X-Centric: A Survey on Compute-, Memory- and Application-Centric Computer Architectures

Sven Rheindt
sven.rheindt@tum.de
Technical University of Munich
Munich, Germany

Temur Sabirov
Technical University of Munich
Munich, Germany

Oliver Lenke
Technical University of Munich
Munich, Germany

Thomas Wild
Technical University of Munich
Munich, Germany

Andreas Herkersdorf
Technical University of Munich
Munich, Germany

ABSTRACT

Big Data and machine learning constitute the multifaceted challenge of computer engineering in the past decade. The meaningful processing of vast amounts of unstructured data from a myriad of sensors and devices is a complicated endeavor already. Aggravated by the need to enter the extremely power- and resource-constrained pocket-size mobile domain, the computing as we know it is rapidly evolving. Data-centric in- and near-memory computing, as well as highly heterogeneous accelerator-equipped application-centric architectures, are on the rise to tackle the unsatisfiable demand for evermore compute performance and efficiency.

To learn from these innovations, this paper surveys compute-, memory-, and application-centric architectures and related programming paradigms and analyzes prominent chances and challenges. The key insights from the particular domains are: 1) The high nominal processing performance of compute-centric systems is thwarted by massively decreasing data-to-task locality and increased data movement. Nevertheless, the commodity of shared-memory programming and the presence of widespread legacy applications keep this domain alive. 2) Memory-centric designs help to mitigate the data locality wall and significantly improve power and performance efficiency. However, a memory-centric programming paradigm is still missing. 3) Heterogeneity, customization, and established ecosystems (like for mobile devices) enable application-centric optimization under often tight thermal, power, and resource constraints. However, a holistic SoC-level design approach is required to utilize and program the diversity of processing units in different application domains efficiently.

A one-size-fits-all architecture approach seems not in sight because of the wide diversity in domain-specific requirements and constraints. Therefore, established ecosystems, 3D-stacked logic-enhanced memory devices, and commoditized architecture-aware programming models seem fundamental for performant and programmable future-proof computer architectures.

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422792

CCS CONCEPTS

• **Computer systems organization** → **Multicore architectures**;
• **Computing methodologies** → *Parallel programming languages.*

KEYWORDS

computer architecture, programming model, architecture evolution, memory-centric, application-centric, near-memory computing, survey, roofline model, heterogeneous architecture, mobile device

ACM Reference Format:
Sven Rheindt, Temur Sabirov, Oliver Lenke, Thomas Wild, and Andreas Herkersdorf. 2020. X-Centric: A Survey on Compute-, Memory- and Application-Centric Computer Architectures. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3422575.3422792

1 INTRODUCTION

"*The world's most valuable resource is no longer oil, but data.*" [135] is more than an article in the May 2017 issue of The Economist, asking for a regulation of the internet giants. It is pointing to the development of the rapidly growing global data sphere, which is, according to IDC market research, still in its infancy [64]. It is, however, not solely the vast amount of big data, supposedly 165-175 Zettabytes by 2025, that needs to be processed. The extraction of meaningful information out of the often highly unstructured data, originating from massively distributed sensors, mobile and IoT (Internet of Things) devices, cyber-physical systems, social networks, or multimedia applications, is an ever-growing challenge that needs to be dealt with [64, 118]. Especially since big data and machine learning-enabled data analytics in real-time have entered the pocket-size mobile domain, where computing resources are limited and highly power-constrained. Therefore, it is more important than ever to "*work smart not hard*" [64].

Performance scaling of computer architectures has been a ubiquitous and multifaceted problem throughout its history. There are many important aspects besides providing evermore nominal compute performance through homo- and heterogeneous multi- and many-core architectures with increased transistor and core counts: 1) How can those architectures be efficiently programmed to exploit the nominal compute performance? 2) How can they be properly utilized under a given thermal constraint? 3) How do the application data sets comply with the underlying processor, memory,

and interconnect architecture? 4) How to keep the architectures user-friendly in terms of programmability? And many more.

In the past, crucial architectural innovations were triggered by hitting several *walls* of computer architecture. The *memory wall* has led to sophisticated cache hierarchies and many micro architectural developments. The *power wall* (i.e., the end of Dennard scaling) was tackled by the shift from single- to multi- and many-core architectures and even bigger caches. Meanwhile, as described earlier, application requirements and the data sphere are rapidly changing over the last years. Application's data sets used to have plenty of spatial and temporal locality, thus being a good fit for caching. However, the emergence of application classes with large, unstructured, irregular, and thus cache-unfriendly data sets restricts their efficiency. Both changing application characteristics and today's heterogeneous many-core architectures, lead to a reduced data-to-task locality. At the MEMSYS 2017 conference, Peter Kogge, University of Notre Dame, showed evidence for this new *locality wall*, especially for the arising memory-intensive application [78]. For those, data transfers between processor cores and main memory contribute significantly to the overall energy consumption and performance limitations since accesses to global memory compared to local registers have several magnitudes higher energy costs and access latencies [9, 78, 118]. The data locality wall, being closely tied to the "von Neumann bottleneck", i.e., processors exhibiting limited memory bandwidth, has been further confirmed by several papers from industry and academia [14, 31, 45, 69, 79, 118].

