms of promised throughput and efficiency are particularly amenable in the energy-constrained context of data-centers. They can be made available to users by ad-hoc services that manage request queuing and scheduling, but the most straightforward way is to grant access from virtual environments, giving to users the sensation of exclusive ownership that they would have on their personal workstation. To achieve this, accelerators need to be *virtualized*.

Virtualization is important also to support remote accelerators: a *virtualized remote accelerator* would be seamlessly used in a user application as if it were physically present on the current machine, while the backend would take care of forwarding commands to the nodes that physically host the accelerator. This would allow a more natural way to share limited resources and promote interesting computing scenarios – e.g., fog computing [396].

5.5.1. State-of-the-art solution: Per-accelerator approach

Some accelerators already support virtualization and are successfully employed in data-centers. Few examples are AMD's GPUs, that mount a hypervisor agent [74], NVIDIA GPUs, with their NVIDIA vGPU technology [138], and Google TPUs, that recently announced Cloud TPU VMs [397] to control TPUs from virtual machines. Other examples include FPGAs [398–400]; Xeon Phi [401]; GPUs in automotive environments [402]; and GPUs on servers with API remoting [403,404], hardware-assisted [405,406], and all-software [407] techniques. These solutions are very specific to selected scenarios, and cannot adapt to the general case. Although individual accelerators present individual challenges related to their architecture, whether they are reconfigurable or not, and their programming environment, a general solution should be investigated. It would benefit accelerator designers and users, freeing the former from designing ad-hoc solutions and giving to the latter the ability of using any accelerator in virtual environments. The first question to answer in search of a general solution is what level of the hardware–software stack it should target.

5.5.2. State-of-the-art solution: API remoting

Yu et al. well summarize the difficulty of finding a general solution from a full-software perspective [408]. They identify in the opaque proprietary programming layers (user-mode API, user-mode drivers, kernel-mode drivers) the *impossibility* to virtualize accelerators by interposing virtualization code in the middle of the stack. Their approach is to rely on API remoting: maintaining an API server on the host or in a dedicated VM that intercepts user-level API calls. They propose Automatic Virtualization of Accelerators (AvA), which provides a declarative API specification language. Starting with an API specification in the form of an annotated header, AvA virtualizes it and generates a paravirtual[3] communication infrastructure [408].

AvA targets the highest level of the software stack, leaving the lower layers unaltered. The major criticism that can be moved to this approach, which is common to those solutions based on API remoting, is that it has a very difficult goal to achieve: a one-to-one correspondence between original and remoted API including syntax, semantics, and side effects. This requires a deep understanding of the user API and the ability to interpret it correctly, which is severely impacted by the possible presence of undocumented constraints, the size of the API, and its stability over time.

5.5.3. State-of-the-art solution: SoC hardware module

As an example of an opposite approach, Govindarajan et al. propose the use of a dedicated, programmable microcontroller on SoCs to proxy requests from multiple sources and schedule them on the corresponding devices [409]. A software entity (driver or runtime) would provide additional logic to identify the source of requests, generate the transaction credentials, and orchestrate scheduling on the accelerator. The credentials are used to tag transactions with client information, later used to perform translations in system-level IOMMU on a per-client basis.

5.5.4. Perspective: A truly general solution

We acknowledge the difficulties analyzed by Yu et al. [408] in intervening in the lower programming layers, as these are inter-dependent and opaque, and we think that a more complete solution should target the hypervisor or even the underlying hardware level. As we highlighted in Table 8, the set of user-space APIs and frameworks is already big and even fast-changing. Unless a proposed solution can address all of them transparently, approaches that require a translation, albeit semi-automatic, of each API are not feasible at the current state of things. The risk is to translate a subset of APIs, allow users to run a subset of applications in the virtual environment and force them to port the remaining ones. As users are steadily migrating to the cloud an increasing number of services, an ideal solution would allow them to seamlessly port their applications and run them in the cloud.

In our opinion, the solution proposed by Govindarajan et al. in [409] goes in a promising direction. Its support of software-driven scheduling is fundamental to foster various scenarios. The simplest example is an automatic serialization of concurrent accesses to the accelerator, so to grant exclusive access to one client at a time. It is the most appropriate technique for *simple* accelerators that do not have instruments to manage various clients executing together, and also the most effective to achieve isolation. Conversely, more sophisticated accelerators could take advantage of *on-board* scheduling or partitioning techniques that would allow multiple concurrent applications to execute even in a sand-boxed, secure way. In the first case, the software-driven scheduling would not be necessary, and could even act as a pass-through and rely on on-board scheduling techniques. In the second case, the interaction would prove more complex, as the software entity should be aware of the resources needed by the

[3] Intended as the adaptation of a software interface so it can be used in a virtual environment.

transaction and the current resource availability on the accelerator. A tight integration with proprietary virtualization techniques would probably be necessary.

In order to be truly general, the solution proposed by Govindarajan et al. in [409] needs to be extended to target not only SoCs, but also PCI-e based accelerators. This can be achieved by moving part of the logic into the PCI-e controller or by expanding the PCI-e architecture à la CCIX [47]. As virtualization needs increase over time, we may expect in the future a convergence on a standard, vendor-independent solution that accelerator designers could adopt as a common interface to provide virtualization capabilities.

5.6. Programming accelerators

Today's world is heavily shaped by information and the many ways it can be manipulated and used. Computers, in their diverse incarnations ranging from wearable devices to data-centers, are the main instrument for this work. Governments, companies, universities, research centers, and even private citizens maintain billions of lines of computer code[4] that shape every aspect of modern life; they are the backbone of uncountable technologies and services we use every day, and have a primary function in driving scientific progress as a whole. These lines of code would be orders of magnitude more and would be unmanageable without the advancements of the past decades in high-level programming languages and compiler technology.

High-level languages allow programmers to formulate elaborate operations and abstract concepts with few expressive lines of code. Not only they offer a degree of abstraction and compactness that limits the program size, they are also closer to natural language than assembly or machine code, and allow expressing the entities representing both the *problem-space* and the *solution-space* more directly. As a consequence, writing, reading, maintaining, and debugging code is easier — in other words, *programmer's productivity*, programs correctness and reliability, as well as systems safety are considerably higher. This is due also to the concomitant advancements in compiler technology. Modern compilers include tremendous optimization capabilities that can target performance, power consumption, or size of the produced code even starting from very high-level computations and data structures. Thanks to them, most of the time today's programmers can concentrate on high-level coding only, relying almost completely on their compiler and avoiding tedious and error-prone hand-tuning of the compiled code altogether.

Fifty years of high-level languages and compilers advancements are supported by a flourishing of an ecosystem that includes libraries, frameworks, debuggers, IDEs, deployment, and monitoring tools. However, this rich ecosystem is mostly about computing on general-purpose processors, and its evolution led to the point that the underlying architecture has been progressively abstracted away. General-purpose code is mostly *portable*, at least for what concerns the target architecture that usually comes into play only at compilation time.

The emergence of multi-core processors suddenly broke the abstraction layers, putting architecture back in the foreground, even in high-level languages that strove to keep it hidden from the programmer. Parallel programmers need to think about their programs as aggregates of separable threads of execution, to be kept as independent as possible, minimizing synchronization points, and avoiding undesirable outcomes such as data races, deadlocks, and load unbalancing. Some libraries and frameworks help them to achieve this and take care of some complexity (e.g., OpenMP [60], C++17 Parallel STL [411], etc.), but they are mostly data-parallel frameworks. Abstracting away multi-threaded and multi-core parallel programming and delegating these activities to a parallelizing compiler is still an open research area [412–415], which is expected to receive great effort from both research and companies in the near future.

5.6.1. State-of-the-art solution: GPUs and the data-parallel ecosystem

The diffusion of GPUs complicated the picture even more. The first examples of GPGPU were an attempt of adapting GPUs' highly parallel architectures to perform workloads not related to graphics.[5] At first, they were coded by the means of *shaders*, but the appearance of NVIDIA CUDA programming language in 2006 [65] allowed researchers and practitioners to switch to a more natural tool. CUDA is an extension of the popular C++ programming language. It features a special syntax to launch kernel functions on the device and a few keywords to tag kernel functions, device functions, and data to be stored in special memory resources. It has its compiler (nvcc), runtime, and a rich ecosystem that includes debugger, profiler, and libraries (linear algebra, NN, RNG, etc.). Although it succeeds in abstracting various details of the underlying architecture away, it still requires programmers to write complex code structured as a 1D/2D/3D grid of 1D/2D/3D blocks of indexed threads that execute in groups of 32 at once (*warps*), manipulate data in the device memory with the help of special low-latency memories (shared, constant, texture) optimized for different access patterns (coalesced access, broadcast, spatially local), and can synchronize both at block-level or at warp-level. Despite its complexity, CUDA contributed fundamentally to the diffusion of GPUs, arguably the most widespread accelerators today.

In 2009, OpenCL [57,58,309] was proposed as the first and still most popular attempt at achieving code portability across different platforms. It is a vendor-independent data-parallel framework that targets multi-core CPUs, GPUs, and even FPGAs — so, it is not even limited to data-parallel accelerators, albeit its programming model is purely data-parallel. It is deeply inspired by CUDA, but adds the possibility of writing truly heterogeneous code, as different platforms can be targeted in the same executable. However, it has a lower-level interface compared to CUDA, and despite its higher portability, OpenCL's adoption is still lower than CUDA's, also thanks to the rich ecosystem of libraries and tools of the latter. Currently, other solutions like Sycl [59], Kokkos [355], SkelCL [356], and PHAST [360] try to conciliate performance, portability, and productivity, often relying on lower-level approaches like CUDA and OpenCL as backends hidden to the programmer. However, despite their higher level coding facilities, these solutions are not replacing existing lower-level ones, yet. The momentum of industry and big project codebases promote the consolidation of existing approaches, as usual, while new and smaller projects should embrace the vanguard of new approaches.

5.6.2. State-of-the-art solution: The machine learning ecosystem

While Data-parallel is a wide domain that encompasses a great variety of applications, ML is not as wide, and the majority of ML applications involve either running or training a model, usually a NN. A successful framework in the Data-parallel domain needs to provide enough abstraction to achieve productivity, while still guaranteeing enough freedom and fine-grain control to express an uncountable variety of applications with few in common (e.g., embarrassingly parallel, mono-/multi-dimensional stencil or scan, histogram, etc.). A successful ML framework needs to help programmers to automatize the expression of the *structure* of the problem, so they can work on its tuning. For this reason, the most popular frameworks and libraries in the ML domain have generally a higher level of abstraction than their data-parallel equivalents (e.g., managing NN layers and relationships between them).

Most popular ML-oriented libraries and frameworks commonly used today allow programmers to code applications that present several opportunities for data-parallelism. GPUs were already widespread and were the most common data-parallel platforms, together with multi-core CPUs, when these tools first hit the market: Caffe [64] was

[4] In 2019, the GHTorrent dataset contained 125 million projects from GitHub only, which is the most popular code repository website [410].

[5] These workloads are best described as *data-parallel*, albeit the GPGPU acronym refers to them as *General Purpose*.

launched in 2013, TensorFlow [56] and Keras [359] in 2015, PaddlePaddle [307] in 2016, PyTorch [305] and ONNX [306] in 2017. Therefore, all of them support at least multi-core CPUs and GPUs as native platforms of execution. Basically, they are conceived as high-level interfaces based on lower-level backends. This layered architecture has the advantage of integrating seamlessly with novel accelerators: it would be sufficient that accelerator manufacturers provide an ad-hoc backend to manage all the accelerator-specific code under the hood. Programmers already proficient with their favorite tool can port their code on a novel accelerator and start developing for it without the need of learning new programming models, in principle. As shown in Table 8, this is the solution adopted in many cases and we expect it to consolidate even more if the contour conditions remain stable. Moreover, this trend has the side effect of helping to consolidate well-established solutions further, pushing towards a standardization of the ecosystem.

5.6.3. Perspective: The future of accelerator programming

In both data-parallel and ML domains, we are witnessing a push towards a standardization of ecosystems, with few proposals dominating the landscape. Novel accelerators usually provide backends/compilers to these existing solutions, easing accelerator adoption and taking advantage of a consistent body of applications immediately on launch. We are also witnessing proposals that try to elevate the abstraction level or increase portability, but these are failing in replacing well-established solutions. Overall, the strategy of supporting what is already widespread seems to be the most effective.

The situation is not as neat for other domains where a widespread solution does not exist yet, like Dataflow or Graph processing. In these cases, accelerator manufacturers can try to adapt a solution commonly used in other domains for their needs and take advantage of the popularity of the solution to attract developers, as done by the authors of IMP [226] and Heterogeneous-PIM [212] with TensorFlow and OpenCL, respectively, but also by NVIDIA with CUDA, that expanded a very popular general-purpose language (C++) with data-parallel capabilities. This solution could not be available for accelerators that operate in particular and/or novel domains. An *original* solution may be mandatory. In this case, given the trends in other domains, we argue that the only fundamental requirements are the ability of reaching high-performance/efficiency (i.e., effectively exploiting the underlying hardware) and a high level of abstraction. A novel accelerator could potentially be a fundamental enabler for a number of applications and services, but after decades of high-level programming languages and frameworks, developers are unlikely to want to go back to much lower-level techniques. This could delay the adoption of new approaches up to the moment when other constraints, e.g., emerging from the inefficiency of high-level solutions, will appear clear. From a slightly different perspective, the performance and/or efficiency gains provided by a new accelerator with respect to more traditional solutions may be counterbalanced by the productivity loss of the programmers, and thus prove not convenient overall. Conversely, a productive programming paradigm, even if completely different from well-established approaches, could be able to gain consensus in a relatively short time.

All the analysis done so far has a mono-domain perspective, since the majority of applications we witness today have this characteristic. However, with the ongoing trend of integrating on SoCs various accelerators specialized for different tasks, the opportunity of leveraging a tighter cooperation between them should be extensively studied from a programming perspective too. Multi-domain frameworks or at least interoperability between different ones should be investigated, because it could stimulate novel, complex applications that we cannot easily envision otherwise. Some examples of the possibilities of this *contamination* can be taken from the recent achievements of ML techniques applied to other fields: resolution upscaling, autonomous driving, cancer detection, gaming [21]. This cooperation between accelerators needs to be investigated from a perspective that tends to unite them, rather than separate them through vertical approaches. The offload model needs to evolve significantly with the help of powerful hardware–software abstractions (e.g., Unified Virtual Memory, data coherence) and runtimes, to arrive at seeing the whole system as a single development platform with different processing components that can be used transparently, the same way as we see our CPU as a single processing element and ignore its nature as an aggregate of functional units specialized for different operations. Given the trends in various domains, in the future this might be achieved with high-level frameworks backed by integrated compilers and runtime that are able to automatically identify, at various granularities, the high-level operations patterns and calculations that can be demanded to individual accelerators. In our view, the research in the accelerator programming field should point towards this goal, in order to give to programmers of heterogeneous systems the productivity level that they are experiencing in homogeneous contexts.

6. Related work

Between the many previous works that discuss accelerators, a paper by Caşcaval et al. [304] stands out for the similarity of intentions with ours. In that paper, the authors present and discuss a taxonomy of accelerator architectures. We got inspiration from it, but we present a richer taxonomy that includes more aspects and discuss the high variety of accelerators we witness nowadays. In our study, hardware and software aspects are presented in more details, so we prefer a classification of families (e.g., NVIDIA GeForce RTX 30xx [138] or Intel Graphics Technology (Gen11) [105]), rather than broad categories (e.g., GPUs). Moreover, we do a performance comparison of many accelerator models pertaining to two well-represented domains (Data-parallel and ML), and discuss some trends. Finally, we analyze open challenges that involve accelerators horizontally, and enrich the discussion with state-of-the-art solutions and prospective research directions.

To the best of our knowledge, the work by Caşcaval et al. is the only one that tackles accelerators *in general* and describes them from a holistic perspective. Other works keep part of this generality to some extent, but restrict their scope to one or more aspects (e.g., [18,21–23,292,294,297–299,394,416–441]). We discuss these works in the following.

6.1. Accelerators in particular domains

The most common approach when discussing accelerators is to restrict their scope to a single *domain*, i.e., the class of applications that can be executed on the accelerator. Between them, the most represented domain is by far AI/ML, due to the growing interest in this area and the consequent explosion of related accelerator proposals, as we show in this survey. For instance, in [416], Du et al. compare two classes of AI-oriented accelerators: those inspired to ML and those inspired to neuroscience (called *neuromorphic*), from performance, accuracy, and cost standpoints. In [21], Sze et al. discuss DNNs in-depth in a comprehensive tutorial that covers many aspects: history, applications, popular models, hardware platforms, and so on. In particular, in Sections V and VI they discuss DNN processing on accelerators. In [417], Li et al. discuss various neural-network accelerators and acceleration techniques, analyzing in-depth the DianNao family [181]. In [418], Reuther et al. discuss many accelerators and processors for machine learning, focusing on performance and efficiency figures. In [419], Mittal and Umesh survey works on spintronic accelerators for NNs and PIM. The same authors study RNN acceleration on GPUs, FPGAs, and ASICs [420]. They classify accelerators with respect to their core-computation engine (matrix–matrix multiplication, matrix–vector multiplication, etc.) and discuss optimization and simplification techniques. In [421], Deng et al. discuss in-depth compression in neural networks, and dedicate Section 4 to hardware accelerators. In [422],

Chen et al. analyze accelerators for DNNs. They identify four broad categories: on-chip accelerators, stand-alone accelerators, accelerators with emerging memories, and accelerators for emerging applications. For each category, they discuss some sample accelerators from both industry and academia. In [423], Moolchandani et al. classify and analyze works proposing acceleration of the CNN inference phase on ASICs. They group prior works in four groups, with respect to the metric they aim to reduce: computation time, number of computations, memory access time, and size of memory footprint. In [424], Mittal and Vibhu present an overview of accelerator architectures for 3D CNNs. They present prior works on the topic and discuss algorithmic optimizations.

Some works restrict their scope further, considering only FPGA-based AI accelerators. In [442], Shawahna et al. analyze the then current status of CNN acceleration, various optimization strategies adopted and classify the existing proposals. In [425], Shen et al. present a technique to design FPGA-based accelerators for CNNs, that they call multi-CLP (Convolutional Layer Processor). In [298], the authors present a survey of FPGA-based accelerators for Deep Learning, mainly CNNs. They discuss existing solutions and identify promising development strategies. A similar work is published by Mittal [297], which presents a survey of Convolutional and Binary Neural Network-oriented accelerators on FPGAs.

Data-parallel is another well represented domain in the accelerator world, since it comprehends all the contemporary GPUs and some notable, albeit abandoned, examples like the Intel Xeon Phi family. However, as far as we know, the only work that studies them *in general* is a paper by Lee et al. [426]. Here, the authors study data-parallel accelerators' micro-architectures comparing SIMD, MIMD, and vector technologies. They study the implications of each model to programmability, area, and efficiency, proposing their own vector-thread solution.

Other notable works include a paper by Gui et al. [427] and a book by Kurzak et al. [428]. In the former, the authors present many accelerators for graph processing based on different paradigms, and many aspects associated to graph acceleration like runtime support, scheduling schemes, etc. In the latter, the authors discuss the acceleration of scientific computing workloads, performed mainly on multicore processors, the Cell Broadband Engine [290], and GPUs [428].

6.2. Energy-constrained scenarios

As we said, the main reason of interest in accelerators is their promise of improving non-functional requirements such as performance and efficiency with respect to general purpose solutions. Efficiency, in particular, is of the utmost importance in energy-constrained contexts, such as embedded and data-centers.

In embedded, emerging applications like edge computing and IoT would particularly benefit from tiny accelerators that would perform their tasks for a fraction of the energy needed by general-purpose processors and micro-controllers. In Iniewski's book about embedded systems [434], various chapters are dedicated to hardware acceleration. Chapter 2 is dedicated to hardware for computational biology, Chapter 3 to embedded GPUs, Chapters 5, 6, and 7 to FPGAs, and Chapter 14 to reconfigurable architectures for cryptography. In [435], Moyer and Watanabe discuss accelerators in embedded systems from an architectural perspective. They focus on accelerator characteristics (blocking or not, shared or not, signaling completion or not, etc.) and their role in achieving integration into a larger system. In [436], Cardoso et al. dedicate Chapter 7 to embedded heterogeneous computing platforms. They discuss code retargeting to leverage hardware accelerators (mainly GPUs and FPGAs).

Hardware acceleration is taking momentum in data-centers too. Cloud applications like big-data and AI (in particular, training of massive NNs with millions of parameters) pose challenges that cannot be solved with general-purpose processors under stringent power budget requirements. A notable work that discusses the employment of accelerators in data centers is a book by Kachris et al. [394]. Here, the authors present many related research projects, the integration of programmable resources into the cloud and edge infrastructure, or a study of FPGA deployment as computing nodes, which is also the focus of [294].

6.3. Reconfigurable accelerators

We extensively discussed reconfigurable architectures, that currently include FPGAs, CGRAs, and accelerators that employ reconfigurability in a minor form (e.g., DRISA [186], FPSA [192], Micron Automata Processor [231]).

In [429], Chattopadhyay surveys reconfigurable architectures. He analyzes their evolution, classifies them based on their structure and their application domain, discusses tools and proposes future challenges. In [430], Tessier et al. discuss FPGAs and CGRAs. They provide a brief history and many relevant aspects such as usage, energy, emulation, languages, tools, runtime reconfiguration, and applications. In [431], DeHon discusses many aspects of reconfigurable architectures, such as energy, data/instruction locality, wiring, and area, with a particular attention to their mathematical modeling. In [432], Wijtvliet et al. propose a definition of CGRA and describe 36 architectures proposed in the previous 25 years (1991–2016). Then, they present a multi-level taxonomy and classify all the discussed examples according to it. Their work is continued and expanded by Liu et al. [18], which present a comprehensive survey on CGRAs. They define CGRAs and propose a taxonomy based on their programming, computation, and execution models. Then, they analyze related challenges and discuss CGRAs' architecture and application development in-depth. Skliarova and Sklyarov present in their book various topics related to FPGA-based accelerators [292]. In the first two chapters, they introduce reconfigurable devices and their tools, and the architectures of FPGA-based accelerators. Chapters 3 to 5 focus on the employment of accelerators in selected domains, like data search, data sort, etc. In Chapter 6, they explore some hardware–software co-design techniques. We already mentioned [297,298,425] that discuss AI accelerators implemented on FPGAs.

6.4. PIM accelerators

The recent diffusion of in-memory accelerators inspired some surveys on the matter. Fujiki et al. published a book about in- and near-memory computing [23]. Chapter 6 focuses on accelerators that leverage this design principle. They focus on ML, automata, graph, database, and genomics accelerators, describing in-depth ISAAC [229] and Neural Cache [443] for ML workloads and Micron Automata Processor [231–235] for automata acceleration. Chapter 7 focuses on programming models for this class of accelerators. In [433], Mittal et al. survey SRAM-based in-memory-computing techniques, focusing on ML and Automata accelerators.

6.5. Accelerators' specific facets

At this point, the readers should be familiar with the vastness of the accelerator topic. Many facets are involved, and there are works that focus on one aspect in particular. To name a few, in [22], Buchty et al. present a survey of heterogeneous programming techniques on multicores and accelerators. In [437], Hawick and Playne study the software ecosystem of hardware accelerators and the limited opportunities to achieve interoperability between accelerators. They discuss developmental directions to improve this situation. In [438], Cota et al. study and evaluate the effects of coupling techniques between integrated accelerators and other on-chip components, such as cores and caches. In [439], Verbanescu and Shen present various techniques

to leverage host and accelerator together in heterogeneous code, as opposed to the *offload model*. In [440], Margerm et al. present TAPAS, an HLS toolchain to generate accelerators from programs with dynamic parallelism, supporting also heterogeneous, nested, and recursive parallelism. In [441], Addazi et al. advocate for data-parallel Action Language for Foundational UML (Alf) to express computation-intensive parts of an application, so to eliminate architecture-specific code and increase safety in accelerated embedded scenarios.

7. Conclusions

In this survey, we discussed accelerators in-depth adopting a holistic perspective. We presented the many aspects that constitute their design space by proposing a taxonomy organized in fourteen categories, grouped in four macro-categories: general aspects, host coupling, architecture, and software aspects. We categorized about 100 accelerators from the last decade according to it, taking into consideration both commercial products and academic proposals. Some general considerations emerged from the overall categorization:

- Academia is investing a lot of effort in PIM accelerator research, mostly used for ML, Graph processing, and Genome sequence alignment tasks. Memristors are the preferred in-memory approach, while those based on 3D-stacked technologies are common near-memory solutions. In the latter case, general-purpose processors, simple logic, or even complex structures like arrays of PEs and tensor cores have been successfully integrated in the logic die.
- Industry adopts a less experimental strategy, preferring more traditional options like Manycores and GPUs, but Spatial and Systolic designs are gaining increasing attention. The former are adapt to classic workloads (e.g., data-parallel), while the latter are fruitfully employed in ML. These have generally rich ecosystems, with GPUs having the richest.
- CGRAs are another frontier of academic research. They are a new class of reconfigurable accelerators suited for Dataflow, but also ML and even Data-parallel tasks.
- ML is by far the most popular domain, especially CNNs/DNNs. Inference is targeted three times more than training, and basically every accelerator type has been employed to accelerate this task.
- Host coupling is usually overlooked by academic works. When this does not happen, PCI-e and on-die integration are the preferred connection strategies, even if the diffusion of PIM solutions is making integration on HMC logic die as common. This determines a prevalence of DRAM as the preferred level of the memory hierarchy that is shared with the CPU, surpassing even LLC.
- Most accelerators are programmable, with the instructions destined, in half of the cases, to special-purpose computing resources, such as spatial arrays and vector processors. Despite this, fixed-function hardware blocks are also used — especially for ML-oriented activation functions. Ad-hoc programming languages are a surprisingly common way to program accelerators, providing at least a function-level granularity that guarantees high flexibility.
- TensorFlow stands out as the most popular framework/library among accelerators, while OpenCL is the most heterogeneous in the sense that it can run on the highest variety of accelerators, including Manycores, GPUs, PIMs, and FPGAs.