Already in his 1977 ACM Turing award lecture, John Backus stated [10]: "*Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself but where to find it.*"

Ever since, yet especially in the past decade, this has lead to a major rethinking of computer architectural design. Instead of building mainly compute-centric architectures (i.e., designing the systems around the processor and fetching data close to it), more recent memory-centric designs leverage the increased bandwidth and proximity when processing in- or near-memory [78, 118, 125]. Besides this, and although compute-centric general-purpose computing still has a huge market, there is yet another trend towards heterogeneous multi- and many-core architectures with more power-efficient and application-specific accelerators [16, 65]. Prime examples are deep learning and AI accelerators in nearly all of today's high-end mobile devices. This application-centric design trend is a key enabler of real-time and power-efficient, pocket-size, smart data processing: face, speech, text, gesture and activity recognition, many natural language processing (NLP) tasks, computer vision problems, or even augmented reality are only a few examples [65].

Much can be learned from the innovations in all three domains. In the *compute-centric* domain, the authors of [16] provide a survey on general-purpose multi-core processors. By analyzing several commercial products, they concluded that there needs to be a balance between specialization and programmability. In 2008, the authors of [14] analyzed the "*Computing as We Know it*" in view of the upcoming exascale era with emerging memory-intensive application characteristics. They point out several challenges and key research areas for future systems that could enable a 1000× performance increase by 2015. However, the demand for the next 1000× is still not satisfied in 2020. Besides, the *memory-centric* domain has experienced a revival in the last decade. Already in 2013, the authors of [66] presented a taxonomy on processing-in-memory (PIM) and argue for studying fixed-function PIM architectures. The authors of [125] surveyed the evolution of processing-in-memory with a particular focus on memory technology. They elaborate on the differences between the first PIM attempts in the '90s and the innovations of the years prior to 2016. More recently, in 2019, the authors of [126] provided an overview and classification of existing near-memory computing architectures. They discuss arising challenges and open issues, focusing on cache coherence, virtual memory, the lack of programming models and data mapping schemes. Similarly, also the *application-centric* domain undergoes tremendous innovations in the past decade. The authors of [65] presented the evolution of mobile device AI accelerators and also compared them to desktop CPUs and GPUs. They further gave a perspective on future hardware and software developments in the mobile AI domain. However, they confine their analysis to deep learning in the Android ecosystem. Moreover, the authors of [96] have extensively reviewed diverse heterogeneous computing approaches from different perspectives in CPU-GPU systems. They also discussed challenges in achieving synergistic computing due to extremely different architectures and programming models.

All in all, these related surveys are often tied to a single domain. To the best of our knowledge, there is no other work covering all of these, trying to gain holistic inter-domain insights. Therefore, the goal of this paper is to:

- survey the evolution of compute-, memory-, and application-centric systems,
- elaborate on their architectures and programming models,
- apply the Roofline model to these domains, and
- qualitatively analyze their chances and challenges and provide valuable insights.

The paper is structured as follows: Section 2 provides background on the Roofline model and how it can be used to gain first architectural insights. It is followed by the survey and discussion of compute-, memory- and application-centric systems in Section 3, 4 and 5, respectively. Finally, the main insights are summarized in Section 6.

2 INSIGHTS FROM THE ROOFLINE MODEL

With the emergence of numerous diverse and evermore complex architectural design approaches, an intuitive and insightful model to analyze the influence of various architectural aspects is becoming exceedingly valuable. The Roofline model, presented by Williams et al. [138], provides a simple yet powerful means to analyze the performance of applications executed on a multi-core architecture. It further allows to obtain insights on critical system bottlenecks (i.e., to characterize whether a system is bound by its compute performance, or by memory bandwidth and access latencies), and to identify potential optimizations.

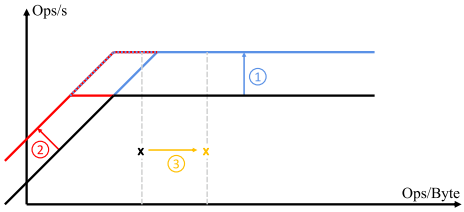

Figure 1: The original Roofline model based on Williams et al. [138].

The Roofline model contains two *roofs*: the horizontal roof represents the nominal compute performance in (floating-point) operations per second, and the diagonal roof denotes the effective streaming memory bandwidth. Both are considered as maximum achievable values, which can be determined e.g., by running dedicated streaming applications and microbenchmarks. Applications, consisting of operations and memory accesses, have a specific operational intensity (operations performed per Byte accessed from main memory: (Fl)Ops/Byte). This value is plotted on the horizontal axis of the diagram and projected vertically until it intersects one of the aforementioned roofs. The vertical intersection of a measured operating point with the horizontal or diagonal roof determines whether the application is compute- or memory-bound, respectively, when more Ops/s would be attained.

In order to analyze the impact of several architectural improvements, we slightly modify the Roofline model, compared to the definition of Williams et al. We assume an operating point being characterized by both dimensions, the operational intensity in Ops/Byte and the actual compute performance, denoted in Ops/s. Obviously, this operating point is always below both boundaries.

Figure 1 is based on the original Roofline model of Williams et al. and depicts how micro-architectural improvements of the processing units as well as the step from single-core to multi-core systems both increase the nominal compute performance. Thus, it leads to the blue shift of the horizontal boundary ①, which makes the system potentially more memory-bound, as the intersection of both roofs shifts to the right. Correspondingly, improvements in memory or interconnect technology effectuate a shift of the diagonal boundary, visualized in red ②, resulting in a higher compute boundedness of the system. The instantiation of caches reduces the overall number of Bytes to/from memory, and thus increases the Ops/Byte, which shifts the operating point to the right ③.