Then, we discussed the aggregated throughput and efficiency of various accelerator models in Data-parallel and ML domains, which are those that usually disclose these data. A general trend towards reducing precision emerges from both, due to the drive of ML that does not need high precision in most cases, especially for inference tasks. ML accelerators use as little as 1-bit data types, albeit in very-low power accelerators. For data-parallel accelerators, throughput is firmly in the hands of Manycore and GPU types, while efficiency is contended between Manycores and CGRAs. ML accelerators show the advantage of specialization, as testified by the far higher efficiency that highly tailored architectures achieve with respect to more general ones, not limited to ML applications. This is true even comparing specialized architectures of about a decade ago with the most recent data-parallel GPUs with roughly the same power budget. Also, in the ML domain there is no neat distinction between accelerator types from the perspective of throughput and efficiency capabilities, as any type can prove competitive.

If the first part of the paper analyzed accelerators with a vertical approach, the second part concentrated more on a horizontal approach, with a discussion of open challenges that pertain to virtually every accelerator. For each open challenge, we analyzed proposed state-of-the-art solutions and provided our prospective directions for future research. The major points can be summarized as follows:

- We expect the research to continue improving technological aspects of memory-rich processors and PIM accelerators, which are common solutions to the memory wall problem. We argue that promising programming solutions should address their similarities with distributed systems, and we argue that silicon photonics has the potential to be a key ingredient in their evolution.
- We argue that a Hybrid Reconfigurable/Programmable Accelerator (HRPA) has the potential to be a promising approach to exploit reconfigurability capabilities beyond FPGAs, adopting the advantages of the CGRA design while improving at least its programmability, which is the main obstacle to CGRAs' widespread adoption.
- We argue that the emerging design trend promoting *self-similar accelerator architectures* (i.e., accelerators with specific accelerating modules within themselves) will consolidate further and should take cooperation between different modules as a first-class design criterion, so to allow a seamless integration of different modules in complex applications.
- We expect that the variety of accelerators and applications will not provide a single solution to the shared need of a unified coherent Virtual Memory space. We expect that, in the future, the support for various scenarios will be pursued both in hardware, with different translation strategies, and in software, with a possible standardization of allocation primitives supporting different caching strategies.
- We argue that virtualization, that was historically addressed with accelerator-specific solutions, should be pursued with a more general approach, giving important advantages to both designers and users. We argue that a full-software strategy is not convenient, and solutions based on hardware modules supporting accelerators with different levels of complexity should be investigated instead.
- We observe that both data-parallel and ML programming ecosystems are going towards standardization, with few dominating proposals. We argue that novel accelerators addressing these domains should continue the trend of supporting well-established frameworks, while accelerators in less-standardized domains should pursue efficiency and productivity above all. We argue that the research in this area should concentrate on a comprehensive, multi-domain framework to gain the ability to target multi-accelerator systems, which may be common in the future, as homogeneous platforms, relying on advanced compilers and runtimes to identify individual acceleratable tasks and manage their targeting.

Overall, the current state-of-the-art and perspectives depicted above underline a dramatic and strategic need of standardization in the different considered directions, so that accelerated architectures will be as naturally employable and manageable as general-purpose ones. In some directions, this process has already started (e.g., data-parallel and ML ecosystems), in others the first steps are being done and tested (e.g., Unified Virtual Memory initiative), while in others no concrete standardization progresses have been done yet (e.g., programming and program orchestration support in PIM architectures).

Finally, we presented some related work that shares our intention of addressing accelerators in general, albeit the majority focus on subsets like single domains or particular accelerator types.

Overall, this work discussed accelerators from various points of view, both vertically, concentrating on the characteristics of single families and models, and horizontally, discussing topics that involve accelerators as a whole. It can serve both as an introduction to professionals that want to approach the accelerator world, and as a comprehensive analysis of the current state of accelerator research, with prospective directions for the future.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.sysarc.2022.102561.

Biagio Peccerillo reports financial support was provided by Huawei Technologies Research & Development (UK) Limited. Co-author Andrea Mondelli is currently employed by Huawei Technologies Research & Development (UK) Limited.

Acknowledgments

This research has been funded by Huawei Technologies Research & Development (UK) Limited.

References

[1] W. Haensch, E.J. Nowak, R.H. Dennard, P.M. Solomon, A. Bryant, O.H. Dokumaci, A. Kumar, X. Wang, J.B. Johnson, M.V. Fischetti, Silicon CMOS devices beyond scaling, IBM J. Res. Dev. 50 (4.5) (2006) 339–361.

[2] M. Bohr, A 30 year retrospective on Dennard's MOSFET scaling paper, IEEE Solid-State Circuits Soc. Newslett. 12 (1) (2007) 11–13.

[3] D.A. Patterson, J.L. Hennessy, Computer Organization and Design RISC-V Edition: the Hardware Software Interface, Elsevier Science, 2017.

[4] J.L. Hennessy, D.A. Patterson, A new golden age for computer architecture, Commun. ACM 62 (2) (2019) 48–60.

[5] H. Esmaeilzadeh, E. Blem, R.S. Amant, K. Sankaralingam, D. Burger, Dark silicon and the end of multicore scaling, in: 2011 38th Annual International Symposium on Computer Architecture, ISCA, 2011, pp. 365–376.

[6] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, Reprinted from the AFIPS conference proceedings, Vol. 30 (Atlantic City, N.J., Apr. 18-20), AFIPS press, Reston, va., 1967, pp. 483-485, IEEE Solid-State Circuits Soc. Newslett. 12 (3) (2007) 19–20.

[7] J.L. Hennessy, D.A. Patterson, Computer Architecture, Sixth Edition: A Quantitative Approach, sixth ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.

[8] M. Zahran, Heterogeneous computing: Here to stay, Queue 14 (6) (2016) 31–42.

[9] S. Patel, W.-M. Hwu, Accelerator architectures, IEEE Micro 28 (2008) 4–12.

[10] T. Nowatzki, V. Gangadhar, N. Ardalani, K. Sankaralingam, Stream-dataflow acceleration, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture, ISCA, 2017, pp. 416–429.

[11] G. Pfister, Why accelerators now? 2009, http://perilsofparallel.blogspot.com/2009/07/why-accelerators-now.html.

[12] W.J. Dally, Y. Turakhia, S. Han, Domain-specific hardware accelerators, Commun. ACM 63 (7) (2020).

[13] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, D. Glasco, GPUs and the future of parallel computing, IEEE Micro 31 (5) (2011) 7–17.

[14] Intel, Intel stratix 10 FPGAs & SoC FPGA, www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html.

[15] X. Li, T. Li, ECOMIPS: An economic MIPS CPU design on FPGA, in: 4th IEEE International Workshop on System-on-Chip for Real-Time Applications, 2004, pp. 291–294.

[16] S. Druva Kumar, P. Sharma, K. Prajwal Shenoy, S.S. Naik, A.S. Lewis, Implementation of 16-bit hack CPU on FPGA, in: 2020 4th International Conference on Intelligent Computing and Control Systems, ICICCS, 2020, pp. 555–559.

[17] K. Papadimitriou, A. Dollas, S. Hauck, Performance of partial reconfiguration in FPGA systems: A survey and a cost model, ACM Trans. Reconfigurable Technol. Syst. 4 (4) (2011).

[18] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, S. Wei, A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications, ACM Comput. Surv. 52 (6) (2019).

[19] Y. Xue, P. Cronin, C. Yang, J. Hu, Non-volatile memories in FPGAs: Exploiting logic similarity to accelerate reconfiguration and increase programming cycles, in: 2015 IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC, 2015, pp. 92–97.

[20] Y. Chen, J. Emer, V. Sze, Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA, 2016, pp. 367–379.

[21] V. Sze, Y. Chen, T. Yang, J.S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE 105 (12) (2017) 2295–2329.

[22] R. Buchty, V. Heuveline, W. Karl, J.-P. Weiss, A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators, Concurr. Comput.: Pract. Exper. 24 (7) (2012) 663–675.

[23] D. Fujiki, X. Wang, A. Subramaniyan, R. Das, In-/Near-Memory Computing, 2021, pp. 1–124.

[24] S. Dave, Y. Kim, S. Avancha, K. Lee, A. Shrivastava, DMazeRunner: Executing perfectly nested loops on dataflow accelerators, ACM Trans. Embed. Comput. Syst. 18 (5s) (2019).

[25] A. Munshi, B. Gaster, T.G. Mattson, J. Fung, D. Ginsburg, OpenCL Programming Guide, first ed., Addison-Wesley Professional, 2011.

[26] J.B. Dennis, D.P. Misunas, A computer architecture for highly parallel signal processing, in: Proceedings of the 1974 Annual ACM Conference - Volume 2, ACM '74, Association for Computing Machinery, New York, NY, USA, 1974, pp. 402–409.

[27] J.B. Dennis, D.P. Misunas, A preliminary architecture for a basic data-flow processor, in: Proceedings of the 2nd Annual Symposium on Computer Architecture, ISCA '75, Association for Computing Machinery, New York, NY, USA, 1974, pp. 126–132.

[28] J.B. Dennis, First version of a data flow procedure language, in: B. Robinet (Ed.), Programming Symposium, Springer, Berlin, Heidelberg, 1974, pp. 362–376.

[29] B. Furht, Handbook of Augmented Reality, Springer Publishing Company, Incorporated, 2011.

[30] T. Huang, Computer vision: Evolution and promise, 1996, URL https://cds.cern.ch/record/400313.

[31] R.L. Rivest, Cryptography, computers in, in: Encyclopedia of Computer Science, John Wiley and Sons Ltd., GBR, 2003, pp. 468–470.

[32] Oracle, What is a database? 2022, URL https://www.oracle.com/database/what-is-database/.

[33] Y. Turakhia, G. Bejerano, W.J. Dally, Darwin: A Genomics co-processor provides up to 15,000X acceleration on long read assembly, in: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 199–213.

[34] J.F. Hughes, A. van Dam, M. McGuire, D.F. Sklar, J.D. Foley, S.K. Feiner, K. Akeley, Computer Graphics: Principles and Practice, third ed., Addison-Wesley Professional, 2014.

[35] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines-A survey, Proc. IEEE 84 (8) (1996) 1090–1123.

[36] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, D. Burger, A cloud-scale acceleration architecture, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE, 2016, pp. 1–13.

[37] S.-W. Hwang, S. Kim, Y. He, S. Elnikety, S. Choi, Prediction and predictability for search query acceleration, ACM Trans. Web 10 (3) (2016).

[38] S. Karandikar, C. Leary, C. Kennelly, J. Zhao, D. Parimi, B. Nikolic, K. Asanovic, P. Ranganathan, A hardware accelerator for protocol buffers, in: MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 462–478.

[39] S. Gong, J. Li, W. Lu, G. Yan, X. Li, ShuntFlow: An efficient and scalable dataflow accelerator architecture for streaming applications, in: 2019 56th ACM/IEEE Design Automation Conference, DAC, 2019, pp. 1–6.

[40] I. Stamoulias, M. Möller, R. Miedema, C. Strydis, C. Kachris, D. Soudris, High-performance hardware accelerators for solving ordinary differential equations, in: Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, HEART2017, Association for Computing Machinery, New York, NY, USA, 2017.

[41] J. Kung, Y. Long, D. Kim, S. Mukhopadhyay, A programmable hardware accelerator for simulating dynamical systems, ACM ACM SIGARCH Comput. Archit. News 45 (2) (2017) 403–415.

[42] G.A. Gillani, A. Krapukhin, A.B.J. Kokkeler, Energy-efficient approximate least squares accelerator: A case study of radio astronomy calibration processing, in: Proceedings of the 16th ACM International Conference on Computing Frontiers, CF '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 358–365.

[43] Y. Huang, N. Guo, M. Seok, Y. Tsividis, S. Sethumadhavan, Evaluation of an analog accelerator for linear algebra, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 570–582.

[44] L. Duch, S. Basu, M. Peón-Quirós, G. Ansaloni, L. Pozzi, D. Atienza, I-DPs CGRA: An interleaved-datapaths reconfigurable accelerator for embedded bio-signal processing, IEEE Embed. Syst. Lett. 11 (2) (2019) 50–53.

[45] R. Taranco, J.-M. Arnau, A. González, A low-power hardware accelerator for ORB feature extraction in self-driving cars, in: 2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, 2021, pp. 11–21.

[46] CCIX Consortium, CCIX Website, 2021, https://www.ccixconsortium.com/.

[47] CCIX Consortium, An introduction to CCIX - white paper, 2019, https://www.ccixconsortium.com/wp-content/uploads/2019/11/CCIX-White-Paper-Rev111219.pdf.

[48] Hybrid Memory Cube Consortium, Hybrid memory cube specification 1.0, Tech. Rep. (2013).

[49] Hybrid Memory Cube Consortium, Hybrid memory cube specification 2.0, Tech. Rep. (2014).

[50] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, X. Qian, GraphP: Reducing communication for PIM-based graph processing with efficient data partition, in: 2018 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2018, pp. 544–557.

[51] W. Zhao, T. Jin, C. Chen, S. Tavallaei, Z. Wu, OCP Accelerator Module Design Specification, Open Compute Project, 2019, v1.0.

[52] B. Brett, Multi-channel DRAM (MCDRAM) and high-bandwidth memory (HBM), 2016, https://www.intel.com/content/www/us/en/developer/articles/technical/multi-channel-dram-mcdram-and-high-bandwidth-memory-hbm.html.

[53] Jedec Solid State Technology Association, High bandwidth memory (HBM) DRAM, 2015, https://composter.com.ua/documents/JESD235A.pdf.

[54] H.-S.P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F.T. Chen, M.-J. Tsai, Metal-oxide RRAM, Proc. IEEE 100 (6) (2012) 1951–1970.

[55] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill -Smith, M. Krounbi, Spin-transfer torque magnetic random access memory (STT-MRAM), ACM J. Emerg. Technol. Comput. Syst. 9 (2) (2013).

[56] Google Brain Team, TensorFlow core, 2015, https://www.tensorflow.org/overview.

[57] Khronos OpenCL Working Group, The OpenCL specification, version 1.2, 2012, URL https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf.

[58] Khronos OpenCL Working Group, The OpenCL specification, version 2.1, 2018, URL https://www.khronos.org/registry/OpenCL/specs/opencl-2.1.pdf.

[59] Khronos OpenCL Working Group, SYCL Provisional specification, version 1.2.1, 2019, URL https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf.

[60] OpenMP Architecture Review Board, OpenMP application program interface, 2013, www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf.

[61] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Prentice-Hall Inc., USA, 1996.

[62] P.J. Menchini, An introduction to VHDL, in: J.P. Mermet (Ed.), Fundamentals and Standards in Hardware Description Languages, Springer Netherlands, Dordrecht, 1993, pp. 359–384.

[63] Khronos Group, Vulkan - Industry forged, 2021, https://www.khronos.org/vulkan/.

[64] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, 2014, arXiv:1408.5093.

[65] NVIDIA, CUDA C Programming guide, 2019, URL docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[66] Qualcomm, Hexagon DSP SDK, 2021, https://developer.qualcomm.com/software/hexagon-dsp-sdk.

[67] Qualcomm, Qualcomm neural processing SDK for AI, 2021, https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk.

[68] Samsung, Samsung neural SDK, https://developer.samsung.com/neural/overview.html.

[69] UPMEM, UPMEM SDK, https://sdk.upmem.com/.

[70] Xilinx, Xilinx vitis - Unified software platform for all developers, 2021, https://www.xilinx.com/products/design-tools/vitis.html.

[71] Intel, Intel quartus prime software suite, https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html.

[72] Huawei, HUAWEI HiAI DDK Quick start, 2020, https://developer.huawei.com/consumer/en/doc/2020105.

[73] Coral, PyCoral API overview, 2020, https://coral.ai/docs/reference/py/.

[74] AMD, RDNA Architecture, 2019, https://www.amd.com/system/files/documents/rdna-whitepaper.pdf.

[75] TechPowerUp, AMD Radeon RX 5300, 2020, https://www.techpowerup.com/gpu-specs/radeon-rx-5300.c3584.

[76] TechPowerUp, AMD Radeon RX 5600 XT, 2020, https://www.techpowerup.com/gpu-specs/radeon-rx-5600-xt.c3474.

[77] TechPowerUp, AMD Radeon RX 5700 XT, 2019, https://www.techpowerup.com/gpu-specs/radeon-rx-5700-xt.c3339.

[78] AMD, AMD RDNA 2, 2021, https://www.amd.com/en/technologies/rdna-2.

[79] TechPowerUp, AMD Radeon RX 6800 XT, 2020, https://www.techpowerup.com/gpu-specs/radeon-rx-6800-xt.c3694.

[80] TechPowerUp, AMD Radeon RX 6900 XT, 2020, https://www.techpowerup.com/gpu-specs/radeon-rx-6900-xt.c3481.

[81] Arm, ARM Mali GPU datasheet, 2021, https://developer.arm.com/-/media/Arm Developer Community/PDF/Mali GPU datasheet/Arm Mali GPU Datasheet 2021.2.pdf.

[82] Arm, Mali-G71, 2016, developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-g71-gpu.

[83] Arm, Mali-G72, 2017, developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-g72-gpu.

[84] Arm, Mali-G76, 2018, developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-g76-gpu.

[85] J. Davies, The Bifrost GPU architecture and the ARM Mali-G71 GPU, in: 2016 IEEE Hot Chips 28 Symposium, HCS, 2016, pp. 1–31.

[86] Arm, Mali-G77, 2019, developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-g77-gpu.

[87] Arm, Mali-G78, 2020, www.arm.com/products/silicon-ip-multimedia/gpu/mali-g78.

[88] Intel, Arria 10 FPGAs & SoCs, www.intel.com/content/www/us/en/products/details/fpga/arria/10.html.

[89] Intel, Arria 10 product table, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/arria-10-product-table.pdf.

[90] Intel, Arria 10 device datasheet, 2020, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_datasheet.pdf.

[91] Intel, Cyclone 10 FPGA, www.intel.com/content/www/us/en/products/details/fpga/cyclone/10.html.

[92] Intel, Cyclone 10 GX device datasheet, 2018, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10gx-51002.pdf.

[93] Intel, Intel MAX 10 FPGA, www.intel.com/content/www/us/en/products/details/fpga/max/10.html.

[94] Intel, Intel stratix 10 GX/SX product table, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-product-table.pdf.

[95] A. Davidson, A New FPGA architecture and leading-edge FinFET process technology promise to meet next-generation system requirements, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01220-hyperflex-architecture-fpga-socs.pdf.

[96] Intel, Intel stratix 10 device datasheet, 2021, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_datasheet.pdf.

[97] M. Langhammer, E. Nurvitadhi, B. Pasca, S. Gribok, Stratix 10 NX architecture and applications, in: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 57–67.

[98] Intel, Intel agilex f-series FPGAs & SoCs, www.intel.com/content/www/us/en/products/details/fpga/agilex/f-series.html.

[99] Intel, Intel agilex device datasheet, 2021, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/agilex/ag_datasheet.pdf.

[100] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, I. Ganusov, Architectural enhancements in intel®agilex™FPGAs, in: Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 140–149.

[101] Intel, Arria V FPGAs and SoC FPGAs. www.intel.com/content/www/us/en/products/details/fpga/arria/v.html.

[102] Intel, Arria V device datasheet, 2019, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-v/av_51002.pdf.

[103] Intel, Cyclone V FPGAs and SoC FPGAs. www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html.

[104] Intel, Cyclone V device datasheet, 2019, www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51002.pdf.

[105] Intel, Intel processor graphics Gen11 architecture, 2019, https://software.intel.com/sites/default/files/managed/db/88/The-Architecture-of-Intel-Processor-Graphics-Gen11_R1new.pdf.

[106] TechPowerUp, Intel UHD graphics G1, 2019, https://www.techpowerup.com/gpu-specs/uhd-graphics-g1.c3447.

[107] TechPowerUp, Intel Iris graphics G4, 2019, https://www.techpowerup.com/gpu-specs/iris-plus-graphics-g4.c3647.

[108] TechPowerUp, Intel UHD graphics G7, 2019, https://www.techpowerup.com/gpu-specs/iris-plus-graphics-g7.c3444.

[109] R. Smith, The intel Xe-LP GPU architecture deep dive: Building up the next generation, 2020, https://www.anandtech.com/show/15973/the-intel-xelp-gpu-architecture-deep-dive-building-up-from-the-bottom.

[110] TechPowerUp, Intel UHD graphics 730, 2021, https://www.techpowerup.com/gpu-specs/uhd-graphics-730.c3765.

[111] TechPowerUp, Intel UHD graphics 770, 2021, https://www.techpowerup.com/gpu-specs/uhd-graphics-770.c3844.

[112] TechPowerUp, Intel Iris Xe graphics G7 96EU, 2020, https://www.techpowerup.com/gpu-specs/iris-xe-graphics-g7-96eu.c3677.

[113] O. Wechsler, M. Behar, B. Daga, Spring hill (NNP-I 1000) Intel's data center inference chip, in: 2019 IEEE Hot Chips 31 Symposium, HCS, 2019, pp. 1–12.

[114] Intel, Intel nervana neural network processor for inference (Intel Nervana NNP-I). https://www.mouser.cn/pdfDocs/16433-1_NNP-announce_NNP-I_brief_v51.pdf.

[115] WikiChip, Neural network processors (NNP) - Intel nervana, https://en.wikichip.org/wiki/nervana/nnp.

[116] WikiChip, NNP-I 1100 - Intel nervana, https://en.wikichip.org/wiki/nervana/nnp/nnp-i_1100.

[117] WikiChip, NNP-I 1300 - Intel nervana. https://en.wikichip.org/wiki/nervana/nnp/nnp-i_1300.

[118] B. Hickmann, J. Chen, M. Rotzin, A. Yang, M. Urbanski, S. Avancha, Intel Nervana Neural Network Processor-T (NNP-T) fused floating point many-term dot product, in: 2020 IEEE 27th Symposium on Computer Arithmetic, ARITH, 2020, pp. 133–136.

[119] Intel, Intel nervana neural network processor for training (Intel Nervana NNP-T. https://en.wikichip.org/w/images/4/40/16433-1_NNP-announce_NNP-T_brief_v4.3.pdf.

[120] WikiChip, NNP-T 1300 - Intel Nervana. https://en.wikichip.org/wiki/nervana/nnp/nnp-t_1300.

[121] WikiChip, NNP-T 1400 - Intel Nervana, https://en.wikichip.org/wiki/nervana/nnp/nnp-t_1400.

[122] Intel, Intel Xeon Phi processor 7210, 2016, https://ark.intel.com/content/www/us/en/ark/products/94033/intel-xeon-phi-processor-7210-16gb-1-30-ghz-64-core.html.

[123] Intel, Intel Xeon Phi processor 7250, 2016, https://ark.intel.com/content/www/us/en/ark/products/94035/intel-xeon-phi-processor-7250-16gb-1-40-ghz-68-core.html.

[124] Intel, Intel Xeon Phi processor 7290, 2016, https://ark.intel.com/content/www/us/en/ark/products/95830/intel-xeon-phi-processor-7290-16gb-1-50-ghz-72-core.html.

[125] S. Mittal, A survey on evaluating and optimizing performance of Intel Xeon Phi, Concurr. Comput. Prac. Exper. 32 (19) (2020).

[126] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.-C. Liu, Knights landing: Second-generation Intel Xeon Phi product, IEEE Micro 36 (2) (2016) 34–46.

[127] Intel, Intel Xeon Phi processor 7235, 2017, https://ark.intel.com/content/www/us/en/ark/products/128694/intel-xeon-phi-processor-7235-16gb-1-3-ghz-64-core.html.

[128] Intel, Intel Xeon Phi processor 7285, 2017, https://ark.intel.com/content/www/us/en/ark/products/128691/intel-xeon-phi-processor-7285-16gb-1-3-ghz-68-core.html.

[129] Intel, Intel Xeon Phi processor 7295, 2017, https://ark.intel.com/content/www/us/en/ark/products/128690/intel-xeon-phi-processor-7295-16gb-1-5-ghz-72-core.html.

[130] NEC Corporation, SX-aurora TSUBASA architecture guide revision 1.1, 2018, https://www.hpc.nec/documents/guide/pdfs/Aurora_ISA_guide.pdf.

[131] NEC Corporation, NEC SX-Aurora TSUBASA - Vector engine, https://www.nec.com/en/global/solutions/hpc/sx/vector_engine.html.

[132] K. Komatsu, S. Momose, Y. Isobe, O. Watanabe, A. Musa, M. Yokokawa, T. Aoyama, M. Sato, H. Kobayashi, Performance evaluation of a vector supercomputer SX-aurora TSUBASA, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18, IEEE Press, 2018.

[133] NEC Corporation, NEC Vector supercomputer - SX-aurora TSUBASA generation 2, 2020, www.nec.com/en/global/solutions/hpc/sx/docs/SX_Aurora_TSUBASA_brochure_2020_oct.pdf.

[134] NEC Corporation - AI Platform Division, An evolved and brand new technology - SX-aurora TSUBASA present & future, 2020, wscad.sbc.org.br/2020/artigos/palestras/slides-nec.pdf.

[135] NVIDIA, NVIDIA Turing GPU architecture, 2018, https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[136] R. Smith, The NVIDIA GeForce RTX 2070 super & RTX 2060 super review: Smaller numbers, bigger performance, 2019, https://www.anandtech.com/show/14586/geforce-rtx-2070-super-rtx-2060-super-review.

[137] R. Smith, NVIDIA Unveils "Titan RTX" video card: 2500 USD turing tensor terror out later this month, 2018, https://www.anandtech.com/show/13668/nvidia-unveils-rtx-titan-2500-top-turing.