We propose to use the Roofline model to visualize the influence of typical design approaches in the compute-, memory- and application-centric domains. Throughout the paper, we apply it to exemplary architectures in these domains. We distinguish between improvements of the nominal efficiency of the system, characterized by the two roofs, and variations of the operating point.

3 COMPUTE-CENTRIC SYSTEMS

The Roofline model, as explained in Section 2, focuses on the compute performance and memory bandwidth, as the interplay between processors and memory is crucial for the overall system. Throughout the history of computer engineering, the majority of systems have been compute-centric [14]. Therefore, this survey begins with elaborating on the evolution of compute-centric architectures and their related programming paradigms and discusses several chances and challenges of such systems.

3.1 Evolution of Compute-Centric Systems

The driving forces for increasing single-core processor performance were process scaling, clock frequency scaling, along with supply and threshold voltage scaling, which have been on an exponential trend in the '90s [105]. However, the advantages of particular techniques were counteracted by the adverse effects of others.

Figure 2 depicts significant evolutionary steps of computer architectures. As memory speed increases significantly slower than processor speed, the gap between rapidly growing compute performance and moderately growing memory performance widens continually [108]. Computer architects tried to overcome this by leveraging and optimizing cache hierarchies and bringing data as close as possible to the processors (cf. Figure 2b). However, already in 1994, Wulf et al. pointed out that this will not fully solve the problem as the disparity grows exponentially [140]. Also, the off-chip memory bandwidth loss and the problem of highly distributed and thus cache-unfriendly data structures cannot be compensated well by caching [78, 125].

Despite the *memory wall*, processor performance kept increasing rapidly. As a result, power dissipation and energy consumption, along with increasing design complexity, became an insurmountable barrier for further improvement of clock frequencies [77]. This obstacle of compute-performance scaling is referred to as the *power wall*. As the demand to increase compute-performance continued to rise, companies were exploring different concepts to overcome the power wall. *Multi-core* processors, also referred to as *Chip Multiprocessors* (CMPs), have been introduced. They integrate multiple processing cores onto the same chip. Through the achieved parallelism, i.e., splitting the workload to multiple cores, a multi-core processor is able to perform more tasks while being driven with a lower voltage and frequency [20, 105], of course, bounded by Amdahl's law. Both homogeneous and heterogeneous architectures have seen much interest (cf. Figure 2c and Section 3.2).

Due to the increasing scaling complexity of conventional multi-cores, several recent approaches focus on tile-based or clustered architectures (cf. Figure 2d). These architectures can be scaled better than traditional multi-core processors, and hot-spots can be avoided more easily [137]. Nevertheless, potential data-to-task dis-localities arise when both processing nodes and memory are organized in a physically distributed manner. This uprising challenge was described by Kogge as the *locality wall* in 2017 [78].

3.2 Compute-Centric Architectures

As described above, compute-centric systems lived through a tremendous evolution. These days, a variety of both homogeneous and heterogeneous architectures (cf. Figure 2c) are still leveraged in mainstream chip multiprocessors. Whereas homogeneous architectures combine several identical cores in one system, heterogeneous systems benefit from a combination of different processing elements, such as *strong* and *weak* cores, or processors and additional accelerators. Besides, tile-based architectures, often equipped with a powerful NoC, are also deployed, mainly due to scalability reasons. This section provides an overview of different exemplary designs in these categories. Although tile-based architectures are covered under the umbrella of homogeneous systems, they also exist in a heterogeneous manner [59].

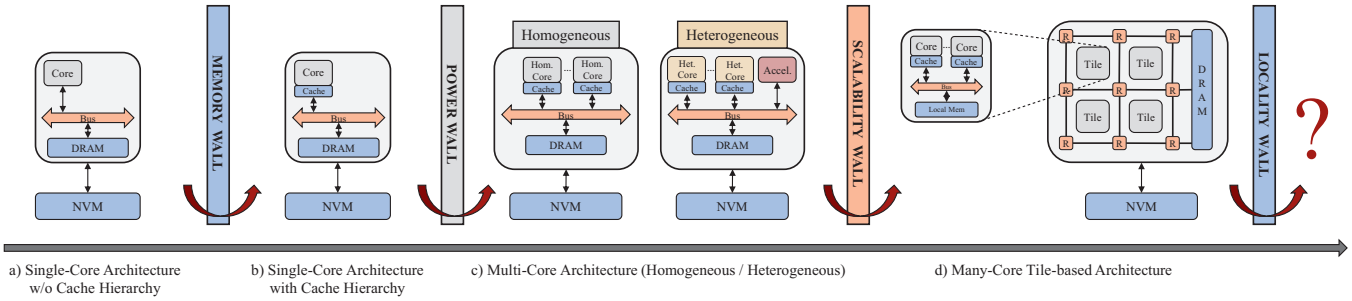

Figure 2: Evolution of compute-centric systems, including the major walls of computer architecture.