[138] NVIDIA, NVIDIA Ampere GA102 GPU architecture, 2020, https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf.

[139] NVIDIA, GeForce RTX 3060 family, 2021, https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3060-3060ti/.

[140] NVIDIA, GeForce RTX 3070 family, 2021, https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3070-3070ti/.

[141] NVIDIA, GeForce RTX 3090, 2021, https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3090/.

[142] Xilinx, Spartan-7, 2018, https://www.xilinx.com/support/documentation/product-briefs/spartan-7-product-brief.pdf.

[143] Xilinx, Spartan-7 FPGAs: Meeting the cost-sensitive market requirements, 2020, https://www.xilinx.com/support/documentation/white_papers/wp483-spartan-7-intro.pdf.

[144] Xilinx, Total power advantage using spartan-7 FPGAs, 2017, https://www.xilinx.com/support/documentation/white_papers/wp488-spartan7-power.pdf.

[145] Xilinx, 7 series FPGAs clocking resources, 2018, https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[146] Xilinx, 7 series - product selection guide, 2020, https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf.

[147] Xilinx, UltraScale Architecture: Highest device utilization, performance, and scalability, 2015, https://www.xilinx.com/support/documentation/white_papers/wp455-utilization.pdf.

[148] Xilinx, UltraScale Architecture and product data sheet: Overview, 2021, https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[149] Xilinx, UltraScale Architecture DSP slice, 2020, https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

[150] Xilinx, Kintex ultrascale+ FPGAs, 2020, https://www.xilinx.com/support/documentation/product-briefs/kintex-ultrascale-plus-product-brief.pdf.

[151] Xilinx, Virtex ultrascale+ FPGAs, 2018, https://www.xilinx.com/support/documentation/product-briefs/virtex-ultrascale-plus-product-brief.pdf.

[152] Xilinx, Versal: The first adaptive compute acceleration platform (ACAP), 2020, https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf.

[153] B. Gaide, D. Gaitonde, C. Ravishankar, T. Bauer, Xilinx adaptive compute acceleration platform: Versal™architecture, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 84–93.

[154] Xilinx, Versal architecture and product data sheet: Overview, 2021, https://www.xilinx.com/support/documentation/data_sheets/ds950-versal-overview.pdf.

[155] Comtech EF Data Corporation, AHA371/AHA372 - PCI Express compression and decompression accelerator card, 2014, http://www.aha.com/Uploads/aha371-372_brief_rev_d1.pdf.

[156] Businesswire, AHA Announces 10 and 20 Gigabit per second GZIP accelerators, 2014, https://www.businesswire.com/news/home/20140828005942/en/AHA-Announces-10-and-20-Gigabit-Per-Second-GZIP-Accelerators.

[157] Comtech EF Data Corporation, AHA374/AHA378 - PCI Express compression and decompression accelerator card, 2015, http://www.aha.com/Uploads/aha374-378_brief_rev_c1.pdf.

[158] L. Promberger, R. Schwemmer, H. Fröning, Assessing the overhead of offloading compression tasks, in: 49th International Conference on Parallel Processing, ICPP Workshops '20, Association for Computing Machinery, New York, NY, USA, 2020.

[159] Businesswire, AHA Announces 40 and 80 Gigabit per sec GZIP accelerators, 2015, https://www.businesswire.com/news/home/20150408006381/en/AHA-Announces-40-and-80-Gigabit-Per-Sec-GZIP-Accelerators.

[160] Comtech EF Data Corporation, AHA604/AHA605 - RSA key and compression accelerators, 2016, http://www.aha.com/Uploads/aha604_605_brief_rev_c.pdf.

[161] J. Cross, Inside Apple's A13 bionic system-on-chip, 2019, https://www.macworld.com/article/233354/inside-apples-a13-bionic-system-on-chip.html.

[162] WikiChip, A13 bionic - Apple, https://en.wikichip.org/wiki/apple/ax/a13.

[163] A. Frumusanu, The Apple iPhone 11, 11 Pro & 11 Pro Max review: Performance, battery, & camera elevated, 2019, https://www.anandtech.com/show/14892/the-apple-iphone-11-pro-and-max-review/6.

[164] J. Cross, A14 bionic FAQ: What you need to know about Apple's 5nm processor, 2020, https://www.macworld.com/article/234595/a14-bionic-faq-performance-features-cpu-gpu-neural-engine.html.

[165] WikiChip, A14 bionic - Apple. https://en.wikichip.org/wiki/apple/ax/a14.

[166] A. Frumusanu, Apple announces the Apple silicon M1: Ditching x86 – What to expect, based on A14, 2020, https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive.

[167] J. Ouyang, X. Du, Y. Ma, J. Liu, 3.3 Kunlun: A 14nm high-performance AI processor for diversified workloads, in: 2021 IEEE International Solid- State Circuits Conference, vol. 64, ISSCC, 2021, pp. 50–51.

[168] J. Ouyang, M. Noh, Y. Wang, W. Qi, Y. Ma, C. Gu, S. Kim, K.-i. Hong, W.-K. Bae, Z. Zhao, J. Wang, P. Wu, X. Gong, J. Shi, H. Zhu, X. Du, Baidu Kunlun an AI processor for diversified workloads, in: 2020 IEEE Hot Chips 32 Symposium, HCS, 2020, pp. 1–18.

[169] R. Kaplan, L. Yavits, R. Ginosasr, BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data, in: Proceedings of the 13th ACM International Systems and Storage Conference, SYSTOR '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 36–48.

[170] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, Y. Chen, Cambricon-X: An accelerator for sparse neural networks, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2016, pp. 1–12.

[171] T. Chou, W. Tang, J. Botimer, Z. Zhang, CASCADE: Connecting RRAMs to extend analog dataflow in an end-to-end in-memory processing paradigm, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52, Association for Computing Machinery, New York, NY, USA, 2019, pp. 114–125.

[172] Cerebras, The future of AI is here, https://cerebras.net/chip/.

[173] S.K. Moore, Huge chip smashes deep learning's speed barrier, IEEE Spectr. 57 (1) (2020) 24–27.

[174] Wafer-scale deep learning, in: 2019 IEEE Hot Chips 31 Symposium, HCS, 2019, pp. 1–31.

[175] Coral, USB accelerator datasheet, 2019, https://coral.ai/static/files/Coral-USB-Accelerator-datasheet.pdf.

[176] Google, Edge TPU, https://cloud.google.com/edge-tpu/.

[177] Google, Edge TPU performance benchmarks, 2020, https://coral.ai/docs/edgetpu/benchmarks/.

[178] Q-engineering, Google coral edge TPU explained in depth, 2021, https://qengineering.eu/google-corals-tpu-explained.html.

[179] A. Biswas, A.P. Chandrakasan, Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications, in: 2018 IEEE International Solid - State Circuits Conference, ISSCC, 2018, pp. 488–490.

[180] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, DaDianNao: A machine-learning supercomputer, in: IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 609–622.

[181] Y. Chen, T. Chen, Z. Xu, N. Sun, O. Temam, DianNao Family: Energy-efficient hardware accelerators for machine learning, Commun. ACM 59 (11) (2016) 105–112.

[182] Y. Turakhia, S.D. Goenka, G. Bejerano, W.J. Dally, Darwin-WGA: A co-processor provides increased sensitivity in whole genome alignments with high speedup, in: IEEE International Symposium on High Performance Computer Architecture, 2019, pp. 359–372.

[183] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, O. Temam, DianNao: A Small-footprint high-throughput accelerator for ubiquitous machine-learning, ACM SIGARCH Comput. Archit. News 42 (1) (2014) 269–284.

[184] M. Kang, S.K. Gonugondla, A. Patil, N.R. Shanbhag, A multi-functional in-memory inference processor using a standard 6T SRAM array, IEEE J. Solid-State Circuits 53 (2) (2018) 642–655.

[185] M. Kang, S. Lim, S. Gonugondla, N.R. Shanbhag, An in-memory VLSI architecture for convolutional neural networks, IEEE J. Emerg. Sel. Top. Circuits Syst. 8 (3) (2018) 494–505.

[186] S. Li, D. Niu, K.T. Malladi, H. Zheng, B. Brennan, Y. Xie, DRISA: A DRAM-based reconfigurable in-situ accelerator, in: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-50 '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 288–301.

[187] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, T. Rosing, DUAL: Acceleration of clustering algorithms using digital-based processing in-memory, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2020, pp. 356–371.

[188] Y. Chen, T. Krishna, J. Emer, V. Sze, 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, in: 2016 IEEE International Solid-State Circuits Conference, ISSCC, 2016, pp. 262–263.

[189] Y. Chen, T. Yang, J. Emer, V. Sze, Eyeriss V2: A flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 292–308.

[190] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, X. Li, FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks, in: IEEE Int. Symposium on High Performance Computer Architecture, 2017, pp. 553–564.

[191] M. Imani, S. Gupta, Y. Kim, T. Rosing, Floatpim: In-memory acceleration of deep neural network training with high precision, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 802–815.

[192] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, Y. Xie, FPSA: A Full system stack solution for reconfigurable reram-based NN accelerator architecture, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 733–747.

[193] A. Nag, C.N. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, P.-E. Gaillardon, GenCache: LEveraging in-cache operators for efficient sequence alignment, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52, Association for Computing Machinery, New York, NY, USA, 2019, pp. 334–346.

[194] J. Redgrave, A. Meixner, N. Goulding-Hotta, A. Vasilyev, O. Shacham, Pixel visual core: Google's fully programmable image, vision, and AI processor for mobile devices, 2018, https://old.hotchips.org/hc30/1conf/1.02_Google_HC30.Google.JasonRedgrave.V01.pdf.

[195] WikiChip, Pixel visual core (PVC) - Google. https://en.wikichip.org/wiki/google/pixel_visual_core.

[196] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T.V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C.R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D.H. Yoon, In-datacenter performance analysis of a tensor processing unit, ACM SIGARCH Comput. Archit. News 45 (2) (2017) 1–12.

[197] Google, Cloud tensor processing units (TPUs), https://cloud.google.com/tpu/docs/tpus.

[198] P. Teich, Tearing apart Google's TPU 3.0 AI coprocessor, 2018, https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/.

[199] D. Patterson, Domain-specific architectures for deep neural networks, 2019, https://inst.eecs.berkeley.edu/~cs152/sp19/lectures/L20-DSA.pdf.

[200] Graphcore, C2 IPU PCIe card, 2020, https://www.graphcore.ai/hubfs/assets/pdf/C2 Card Product Brief.pdf.

[201] Graphcore, The IPU-machine: IPU-M2000. https://www.graphcore.ai/products/mk2/ipu-m2000-ipu-pod4.

[202] Graphcore, IPU-Machine: M2000 - datasheet, 2020, https://docs.graphcore.ai/projects/graphcore-ipu-m2000-datasheet/en/1.0.0/_downloads/3324c02e071a26a432e11753f3fd092f/IPU-Machine_M2000_datasheet.pdf.

[203] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, H. Yang, GraphH: A processing-in-memory architecture for large-scale graph processing, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 38 (4) (2019) 640–653.

[204] T.J. Ham, L. Wu, N. Sundaram, N. Satish, M. Martonosi, Graphicionado: A high-performance and energy-efficient accelerator for graph analytics, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2016, pp. 1–13.

[205] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, X. Qian, GraphQ: Scalable PIM-based graph processing, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52, Association for Computing Machinery, New York, NY, USA, 2019, pp. 712–725.

[206] L. Song, Y. Zhuo, X. Qian, H. Li, Y. Chen, GraphR: Accelerating graph processing using ReRAM, in: 2018 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2018, pp. 531–543.

[207] J.S. Kim, D. Senol Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, O. Mutlu, GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies, BMC Genomics 19 (2) (2018) 89.

[208] Groq, Groq, https://groq.com/.

[209] Groq, GroqCard Accelerator, 2021, https://groq.com/GroqDocs/Product%20Spec%20Sheet%20-%20GroqCard%E2%84%A2%20Accelerator.pdf.

[210] D. Abts, et al., Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads, in: Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA '20, IEEE Press, 2020, pp. 145–158.

[211] Hailo, Hailo-8 AI processor. https://hailo.ai/product-hailo/hailo-8/.

[212] J. Liu, H. Zhao, M.A. Ogleari, D. Li, J. Zhao, Processing-in-memory for energy-efficient neural network training: A heterogeneous approach, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2018, pp. 655–668.

[213] L. Liu, Z. Li, C. Yang, C. Deng, S. Yin, S. Wei, HReA: An energy-efficient embedded dynamically reconfigurable fabric for 13-dwarfs processing, IEEE Trans. Circuits Syst. II Express Briefs 65 (3) (2018) 381–385.

[214] M. Gao, C. Kozyrakis, HRL: Efficient and flexible reconfigurable logic for near-data processing, in: 2016 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2016, pp. 126–137.

[215] Huawei, Atlas 200 AI accelerator module. https://e.huawei.com/en/products/cloud-computing-dc/atlas/atlas-200-ai.

[216] G. Fan, Atlas: Opening the door to AI with massive computing power, Communicate (86) (2018) 36–38.