3.2.1 Homogeneous Architectures. For conventional homogeneous CMPs, there fundamentally exist two different approaches [77]. The *multi-core* approach integrates multiple (up to 64 [112]) powerful, multi-instruction issue, out-of-order homogeneous cores of the same kind into a single processor, pursuing the highest performance of a single or a few threads per core. The indicative examples of desktop multi-core processors are the Intel Core processor family and AMD Ryzen processors [46, 113]. In contrast, the *many-thread* approach utilizes a significantly larger number (up to several thousand for e.g., GPUs [117]) simple, narrow-issue, in-order cores, thus being oriented to throughput-critical parallel applications.

Since their announcement, homogeneous CMPs have also found their application in the server industry, as well as in clouds and clusters [21, 40]. Meanwhile, many homogeneous architectures in this context follow a *tile-based* or *clustered many-core* approach, which brings advantages in interconnect capacity and scalability.

Tilera's TILE64 architecture is a well-known prime example of a tile-based many-core architecture: it consists of 64 compute tiles, each equipped with a 3-way VLIW, general-purpose and power-efficient processor, and a 2-level cache hierarchy. The architecture provides a 5-channel, bi-directional interconnect in the form of a 2D-mesh network. An autonomous DMA unit is facing the potential problem of data-to-task dis-locality – it efficiently performs memory block copies, including cache-to-memory, memory-to-cache, and cache-to-cache copies [12]. Thus, due to low-power consumption and high throughput, Tilera's Tile-series architectures have been widely deployed in servers – addressing both rapidly increasing data volumes and increasing costs of hardware and power [13, 89].

Another example is Intel's Single Chip Cloud Computer (SCC) – a homogeneous *many-core* architecture with 48 cores [61]. This architecture connects 24 tiles over a NoC without hardware cache coherence. Each tile comprises two x86 cores and contains dedicated hardware buffers for message-passing inter-tile communication.

Intel further developed the Xeon Phi processor, another prominent example for a many-core architecture, which is widely used in High-Performance Computing (HPC) [25]. Based on the Intel Many Integrated Core (MIC) architecture [7], two product generations with different architectures, Knights Corner (KNC) and Knights Landing (KNL), were presented: KNC incorporates 57 to 61 in-order cores connected via a system bus and serves as a co-processor connected over PCIe. In contrast, KNL has up to 72 out-of-order multi-threaded cores, organized in tiles, and connected via 2D cache-coherent mesh interconnect [95]. This architecture is shipped in three different configurations – along with stand-alone homogeneous cores, as co-processor connected over PCIe, or in a heterogeneous setup by extending it with an integrated fabric.

Furthermore, Kalray presented a Massively Parallel Processor Array (MPPA) in 2013. Compared to the Tilera approach, this architecture contains 16 compute and one system core in each of the 16 clusters, which corresponds to 256 compute cores and four I/O subsystems in total. The clusters are interconnected by two bidirectional 2D-torus networks for data and control signals, respectively.

Nevertheless, ever-growing and diverse application requirements are pushing homogeneous architectures to their limits [83].

3.2.2 Heterogeneous Architectures. Besides the trend of scaling homogeneous architectures, the past decade revealed that a heterogeneous combination of diverse processing elements within one system is beneficial. These architectures incorporate cores of different size, complexity (e.g., instruction set architecture, thread- or instruction-level parallelism, and in-/out-of-order execution of instructions) and performance as well as additional accelerators. Concerning their structure, most heterogeneous architectures can be classified into *three* types [23]: 1) multiple homogeneous cores plus dedicated hardware accelerators, 2) asymmetry of processing cores, which provide either performance or power efficiency when necessary, and 3) a hybrid composition of the previous types, comprising accelerators, asymmetric along with specialized cores.

The most prominent example of heterogeneous computer architectures combining several identical cores with hardware accelerators is CPU-GPU systems. Mittal et al. stressed the benefit of these architectures compared to stand-alone CPU or GPU solutions as they combine different strengths in a flexible manner [97]. While CPUs are designed for high frequencies and large caches and thus are suited for latency-critical applications, GPUs run at a lower frequency and outperform CPUs in throughput-critical operations. Conventional CPU-GPU systems are designed initially as separate discrete CPUs and GPUs interconnected by a PCIe bus. Recently, as personal devices with emphasis on energy efficiency became popular, chips with a CPU and GPU fabricated on the same die have been introduced, which is then also referred to as *Accelerated Processing Unit* (APU). The first generation of APUs was introduced by AMD in 2011 (AMD Llano) [18]. Intel has presented their integrated graphic processors (IGPs) in Intel Sandy Bridge [144] and Ivy Bridge APUs in 2012 [32].

Another extension to conventional CPU systems is Field Programmable Gate Arrays (FPGAs). Nowadays, they are widely deployed in many domains such as bioinformatics [129], finance [101], digital signal processing [54, 111], as well as machine learning [51]

to accelerate computations. Similar to CPU-GPU systems, conventional discrete CPU-FPGA systems have also been shifted towards first on-package and later on-die integration. Intel's Agilex System-on-Chip (SoC) FPGAs provide multiple options of integration – being part of a 3D system-in-package with a multi-core processor or being attached to a system through a cache and memory coherent interconnect [139]. Moreover, a combination of several DSPs and an FPGA extension has been presented by Yan et al. [90].

Being introduced in 2005, the IBM Cell processor is an example of the second type of heterogeneous architecture that combines different cores (i.e., strong and weak) in one system. The cell processor, developed with a particular focus on video and image processing, is known from the Sony PlayStation 3 [22]. It is composed of one big IBM PowerPC processing element (PPE) and eight small synergistic processing elements (SPEs) connected through a coherent bus.

The third type of heterogeneous architectures can be mostly found in the mobile domain: besides a handful of specialized cores, these architectures are typically harnessing Arm's big.LITTLE CPU design [55], which will be covered extensively in Section 5 in the context of application-centric architectures.

3.3 Roofline Model for Tile-based Systems

The shift from traditional multi-cores to tile-based architectures helped to alleviate interconnect scalability issues. Limited memory bandwidth has been revealed to be a critical bottleneck of traditional systems. However, scalable tile-based architectures suffer from increased data movement and data-to-task dis-localities. To examine the influence of such architectural developments on the system's behavior, we extend the original Roofline model exemplary to tile-based systems. Similarly, an extended Roofline model for heterogeneous architectures is presented in Section 5.3.

Figure 3 visualizes the impact of the shift from traditional multi-cores to tile-based architectures on the nominal compute performance and the effective memory bandwidth of the system. Tile-based systems typically use a scalable NoC-interconnect, which provides a higher effective memory bandwidth per tile compared to a shared bus interconnect. Thus, the diagonal roof, which represents the maximum achievable memory bandwidth, is shifted to the left ①. Note that we apply the Roofline model to each tile separately. Assuming that the operational intensity ($Ops/Byte$) of the application remains constant, a higher memory bandwidth ($Byte/s$) also results in a higher effective compute performance (Ops/s). This does not change the horizontal roof, but the operating point shifts upwards ② since the nominal compute performance can be used more efficiently. Applications with a lower operational intensity (in $Ops/Byte$) are rather memory-intensive and thus profit potentially more from a higher memory bandwidth than compute-intensive applications with high operational intensity.

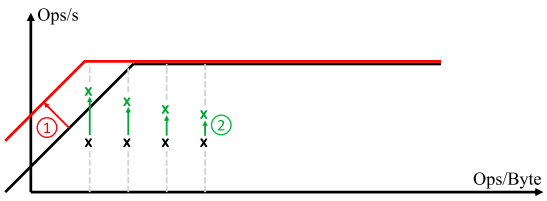

Figure 3: Roofline model for tile-based architectures.

3.4 Compute-Centric Programming

Over decades, advances in compute-performance were achieved by faster and more efficient single-core systems, which corresponded directly to an improved performance in *serial* applications. However, this effect saturated with the focus shift to optimizing multi- and many-core systems. Already in 2009, Feng et al. assumed that *"serial computing is now dead"* and new, parallel programming methods are the only possibilities to exploit the capabilities of modern architectures fully [50].

A programming model is an abstraction of a computer architecture [93]. It is closely tied to the system's *memory organization*, which can be classified into centralized shared memory (SM), distributed shared memory (DSM), and fully distributed memory (DM) architectures. The latter have a private address space per compute node. There exists a huge variety of programming models and languages: some very general-purpose, others tailored to a particular type of architecture. Tightly coupled to the programming models are also the maintenance of cache coherence and the provisioning of memory consistency.

Prior to the advent of parallel programming models [14, 36], *autoparallelization* has been the main workhorse of parallel computing, making excessive use of parallel hardware without any modification of the programs. However, the rise of pointer-dependent programming languages, as well as the continually growing level of parallelism, have led to the advent of real parallel programming models [14, 73].

3.4.1 Parallel Programming Models. This major rethinking of software development, resulting in new parallel programming models, is also referred to as *concurrency revolution* [36]. Besides the problem of dividing the algorithm into small and independent tasks, the programming model also has to comply with the underlying hardware architecture. Especially for inter-processor communication and synchronization, feasible solutions for different sorts of applications had to be found. The authors of [81] analyzed both shared memory and message passing concepts in the context of different application scenarios.

Shared-Memory Model. In the shared-memory model, threads of an application share a common address space and communicate on load-store granularity via the shared memory (SM). Fine-grained data-sharing and dynamic memory access behavior that cannot be foreseen by the compiler can be handled well with a shared memory model, as the programmer does not need to care about data movement [81]. This flexibility requires, however, a careful programming style to handle synchronization correctly and to avoid race conditions [11]. Data consistency has to be provided explicitly, e.g., by dedicated functions like mutual exclusion or semaphores of POSIX threads [47]. Instead of managing cache coherence at low levels, OpenMP, being an industry-standard API [30], can be used to ease parallel programming of shared-memory architectures. Ease of use, along with a high level of abstraction, has made OpenMP widely used in shared-memory architectures. However, besides difficult synchronization, shared memory programming is not an optimal solution if an application shares larger chunks of data.

Message-Passing Model. In contrast to the shared-memory model, distributed memory architectures (DM) do not provide a shared address space. Thus, communication between threads needs to be

established by sending and receiving explicit messages. By combining synchronization with data transfer, they are also well-suited for producer-consumer scenarios. The Message Passing Interface (MPI) has become a standard library for the message-passing programming model, which includes a full range of message-passing primitives [104]. To be executed on message-passing architectures, parallel programs, written in Fortran, C or C++, must only be linked with the MPI library, while being compiled with ordinary compilers. The message-passing model can be applied to both distributed and shared memory architectures [11]. Nevertheless, Kranz et al. argued for building multiprocessors, that comply with both communication strategies, to handle different scenarios efficiently and proposed a prototype of such an architecture [81]. What is still tricky in the context of modern programming languages is the need for serialization of complex and pointer-based data structures, as they cannot be sent directly within a message buffer [100].