[217] Huawei, Atlas AI computing solutions, 2020, https://e.huawei.com/en/material/datacenter/server/740747871439431a81843da6906181d8.

[218] Huawei, DaVinci: A scalable architecture for neural network computing, 2020, https://www.cmc.ca/wp-content/uploads/2020/03/Zhan-Xu-Huawei.pdf.

[219] Huawei, Atlas 300I inference card, https://e.huawei.com/en/products/cloud-computing-dc/atlas/atlas-300-ai.

[220] Huawei, Atlas 300T training card, https://e.huawei.com/en/products/cloud-computing-dc/atlas/atlas-300t-training-9000.

[221] HiSilicon, Kirin 9000, https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-9000.

[222] A. Frumusanu, Huawei announces mate 40 series: Powered by 15.3bn transistors 5nm Kirin 9000, 2020, https://www.anandtech.com/show/16156/huawei-announces-mate-40-series.

[223] D. Wenshuan, Driving AI to new horizons, Communicate (86) (2018) 4–11.
[224] HiSilicon, Kirin 990 5G, https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-990-5G.
[225] A. Frumusanu, The Huawei mate 30 pro review: Top hardware without Google? 2019, https://www.anandtech.com/show/15099/the-huawei-mate-30-pro-review-top-hardware-without-google.
[226] D. Fujiki, S. Mahlke, R. Das, In-memory data parallel processor, in: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–14.
[227] H. Labs, Goya inference platform white paper, 2019, https://habana.ai/wp-content/uploads/pdf/habana_labs_goya_whitepaper.pdf.
[228] H. Labs, Gaudi training platform white paper, 2019, https://habana.ai/whitepaper/Gaudi.pdf.
[229] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J.P. Strachan, M. Hu, R.S. Williams, V. Srikumar, ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA, 2016, pp. 14–26.
[230] H. Mao, M. Song, T. Li, Y. Dai, J. Shu, LerGAN: A zero-free, low data movement and PIM-based GAN architecture, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2018, pp. 669–681.
[231] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, H. Noyes, An efficient and scalable semiconductor architecture for parallel automata processing, IEEE Trans. Parallel Distrib. Syst. 25 (12) (2014) 3088–3098.
[232] K. Wang, K. Angstadt, C. Bo, N. Brunelle, E. Sadredini, T. Tracy, J. Wadden, M. Stan, K. Skadron, An overview of Micron's automata processor, in: 2016 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS, 2016, pp. 1–3.
[233] A. Subramaniyan, R. Das, Parallel automata processor, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture, ISCA, 2017, pp. 600–612.
[234] I. Roy, A. Srivastava, S. Aluru, Programming techniques for the automata processor, in: 2016 45th International Conference on Parallel Processing, ICPP, 2016, pp. 205–210.
[235] S. Mittal, A survey on applications and architectural-optimizations of Micron's automata processor, J. Syst. Archit. 98 (2019) 135–164.
[236] Microsoft, Project catapult, 2018, https://www.microsoft.com/en-us/research/project/project-catapult/.
[237] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, E. Chung, Accelerating Deep Convolutional Neural Networks Using Specialized Hardware, Microsoft Research, 2015.
[238] H. Valavi, P.J. Ramadge, E. Nestler, N. Verma, A mixed-signal binarized convolutional-neural-network accelerator integrating dense weight storage and multiplication for reduced data movement, in: 2018 IEEE Symposium on VLSI Circuits, 2018, pp. 141–142.
[239] C.-X. Xue, W.-H. Chen, J.-S. Liu, J.-F. Li, W.-Y. Lin, W.-E. Lin, J.-H. Wang, W.-C. Wei, T.-W. Chang, T.-C. Chang, T.-Y. Huang, H.-Y. Kao, S.-Y. Wei, Y.-C. Chiu, C.-Y. Lee, C.-C. Lo, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, M.-F. Chang, 24.1 A 1Mb Multibit ReRAM computing-in-memory macro with 14.6ns parallel MAC computing time for CNN based AI edge processors, in: 2019 IEEE International Solid- State Circuits Conference, ISSCC, 2019, pp. 388–390.
[240] H. Kim, J. Sim, Y. Choi, L.-S. Kim, NAND-Net: Minimizing computational complexity of in-memory processing for binary neural networks, in: 2019 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2019, pp. 661–673.
[241] A. Farmahini-Farahani, J.H. Ahn, K. Morrow, N.S. Kim, NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules, in: 2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA, 2015, pp. 283–295.
[242] W. Huangfu, K.T. Malladi, S. Li, P. Gu, Y. Xie, NEST: DIMM Based near-data-processing accelerator for K-mer counting, in: Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD '20, Association for Computing Machinery, New York, NY, USA, 2020.
[243] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, R. Das, Neural Cache: Bit-serial in-cache acceleration of deep neural networks, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, 2018, pp. 383–396.
[244] D. Kim, J. Kung, S. Chai, S. Yalamanchili, S. Mukhopadhyay, Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 380–392.
[245] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang, T.-H. Hsu, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, M.-F. Chang, A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors, in: 2018 IEEE International Solid - State Circuits Conference, ISSCC, 2018, pp. 494–496.
[246] J. Lee, J. Lee, NP-CGRA: Extending CGRAs for efficient processing of light-weight deep neural networks, in: 2021 Design, Automation and Test in Europe Conference and Exhibition, DATE, 2021, pp. 1408–1413.
[247] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, L. Benini, Origami: A convolutional network accelerator, in: Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, GLSVLSI '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 199–204.
[248] L. Cavigelli, L. Benini, Origami: A 803-GOp/s/W convolutional network accelerator, IEEE Trans. Circuits Syst. Video Technol. 27 (11) (2017) 2461–2475.
[249] L. Song, X. Qian, H. Li, Y. Chen, PipeLayer: A pipelined ReRAM-based accelerator for deep learning, in: 2017 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2017, pp. 541–552.
[250] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, K. Olukotun, Plasticine: A reconfigurable architecture for parallel patterns, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture, ISCA, 2017, pp. 389–402.
[251] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, G. De Micheli, The programmable logic-in-memory (PLiM) computer, in: 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE, 2016, pp. 427–432.
[252] M. Gao, G. Ayers, C. Kozyrakis, Practical near-data processing for in-memory analytics frameworks, in: 2015 International Conference on Parallel Architecture and Compilation, PACT, 2015, pp. 113–124.
[253] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, Y. Xie, PRIME: A Novel processing-in-memory architecture for neural network computation in ReRAM-based main memory, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 27–39.
[254] P. Srivastava, M. Kang, S.K. Gonugondla, S. Lim, J. Choi, V. Adve, N.S. Kim, N. Shanbhag, PROMISE: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, 2018, pp. 43–56.
[255] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, Y. Chen, PuDianNao: A polyvalent machine learning accelerator, in: ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, 2015, pp. 369–381.
[256] A. Ankit, I.E. Hajj, S.R. Chalamalasetti, G. Ndu, M. Foltin, R.S. Williams, P. Faraboschi, W.-m.W. Hwu, J.P. Strachan, K. Roy, D.S. Milojicic, PUMA: A Programmable ultra-efficient memristor-based accelerator for machine learning inference, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 715–731.
[257] O. Akbari, M. Kamal, A. Afzali -Kusha, M. Pedram, M. Shafique, PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture, in: 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE, 2018, pp. 413–418.
[258] L. Wu, A. Lottarini, T.K. Paine, M.A. Kim, K.A. Ross, The Q100 database processing unit, IEEE Micro 35 (3) (2015) 34–46.
[259] L. Wu, A. Lottarini, T.K. Paine, M.A. Kim, K.A. Ross, Q100: The architecture and design of a database processing unit, SIGPLAN Not. 49 (4) (2014) 255–268.
[260] Qualcomm, Snapdragon 865, www.qualcomm.com/products/snapdragon-865-5g-mobile-platform.
[261] A. Frumusanu, The snapdragon 865 performance preview: Setting the stage for flagship Android 2020, 2019, https://www.anandtech.com/show/15207/the-snapdragon-865-performance-preview-setting-the-stage-for-flagship-android-2020.
[262] L. Codrescu, Qualcomm hexagon DSP: An architecture optimized for mobile and multimedia communications, 2013, https://developer.qualcomm.com/qfile/27696/qualcomm-hexagon-architecture.pdf.
[263] Qualcomm, Snapdragon 888, www.qualcomm.com/products/snapdragon-888-5g-mobile-platform.
[264] A. Frumusanu, Qualcomm details the snapdragon 888: 3rd gen 5G & Cortex-Xl on 5nm, 2020, https://www.anandtech.com/show/16271/qualcomm-snapdragon-888-deep-dive/2.
[265] W. Huangfu, S. Li, X. Hu, Y. Xie, RADAR: A 3D-ReRAM based DNA alignment accelerator architecture, in: Proceedings of the 55th Annual Design Automation Conference, DAC '18, Association for Computing Machinery, New York, NY, USA, 2018.
[266] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, T. Rosing, RAPID: A ReRAM processing in-memory architecture for DNA sequence alignment, in: 2019 IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED, 2019, pp. 1–6.
[267] L. Liu, C. Deng, D. Wang, M. Zhu, S. Yin, P. Cao, S. Wei, An energy-efficient coarse-grained dynamically reconfigurable fabric for multiple-standard video decoding applications, in: Proceedings of the IEEE 2013 Custom Integrated Circuits Conference, 2013, pp. 1–4.
[268] S.K. Gonugondla, M. Kang, N. Shanbhag, A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip training, in: 2018 IEEE International Solid - State Circuits Conference, ISSCC, 2018, pp. 490–492.

[269] J. Song, Y. Cho, J. Park, J. Jang, S. Lee, J. Song, J. Lee, I. Kang, An 11.5 TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc, in: 2019 IEEE International Solid-State Circuits Conference, ISSCC, 2019, pp. 130–132.

[270] Samsung, Mobile processor exynos 9825, 2019, https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9825/.

[271] Samsung, Mobile processor exynos 990, 2020, https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-990/.

[272] J. Yang, Y. Kong, Z. Wang, Y. Liu, B. Wang, S. Yin, L. Shi, 24.4 Sandwich-RAM: An energy-efficient in-memory BWN architecture with pulse-width modulation, in: 2019 IEEE International Solid- State Circuits Conference, ISSCC, 2019, pp. 394–396.

[273] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, O. Temam, ShiDianNao: Shifting vision processing closer to the sensor, in: ACM/IEEE International Symposium on Computer Architecture, ISCA, 2015, pp. 92–104.

[274] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, H.-P. Li, Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 236–249.

[275] S. Jain, A. Ranjan, K. Roy, A. Raghunathan, Computing in memory with spin-transfer torque magnetic RAM, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 26 (3) (2018) 470–483.

[276] Y. Kwon, Y. Lee, M. Rhu, 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 '52, Association for Computing Machinery, New York, NY, USA, 2019, pp. 740–753.

[277] WikiChip, FSD chip - Tesla, https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip.

[278] Ian Cutress, Hot chips 31 live blogs: Tesla solution for full self driving, 2019, https://www.anandtech.com/show/14766/hot-chips-31-live-blogs-tesla-solution-for-full-self-driving.

[279] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, A scalable processing-in-memory accelerator for parallel graph processing, in: International Symposium on Computer Architecture, ISCA, 2015, pp. 105–117.

[280] M. Gao, J. Pu, X. Yang, M. Horowitz, C. Kozyrakis, TETRIS: SCalable and efficient neural network acceleration with 3D memory, ACM SIGARCH Comput. Archit. News 45 (1) (2017) 751–764.

[281] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, H. Yang, TIME: A Training-in-memory architecture for RRAM-based deep neural networks, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 38 (5) (2019) 834–847.

[282] U.A. I., Untether AI - ushering in the PetaOPS era for artificial intelligence, 2021, https://www.untether.ai/.

[283] K. Morris, Untether AI - paddling PetaOps, 2020, https://www.eejournal.com/article/untether-ai-peddling-petaops/.

[284] L. Gwennap, Untether delivers at-memory AI, 2020, https://www.linleygroup.com/newsletters/newsletter_detail.php?num=6230.

[285] UPMEM, Compute where the data is and without inter-node transfers, https://www.upmem.com/technology/.

[286] F. Devaux, The true processing in memory accelerator, in: 2019 IEEE Hot Chips 31 Symposium, HCS, 2019, pp. 1–24.

[287] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, M. Shafique, X-CGRA: AN energy-efficient approximate coarse-grained reconfigurable architecture, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (10) (2020) 2558–2571.

[288] R. Andri, L. Cavigelli, D. Rossi, L. Benini, YodaNN: An architecture for ultralow power binary-weight CNN acceleration, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (1) (2018) 48–60.

[289] Intel, 8087 Math coprocessor, 1989, https://pdf1.alldatasheet.com/datasheet-pdf/view/90863/INTEL/8087.html.

[290] C.R. Johns, D.A. Brokenshire, Introduction to the cell broadband engine architecture, IBM J. Res. Dev. 51 (5) (2007) 503–519.

[291] S. Greengard, GPUs reshape computing, Commun. ACM 59 (9) (2016) 14–16.