Partitioned Global Address Space. The partitioned global address space (PGAS) model represents a mixture of the shared-memory and message-passing model, combining the advantages of each approach. The PGAS memory model is based on a DSM memory organization, harnessing the global address space to improve productivity, meanwhile exploiting data locality for better scalability of large-scale architectures and thus increasing the system performance [33]. The most prominent PGAS languages are Unified Parallel C [27], Co-array Fortran [103], and Titanium [142], each being an extension to a corresponding base language. More recent representatives of PGAS languages are X10 [119] and Chapel [37].

Heterogeneous Programming Paradigms. Since general-purpose computation on GPUs (GPGPUs) has appeared to exploit the potential of novel CPU-GPU systems also for non-graphics applications, several heterogeneous parallel programming (HPP) models have been presented by different industry companies [73]. They often require a different programming style as processing elements are usually in a hierarchical relationship, such as *host* and *device*. In 2006, NVIDIA introduced CUDA (Compute Unified Device Architecture) – a parallel programming model, which enables NVIDIA GPUs to be programmed in C, C++, Fortran, and other languages [68]. Similarly, OpenCL has become an open standard for parallel programming of heterogeneous architectures, including other available processing units like digital signal processors (DSP), field-programmable gate arrays (FPGA), or hardware accelerators [56]. These predominant HPP models (CUDA and OpenCL) have been classified by Belikov et al. as *low-level* programming models [11]. Programmers are burdened with detailed control of communication and different memory accesses. Initially developed by Khronos Group, OpenCL has been further adapted by leading industry companies (Apple, Intel, AMD, and Arm) to be leveraged in their products [56]. Higher-level programming models abstract the underlying GPU memory to the programmer and thus ease software prototyping and debugging [11]. An example of this would be OmpSs, which is a high-level extension to OpenCL [39].

3.4.2 Cache Coherence. With the presence of parallel programming, we are facing the tough challenge of keeping the shared caches of multiple cores coherent, also known as the *cache coherence problem*. Necessary cache write-back and invalidation commands need to be enforced to avoid an inconsistent memory view. A hardware coherence protocol generally performs better than a software implementation and further relieves the programmer from this duty [91]. However, Choi et al. doubt the existence of hardware on-chip coherence implementations in future many-core systems. They pointed out the lousy scaling behavior in terms of power consumption, verification complexity, and storage overhead [24]. In contrast, Martin et al. take it for granted that hardware coherence mechanisms will also be present in future systems. They indicate that the step from hardware coherence to a software implementation would not reduce the complexity but solely shift it to the software domain. A fundamental rethinking of operation systems and application design, all written mainly for cache-coherent systems, would be the consequence. Besides the performance superiority of hardware coherence protocols, they presented an approach of scalable hardware implementations. An analysis showed moderate scaling overheads in terms of interconnect traffic, storage costs, latencies, and power consumption [91]. Further, Srivatsa et al. proposed a flexible and dynamically re-configurable *region-based cache coherence* mechanism, which comprises only a subset of compute nodes depending on the current application scenario and coherence is only provided for this subset [130].

Further, the choice of a consistency model has a considerable impact on both the programming effort and the performance of the system [16]. In comparison to strong consistency models, relaxed consistency models (e.g., acquire-release) provide better performance at the cost of programming effort.

3.5 Discussion of Compute-Centric Systems

In the conclusion of this section, we discuss several chances and challenges of compute-centric systems and compare representative examples regarding the following essential design characteristics in Figure 4. *Compute Effectiveness* describes how effective the nominal compute performance is typically exploited. Limitations like Amdahl's law, synchronization overhead, or data-to-task dislocalities significantly reduce the compute effectiveness of a system. In contrast, *Data Access Efficiency* characterizes how efficiently the processor or other processing units can access data. In particular, latencies and bandwidth between core and memory are very crucial for data access efficiency. High *Programmability* corresponds to a low programming effort. For instance, software developers immensely benefit from high-level, architecture-unaware programming. An architecture with a high *Flexibility* can be deployed in a wide range of use-cases. *Power Efficiency* is also an important characteristic especially relevant for systems that are thermal- and energy-constrained.

Regarding the architectures mentioned above, traditional homogeneous architectures can be programmed easiest, as legacy and commodity parallel programming models can be applied without further adjustments. In contrast, heterogeneous and tile-based architectures require more complicated programming paradigms, mapping compute kernels to various processing units and establishing synchronization, respectively. Tile-based and heterogeneous compute-centric architectures provide higher compute performance than traditional multi-core designs, as the latter suffer from the limited scalability and memory bandwidth. However, the compute

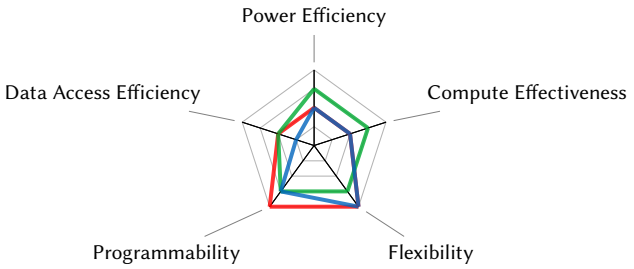

Figure 4: Comparison of different compute-centric architectures: Homogeneous Architecture (red), Heterogeneous Architecture (green), Tile-Based Architecture (blue)

effectiveness of tile-based systems diminishes as they suffer from data-to-task dis-localities. Heterogeneous architectures provide less flexibility than the other considered compute-centric architectures due to the inherent structure and specialization of equipped processing units. In terms of power efficiency, this specialization brings advantages. Apart from that, a general statement is hard to make. Data access efficiency is primarily dependent on the system's structure and architecture. Potentially missing last-level shared cache, distributed memory, and complicated coherence maintenance are diminishing the data access efficiency of compute-centric architectures. Nevertheless, the dedicated architectural design of numerous heterogeneous and tile-based architectures, mostly deployed in servers, elevates the data access efficiency of such systems.

In conclusion, compute-centric systems offer the following chances and challenges:

Chance CC+1: Compute-centric designs provide high nominal compute performance. A rapid evolution towards multi- and many-core architectures enabled the development of powerful systems. With the trend towards tile-based architectures, scalable solutions were found to combine hundreds of cores on a chip. However, to exploit this nominal compute performance, efficient solutions for synchronization and coherency challenges are mandatory.

Chance CC+2: Compute-centric systems have established programming paradigms. Commoditized programming models with the support of legacy code ease the programmability of compute-centric architectures. For both shared memory and message passing paradigms, programming standards have been widely established. The use of high-level programming languages facilitates the development of applications without being tied to specific hardware architectures.

Chance CC+3: The wide application area of compute-centric architectures is unrivaled. Many compute-centric designs are well-suited for general-purpose usage. High compute performance, in conjunction with potential out-of-the-box usage and suitability for most application scenarios, further emphasize compute-centric architectures.

Challenge CC-1: Increased data movement limits the benefits of high processor performance and restricts scalability. The traditional compute-centric design approach is based on data transfer between processor and memory. However, when scaling compute-centric systems, a hierarchy of distributed memories and distributed processing elements cannot be avoided. As data sets grow faster than memory and interconnect bandwidth, the *von Neumann bottleneck* limits the benefits of high nominal compute performance. Due to the emergence of highly pointer-based and cache-unfriendly data structures, caching is not able to recover the lost off-chip memory bandwidth. Thus, data traffic and synchronization are significant challenges and limit architecture scaling.

Challenge CC-2: Maintaining coherence and consistency is challenging. Increasing processor counts in a system and advanced interconnects worsen the synchronization and coherence traffic and management, especially for highly scalable systems.

Challenge CC-3: User-friendly programming paradigms do not fully exploit the architecture. More efficient programming would require another "revolution" of programming paradigms. In general, a trade-off is visible between an easy and thus widely usable or an efficient and architecture-aware programming model.

4 MEMORY-CENTRIC SYSTEMS

"*Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck.*" [10] This statement of John Backus from 1977 had to hover for almost 40 years until a major rethinking of computer architectural design occurred. Although the nominal compute-performance of processors enabled by Moore's law, the scalability of advanced interconnects, the hidden memory access latency through sophisticated cache hierarchies, and the avoidance of hot-spots in physically distributed memory systems brought computer engineering this far, there are still many challenges. Data movement in general and traffic of inter-process communication, in particular, are nowadays significant power consuming and performance hindering effects [9, 31, 78, 118]. This is worsened by the rapidly growing and immensely cache-unfriendly data sets of today's applications that far exceed the capacity of modern multi-level cache hierarchies [78, 118]. The architectural developments in conjunction with new application requirements lead to an ever growing data-to-task dis-locality, the so called *locality wall* [14, 78, 79, 118] (cf. Figure 2). As AMD stated in the Memory-Centric Architecture session of the DAC 2018 conference, the management of data locality and the understanding of data-to-task affinity are yet unsolved problems (for HPC applications) [45, 69].

Nevertheless, primarily the last decade has led to a breakthrough in mitigating this problem. Instead of building mainly compute-centric systems around the processor cores and thus fetching and caching data close to the processor, memory-centric architectures follow the reverse approach [78, 118, 125]. Instead of moving data, computations are performed close to or even in memory. Thus, in- and near-memory computing helps to bridge the widening gap between huge data sets and limited cache sizes, reduces data movement and interconnect traffic and thus lowers the energy footprint of applications.

This section elaborates on the evolution of in- and near-memory computing and analyzes the chances and challenges of several architectures and the corresponding programming paradigms.

4.1 Evolution of Memory Technology

The revival of in- and near-memory computing research has been facilitated by the groundbreaking innovations of memory technology in the past decade. Despite the continually increasing efficiency of DRAM architectures due to improved process technologies and

higher bus frequencies, this progress could not bridge the widening gap between rapidly scaling processor performance and moderately scaling memory speed [125]. According to Burger et al. and Patterson, this divergence and the limited bandwidth have been hindering the efficiency of computer architectures [19, 107].