[292] I. Skliarova, V. Sklyarov, FPGA-BASED Hardware accelerators, in: Lecture Notes in Electrical Engineering, vol. 566, Springer, Cham, 2019, p. XVI, 245.

[293] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, H. Shen, An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 28 (9) (2020) 1953–1965.

[294] N. Mohammedali, M.O. Agyeman, A study of reconfigurable accelerators for cloud computing, in: Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control, ISCSIC '18, Association for Computing Machinery, New York, NY, USA, 2018.

[295] M. Ledwon, B.F. Cockburn, J. Han, High-throughput FPGA-based hardware accelerators for deflate compression and decompression using high-level synthesis, IEEE Access 8 (2020) 62207–62217.

[296] J. Fowers, J.-Y. Kim, D. Burger, S. Hauck, A scalable high-bandwidth architecture for lossless compression on FPGAs, in: 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, 2015, pp. 52–59.

[297] S. Mittal, A survey of FPGA-based accelerators for convolutional neural networks, Neural Comput. Appl. 32 (4) (2020) 1109–1139.

[298] A.G. Blaiech, K. Ben Khalifa, C. Valderrama, M.A.C. Fernandes, M.H. Bedoui, A survey and taxonomy of FPGA-based deep learning accelerators, J. Syst. Archit. 98 (2019) 331–345.

[299] L. Liu, J. Luo, X. Deng, S. Li, FPGA-based acceleration of deep neural networks using high level method, in: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC, 2015, pp. 824–827.

[300] W.A. Wulf, S.A. McKee, Hitting the memory wall: Implications of the obvious, ACM SIGARCH Comput. Archit. News 23 (1) (1995) 20–24.

[301] Apple, M1, 2020, https://www.apple.com/uk/mac/m1/.

[302] D. Martin, Intel axes nervana AI chips in favor of Habana labs, 2020, https://www.crn.com/news/components-peripherals/intel-axes-nervana-ai-chips-in-favor-of-habana-labs.

[303] D. Giri, P. Mantovani, L.P. Carloni, Accelerators and coherence: An SoC perspective, IEEE Micro 38 (6) (2018) 36–45.

[304] C. Caşcaval, S. Chatterjee, H. Franke, K. Gildea, P. Pattnaik, A taxonomy of accelerator architectures and their programming models, IBM J. Res. Dev. 54 (2010) 5.

[305] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: AN imperative style, high-performance deep learning library, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2019.

[306] J. Bai, F. Lu, K. Zhang, et al., ONNX: Open Neural Network Exchange, GitHub, 2019, https://github.com/onnx/onnx.

[307] Y. Ma, D. Yu, T. Wu, H. Wang, PaddlePaddle: AN open-source deep learning platform from industrial practice, Front. Data Domputing 1 (1) (2019) 105–115.

[308] G.B. Team, TensorFlow - for mobile and IoT, https://www.tensorflow.org/lite.

[309] Khronos OpenCL Working Group, The OpenCL specification, version 2.0, 2015, URL https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf.

[310] J.-l. Gailly, P. Eggert, J. Meyering, GNU Gzip - summary, 2006, https://savannah.gnu.org/projects/gzip/.

[311] G. Roelofs, J.-l. Gailly, M. Adler, Zlib, 2017, https://zlib.net/.

[312] The Apache Software Foundation, Apache hadoop, 2021, https://hadoop.apache.org/.

[313] The OpenSSL Project, OpenSSL - cryptography and SSL/TLS toolkit, 1999, https://www.openssl.org/.

[314] AMD, ROCm, a new era in open GPU computing, 2016, https://rocm.github.io/index.html.

[315] OpenACC-Standard.org, The OpenACC application programming interface, 2019, https://www.openacc.org/sites/default/files/inline-images/Specification/OpenACC.3.0.pdf.

[316] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: A unified platform for task scheduling on heterogeneous multicore architectures, Concurr. Comput. Prac. Exper. 23 (2) (2011) 187–198.

[317] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, S. Amarasinghe, Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines, SIGPLAN Not. 48 (6) (2013) 519–530.

[318] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R.M. Badia, E. Ayguade, J. Labarta, Productive cluster programming with OmpSs, in: Proceedings of the 17th International Conference on Parallel Processing - Volume Part I, Euro-Par'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 555–566.

[319] Apple, Metal - Accelerating graphics and much more, 2021, https://developer.apple.com/metal/.

[320] Apple Inc., Core ML, https://developer.apple.com/machine-learning/core-ml/.

[321] J. Selig, The cerebras software development kit: A technical overview, 2022, https://f.hubspotusercontent30.net/hubfs/8968533/Cerebras SDK Technical Overview White Paper.pdf.

[322] Y. Ji, Y. Zhang, W. Chen, Y. Xie, Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler, SIGPLAN Not. 53 (2) (2018) 448–460.

[323] Google, Camera API, 2021, https://developer.android.com/guide/topics/media/camera.

[324] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[325] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 785–794.

[326] Graphcore, Poplar graph framework software, https://www.graphcore.ai/products/poplar.
[327] N. Sundaram, N. Satish, M.M.A. Patwary, S.R. Dulloor, M.J. Anderson, S.G. Vadlamudi, D. Das, P. Dubey, GraphMat: High performance graph analytics made productive, Proc. VLDB Endow. 8 (11) (2015) 1214–1225.
[328] Hailo, Dataflow compiler - A complete & scalable software toolchain, https://hailo.ai/product-hailo/hailo-dataflow-compiler/.
[329] Huawei, CANN chip enablement - Improving development efficiency to better match the Ascend chip enablement, https://e.huawei.com/en/products/cloud-computing-dc/atlas/cann.
[330] MindSpore, Programming guide, 2021, https://www.mindspore.cn/docs/programming_guide/en/master/index.html.
[331] Huawei, MindX SDK, https://support.huaweicloud.com/intl/en-us/mindxsdk/.
[332] Google, Neural networks API, 2021, https://developer.android.com/ndk/guides/neuralnetworks.
[333] Intel, DSP builder for intel FPGAs, www.intel.com/content/www/us/en/software/programmable/quartus-prime/dsp-builder.html.
[334] Intel, Intel high level synthesis compiler, https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html.
[335] Xilinx, Xilinx design flow for intel FPGA and SoC users, 2018, https://www.xilinx.com/support/documentation/sw_manuals/ug1192-xilinx-design-for-intel.pdf.
[336] Intel, Supported APIs for intel graphics controllers, 2021, https://www.intel.com/content/www/us/en/support/articles/000005524/graphics.html.
[337] Apache Incubator, A flexible and efficient library for deep learning, 2018, https://mxnet.apache.org/versions/1.9.0/.
[338] N. Rotem, J. Fix, S. Abdulrasool, S. Deng, R. Dzhabarov, J. Hegeman, R. Levenstein, B. Maher, N. Satish, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, Glow: Graph lowering compiler techniques for neural networks, 2018, CoRR, arXiv:1805.00907.
[339] Intel, OpenVINO - Deploy high-performance deep learning inference, 2021, https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html.
[340] Intel, nGraph, https://www.intel.com/content/www/us/en/artificial-intelligence/ngraph.html.
[341] S. Cyphers, A.K. Bansal, A. Bhiwandiwalla, J. Bobba, M. Brookhart, A. Chakraborty, W. Constable, C. Convey, L. Cook, O. Kanawi, R. Kimball, J. Knight, N. Korovaiko, V. Kumar, Y. Lao, C.R. Lishka, J. Menon, J. Myers, S.A. Narayana, A. Procter, T.J. Webb, Intel nGraph: An intermediate representation, compiler, and executor for deep learning, 2018, arXiv preprintarXiv:1801.08058.
[342] K. Angstadt, W. Weimer, K. Skadron, RAPID Programming of pattern-recognition processors, in: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 593–605.
[343] Synario, VHDL Reference manual, 1997, https://www.ics.uci.edu/~jmoorkan/vhdlref/Synario VHDL Manual.pdf.
[344] NEC Corporation, NEC SX-Aurora TSUBASA compilers, libraries and tools, 2021, www.nec.com/en/global/solutions/hpc/sx/tools.html?.
[345] J. Sanders, E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, first ed., Addison-Wesley Professional, 2010.
[346] MathWorks, MATLAB GPU computing support for NVIDIA CUDA-enabled GPUs, 2021, https://www.mathworks.com/solutions/gpu-computing.html.
[347] NVIDIA, DirectX 12, 2021, https://www.nvidia.com/en-us/geforce/technologies/dx12/.
[348] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, K. Olukotun, Automatic generation of efficient accelerators for reconfigurable hardware, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA, 2016, pp. 115–127.
[349] J. Talbot, R.M. Yoo, C. Kozyrakis, Phoenix++: Modular MapReduce for shared-memory systems, in: Proceedings of the Second International Workshop on MapReduce and Its Applications, MapReduce '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 9–16.
[350] M. Innes, E. Saba, K. Fischer, D. Gandhi, M.C. Rudilosso, N.M. Joy, T. Karmali, A. Pal, V. Shah, Fashionable modelling with flux, 2018, CoRR, arXiv:1811.01457.
[351] M. Innes, Flux: Elegant machine learning with Julia, J. Open Source Softw. (2018).
[352] Xilinx, Vivado design suite - HLx edition, 2021, https://www.xilinx.com/products/design-tools/vivado.html.
[353] Xilinx, Verilog reference guide, 1999, http://in.ncu.edu.tw/ncume_ee/digilogi/vhdl/Verilog_Reference_Guide.pdf.
[354] MathWorks, Xilinx FPGAs and Zynq SoCs - Model, verify, and program your algorithms on xilinx devices, 2021, https://www.mathworks.com/solutions/fpga-asic-soc-development/xilinx.html.
[355] H.C. Edwards, C.R. Trott, Kokkos: Enabling performance portability across manycore architectures, in: 2013 Extreme Scaling Workshop, XSW 2013, 2013, pp. 18–24.
[356] M. Steuwer, P. Kegel, S. Gorlatch, SkelCL - A portable skeleton library for high-level GPU programming, in: Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 1176–1182.
[357] M. Woo, J. Neider, T. Davis, D. Shreiner, OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, Addison-Wesley Longman Publishing Co., Inc., 1999.
[358] AMD, AMD And microsoft DirectX 12, 2021, https://www.amd.com/en/technologies/directx12.
[359] F. Chollet, et al., Keras, 2015, https://keras.io.
[360] B. Peccerillo, S. Bartolini, PHAST - A Portable high-level modern C++ programming library for GPUs and multi-cores, IEEE Trans. Parallel Distrib. Syst. 30 (1) (2019) 174–189.
[361] SiSoft, Intel arc alchemist graphics card lineup detailed, 2022, https://www.techpowerup.com/292252/intel-arc-alchemist-graphics-card-lineup-detailed.
[362] Intel, Intel stratix 10NX FPGAs, https://www.intel.it/content/www/it/it/products/details/fpga/stratix/10/nx.html.
[363] S.A. McKee, R.W. Wisniewski, Memory wall, in: D. Padua (Ed.), Encyclopedia of Parallel Computing, Springer US, Boston, MA, 2011, pp. 1110–1116.
[364] G. Bonshor, AMD releases milan-X CPUs with 3D V-cache: EPYC 7003 up to 64 cores and 768 MB L3 cache, 2022, https://www.anandtech.com/show/17323/amd-releases-milan-x-cpus-with-3d-vcache-epyc-7003.
[365] H.S. Stone, A logic-in-memory computer, IEEE Trans. Comput. C-19 (1) (1970) 73–78.
[366] P. Siegl, R. Buchty, M. Berekovic, Data-centric computing frontiers: A survey on processing-in-memory, in: Proceedings of the Second International Symposium on Memory Systems, MEMSYS '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 295–308.
[367] F. Gao, G. Tziantzioulis, D. Wentzlaff, ComputeDRAM: In-memory compute using off-the-shelf DRAMs, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52, Association for Computing Machinery, New York, NY, USA, 2019, pp. 100–113.
[368] X. Xin, Y. Zhang, J. Yang, ROC: DRAM-based processing with reduced operation cycles, in: Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19, Association for Computing Machinery, New York, NY, USA, 2019.
[369] A.B. Yoo, M.A. Jette, M. Grondona, SLURM: Simple linux utility for resource management, in: D. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 44–60.
[370] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running Dive Into the Future of Infrastructure, first ed., O'Reilly Media, Inc., 2017.
[371] A. García-Guirado, R. Fernández-Pascual, J.M. García, S. Bartolini, Managing resources dynamically in hybrid photonic-electronic networks-on-chip, Concurr. Comput.: Pract. Exper. 26 (15) (2014) 2530–2550.
[372] HP, The machine: A new kind of computer, https://www.hpl.hp.com/research/systems-research/themachine/.
[373] S. Bartolini, L. Benini, K. Bertels, S. Blanas, U. Brinkschulte, P. Carpenter, G. De Micheli, M. Duranton, B. Falsafi, D. Fey, S. Hamdioui, C. Hochberger, A. Mendelson, D. Meyer, I. Polian, U. Rückert, X. Salazar, W. Schindler, P. Stenstrom, T. Ungerer, Eurolab4HPC long-term vision on high-performance computing, in: T. Ungerer, P. Carpenter (Eds.), Fundamentals and Standards in Hardware Description Languages, second ed., 2020.
[374] Optalysys, The multiply and Fourier transform unit: A micro-scale optical processor, 2020, https://optalysys.com/s/Multiply_and_Fourier_Transform_white_paper_12_12_20.pdf.
[375] J. Cong, H. Huang, C. Ma, B. Xiao, P. Zhou, A fully pipelined and dynamically composable architecture of CGRA, in: 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, 2014, pp. 9–16.
[376] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019, pp. 1–84.
[377] C. Nicol, A coarse grain reconfigurable array (CGRA) for statically scheduled data flow computing, 2017, http://www.silicon-ukraine.com/public_materials/2017_10_08_msu_rountable/background/CGRA+Whitepaper.pdf.
[378] A.A.A. Donovan, B.W. Kernighan, The Go Programming Language, first ed., Addison-Wesley Professional, 2015.
[379] NVIDIA, NVIDIA TEsla V100 GPU architecture, 2017, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.
[380] S.M. Trimberger, Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology, Proc. IEEE 103 (3) (2015) 318–331.
[381] HSA Foundation, HSA Foundation, 2017, URL http://www.hsafoundation.com/.
[382] CXL Consortium, Compute express link: The breakthrough CPU-to-device interconnect, 2022, https://www.computeexpresslink.org/.
[383] Y. Hao, Z. Fang, G. Reinman, J. Cong, Supporting address translation for accelerator-centric architectures, in: 2017 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2017, pp. 37–48.

[384] P. Vogel, A. Marongiu, L. Benini, Lightweight virtual memory support for many-core accelerators in heterogeneous embedded SoCs, in: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS, 2015, pp. 45–54.
[385] S. Haria, M.D. Hill, M.M. Swift, Devirtualizing memory in heterogeneous systems, SIGPLAN Not. 53 (2) (2018) 637–650.
[386] N. Parris, Extended system coherency: Part 1 - Cache coherency fundamentals, 2013, https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/extended-system-coherency---part-1---cache-coherency-fundamentals.
[387] M. Dashti, A. Fedorova, Analyzing memory management methods on integrated CPU-GPU systems, in: Proceedings of the 2017 ACM SIGPLAN International Symposium on Memory Management, ISMM 2017, Association for Computing Machinery, New York, NY, USA, 2017, pp. 59–69.
[388] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K.T. Malladi, H. Zheng, O. Mutlu, CoNDA: Efficient cache coherence support for near-data accelerators, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 629–642.
[389] P. Boudier, G. Sellers, Memory system on fusion APUs - The benefits of zero copy, 2011, https://developer.amd.com/wordpress/media/2013/06/1004_final.pdf.
[390] J. Fang, S. Liu, X. Zhang, Research on cache partitioning and adaptive replacement policy for CPU-GPU heterogeneous processors, in: 2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, DCABES, 2017, pp. 19–22.
[391] J. Lee, H. Kim, TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture, in: IEEE International Symposium on High-Performance Comp Architecture, 2012, pp. 1–12.
[392] X. Wang, W. Zhang, Cache locking vs. partitioning for real-time computing on integrated CPU-GPU processors, in: 2016 IEEE 35th International Performance Computing and Communications Conference, IPCCC, 2016, pp. 1–8.
[393] J. Power, A. Basu, J. Gu, S. Puthoor, B.M. Beckmann, M.D. Hill, S.K. Reinhardt, D.A. Wood, Heterogeneous system coherence for integrated CPU-GPU systems, in: Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Association for Computing Machinery, New York, NY, USA, 2013, pp. 457–467.
[394] C. Kachris, B. Falsafi, D. Soudris, Hardware Accelerators in Data Centers, first ed., Springer Publishing Company, Incorporated, 2018.
[395] S. Yesil, M.M. Ozdal, T. Kim, A. Ayupov, S. Burns, O. Ozturk, Hardware accelerator design for data centers, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD '15, IEEE Press, 2015, pp. 770–775.
[396] B. Varghese, C. Reaño, F. Silla, Accelerator virtualization in fog computing: Moving from the cloud to the edge, IEEE Cloud Comput. 5 (6) (2018) 28–37.
[397] A. Spiridonov, New cloud TPU VMs make training your ML models on TPUs easier than ever, 2021, https://cloud.google.com/blog/products/compute/introducing-cloud-tpu-vms.
[398] H. Nasiri, M. Goudarzi, Dynamic FPGA-accelerator sharing among concurrently running virtual machines, in: 2016 IEEE East-West Design Test Symposium, EWDTS, 2016, pp. 1–4.
[399] Q. Zhao, M. Iida, T. Sueyoshi, A study of FPGA virtualization and accelerator scheduling, in: Proceedings of the First Workshop on Emerging Technologies for Software-Defined and Reconfigurable Hardware-Accelerated Cloud Datacenters, ETCD'17, Association for Computing Machinery, New York, NY, USA, 2017.
[400] M.H. Quraishi, E.B. Tavakoli, F. Ren, A survey of system architectures and techniques for FPGA virtualization, IEEE Trans. Parallel Distrib. Syst. 32 (9) (2021) 2216–2230.
[401] S. Gerangelos, N. Koziris, vPHI: Enabling Xeon Phi capabilities in virtual machines, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, 2017, pp. 1333–1340.
[402] C. Lee, S. Kim, C. Yoo, VADI: GPU virtualization for an automotive platform, IEEE Trans. Ind. Inf. 12 (1) (2016) 277–290.
[403] K. Hong, I. Jung, W. Ryu, J.K. Choi, A study on GPU virtualization in a virtualized server environment, in: 2014 International Conference on Information and Communication Technology Convergence, ICTC, 2014, pp. 472–473.
[404] X.-L. Wang, H. b. Wang, Y. Sang, Z.-L. Wang, Y.-W. Luo, Optimizing GPU virtualization with address mapping and delayed submission, in: 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst, HPCC, CSS, ICESS, 2014, pp. 413–416.
[405] A. Garg, P. Kulkarni, U. Kurkure, H. Sivaraman, L. Vu, Empirical analysis of hardware-assisted GPU virtualization, in: 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics, HiPC, 2019, pp. 395–405.
[406] U. Kurkure, H. Sivaraman, L. Vu, Virtualized GPUs in high performance datacenters, in: 2018 International Conference on High Performance Computing Simulation, HPCS, 2018, pp. 887–894.
[407] D. Vasilas, S. Gerangelos, N. Koziris, VGVM: Efficient GPU capabilities in virtual machines, in: 2016 International Conference on High Performance Computing Simulation, HPCS, 2016, pp. 637–644.
[408] H. Yu, A.M. Peters, A. Akshintala, C.J. Rossbach, Automatic virtualization of accelerators, in: Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 58–65.
[409] S. Govindarajan, K. Chitnis, M. Mody, G. Shurtz, S. Shivalingappa, T. Kim, Flexible and efficient sharing of high performance hardware accelerators in a safe, secure, virtualized system, in: 2020 IEEE International Conference on Consumer Electronics - Asia, ICCE-Asia, 2020, pp. 1–4.
[410] D. Spinellis, Z. Kotti, A. Mockus, A dataset for GitHub repository deduplication, in: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, 2020, pp. 523–527.
[411] ISO/IEC, Programming Languages — C++, Draft International Standard N4660, 2017, URL https://web.archive.org/web/20170325025026/http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4660.pdf.
[412] J.M. Andión, M. Arenaz, G. Rodríguez, J. Touriño, A novel compiler support for automatic parallelization on multicore systems, Parallel Comput. 39 (9) (2013) 442–460, Novel on-chip parallel architectures and software support.
[413] M. Wolfe, Parallelizing compilers, ACM Comput. Surv. 28 (1) (1996) 261–262.
[414] S. Apostolakis, Z. Xu, G. Chan, S. Campanoni, D.I. August, Perspective: A sensible approach to speculative automatic parallelization, in: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Association for Computing Machinery, New York, NY, USA, 2020, pp. 351–367.
[415] H.-S. Kim, Y.-H. Yoon, S.-O. Na, D.-S. Han, ICU-PFC: An automatic parallelizing compiler, in: Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, vol. 1, 2000, pp. 243–246.
[416] Z. Du, D.D. Ben -Dayan Rubin, Y. Chen, L. He, T. Chen, L. Zhang, C. Wu, O. Temam, Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches, in: Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, Association for Computing Machinery, New York, NY, USA, 2015, pp. 494–507.
[417] Z. Li, Y. Wang, T. Zhi, T. Chen, A survey of neural network accelerators, Front. Comput. Sci. 11 (5) (2017) 746–761.
[418] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, J. Kepner, Survey and benchmarking of machine learning accelerators, in: 2019 IEEE High Performance Extreme Computing Conference, HPEC, 2019, pp. 1–9.
[419] S. Umesh, S. Mittal, A survey of spintronic architectures for processing-in-memory and neural networks, J. Syst. Archit. 97 (2019) 349–372.
[420] S. Mittal, S. Umesh, A survey on hardware accelerators and optimization techniques for RNNs, J. Syst. Archit. 112 (2021) 101839.
[421] L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532.
[422] Y. Chen, Y. Xie, L. Song, F. Chen, T. Tang, A survey of accelerator architectures for deep neural networks, Engineering 6 (3) (2020) 264–274.
[423] D. Moolchandani, A. Kumar, S.R. Sarangi, Accelerating CNN inference on ASICs: A survey, J. Syst. Archit. 113 (2021) 101887.
[424] S. Mittal, Vibhu, A survey of accelerator architectures for 3D convolution neural networks, J. Syst. Archit. 115 (2021) 102041.
[425] Y. Shen, M. Ferdman, P. Milder, Maximizing CNN accelerator efficiency through resource partitioning, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 535–547.
[426] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, K. Asanović, Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators, ACM SIGARCH Comput. Archit. News 39 (3) (2011) 129–140.
[427] C. Gui, L. Zheng, B. He, C. Liu, X. Chen, X. Liao, H. Jin, A survey on graph processing accelerators: Challenges and opportunities, J. Comput. Sci. Tech. 34 (2) (2019) 339–371.
[428] J. Kurzak, D.A. Bader, J. Dongarra, Scientific Computing with Multicore and Accelerators, CRC Press, Inc., USA, 2010.
[429] A. Chattopadhyay, Ingredients of adaptability: A survey of reconfigurable processors, VLSI Des. 2013 (2013).
[430] R. Tessier, K. Pocek, A. DeHon, Reconfigurable computing architectures, Proc. IEEE 103 (3) (2015) 332–354.
[431] A. DeHon, Fundamental underpinnings of reconfigurable computing architectures, Proc. IEEE 103 (3) (2015) 355–378.
[432] M. Wijtvliet, L. Waeijen, H. Corporaal, Coarse grained reconfigurable architectures in the past 25 years: Overview and classification, in: 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS, 2016, pp. 235–244.
[433] S. Mittal, G. Verma, B. Kaushik, F.A. Khanday, A survey of SRAM-based in-memory computing techniques and applications, J. Syst. Archit. 119 (2021) 102276.

[434] K. Iniewski, Embedded Systems: Hardware, Design and Implementation, first ed., John Wiley and Sons Ltd., 111 River Street, Hoboken, NJ, USA, 2012.
[435] B. Moyer, Y. Watanabe, Chapter 13 - hardware accelerators, in: B. Moyer (Ed.), Real World Multicore Embedded Systems, Newnes, Oxford, 2013, pp. 447–480.
[436] J.M.P. Cardoso, J.G.d.F. Coutinho, P.C. Diniz, Embedded Computing for High Performance: Efficient Mapping of Computations using Customization, Code Transformations and Compilation, first ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.
[437] K.A. Hawick, D.P. Playne, Developmental directions in parallel accelerators, AusPDC '14, in: Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing, vol. 152, Australian Computer Society, Inc., AUS, 2014, pp. 21–27.
[438] E.G. Cota, P. Mantovani, G. Di Guglielmo, L.P. Carloni, An analysis of accelerator coupling in heterogeneous architectures, in: 2015 52nd ACM/EDAC/IEEE Design Automation Conference, DAC, 2015, pp. 1–6.
[439] A.L. Varbanescu, J. Shen, Heterogeneous computing with accelerators: An overview with examples, in: 2016 Forum on Specification and Design Languages, FDL, 2016, pp. 1–8.
[440] S. Margerm, A. Sharifian, A. Guha, A. Shriraman, G. Pokam, TAPAS: Generating parallel accelerators from parallel programs, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2018, pp. 245–257.
[441] L. Addazi, F. Ciccozzi, B. Lisper, Executable modelling for highly parallel accelerators, in: Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems, IEEE Press, 2019, pp. 318–321.
[442] A. Shawahna, S.M. Sait, A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, IEEE Access 7 (2019) 7823–7859.
[443] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, R. Das, Neural cache: Bit-serial in-cache acceleration of deep neural networks, in: Proceedings of the 45th Annual International Symposium on Computer Architecture, ISCA '18, IEEE Press, 2018, pp. 383–396.