As a first step to overcome the *bandwidth wall*, 2.5D integration techniques arose to combine multiple processing and memory modules on one silicon interposer [42]. Similarly, 3D-stacking techniques integrate several chips not only into one die but also vertically on top of each other. This reduces memory access latencies and, by decreasing costly off-chip accesses, increases the memory bandwidth significantly. Besides the complex manufacturing and testing processes of 3D-stacked devices, thermal issues of 3D integration are still major obstacles of this technology [74]. Nevertheless, already in 2013, AMD research recognized the enormous impact of new memory technologies, which can "*fundamentally change the landscape of the future computer architecture design*" [141]. Both technologies enable manufacturing chips with sub-modules in different process technologies [125].

Two prime examples of the latest 3D integration enabled memory architectures are the 2nd generation of *High Bandwidth Memory* (HBM) [67, 86] and the *Hybrid Memory Cube* (HMC) [26, 110]. The HBM2, being a JEDEC standard, is used as high bandwidth memory with up to 256 GB/s in many recent devices, e.g., in the Nvidia GA100 GPU [82] or the Intel Stratix 10 MX [35] and Agilex FPGAs [139]. In contrast, the HMC contains a 3D-stacked custom logic layer allowing for either direct integration of dedicated in-memory logic [110] that can be running standalone or a specific interface to be coupled to a host processor like e.g., as MCDRAM in Intel's Xeon Phi Knights Landing architecture, which uses 2.5D integration of the processor with several HMCs [127]. Also, the Intel Stratix 10 MX and Arria 10 FPGAs can be connected to external HMCs [28, 29].

Besides improved DRAM systems, novel non-volatile memory technologies like ferroelectric memory (FeRAM) [75] or magnetic memories (MRAM) [92] became an uprising research domain. They promise comparable access latencies, improved efficiency, and reduced power consumption compared to SRAM and DRAM systems [38]. However, as scaling and stacking these technologies is more challenging than scaling DRAM structures, they are so far used in systems that require only small storage capacities [38, 44].

4.2 In-/Near-Memory Computing

These memory technology innovations, especially the HBM and HMC, are key-enablers and led to the breakthrough of in- and near-memory computing.

4.2.1 Evolution. The idea of *in-memory* computing goes back to the '60s. Already with the invention of the DRAM cell by Dennard in 1966 [34], the first *logic-in-memory* (i.e., processing *in-memory*, PIM) approaches were proposed. Stone et al. described special-purpose, cellular logic-in-memory (CLIM) [94] and a general-purpose logic-in-memory processor only a couple of years later [131]. The first big hype happened in the '90s when Elliot et al. developed computational RAM [43], Kogge proposed EXECUBE [79] and discussed the potential of PIM architectures [80]. At its first research peak in 1996, Patterson et al. presented intelligent RAM (IRAM) [106].

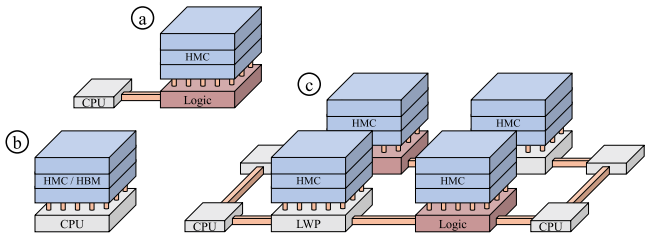

Figure 5: Memory-centric architecture variants employing the HMC or HBM. ⓐ Host-CPU with logic-enhanced HMC. ⓑ HBM or HMC used as stacked high bandwidth memory. ⓒ Heterogeneous architecture incorporating CPUs and HMCs enhanced with logic or lightweight cores (LWPs).

However, the advent of GPUs with high memory bandwidth [85], combined with the technical limitations of 2D integrated PIM, hindered the breakthrough of processing-in-memory. The main reasons were that it is tough to embed performant logic into DRAM technology, which is optimized for memory density, and that a vision for ease of programmability was missing at the time [106, 120].

The situation changed with seminal innovations in DRAM memory technology, as expounded in Section 4.1. The invention of through-silicon vias (TSVs) in 2009 [102] enabled Micron to develop the Hybrid Memory Cube (HMC) in 2011. It separates the logic from several memory layers stacked vertically on top of each other and connected by TSVs [110]. Although such an architecture is commonly referred to as 3D-stacked processing-in-memory (PIM), it is actually closer to *near-memory processing* since the logic is not integrated into the memory arrays themselves [125], which was the big burden for 2D integrated PIM in the '90s.

4.2.2 Architectures. Several approaches and architectures have been proposed, varying in their design, the intended purpose, and the type of integrated processing nodes. 3D-stacked memories like the HMC offer the opportunity to tightly couple processing elements and memory. Processing elements integrated into the logic layer of the HMC encompass a range from fine-granular operations (e.g., atomic operations) to coarse-grain accelerators of bigger functional blocks [66, 110, 125].

As thermal constraints are prevalent in 3D-stacked devices [74], often more energy-efficient processor architectures or lightweight processors (LWP) are employed [99, 114]. Despite the high flexibility of programmable cores, Asghari-Moghaddam et al. showed that these solutions reduce the total energy consumption by 64-68% for specific benchmark applications [99]. In 2016, Yitbarek et al. [143] simulated different architectural solutions based on the HMC. Integrating 16 Intel Atom cores into the logic layer facilitated a speedup of around 3.8x compared to a state-of-the-art quad-core processor connected to the HMC externally. However, a much higher speedup, up to 13x, was possible with dedicated hardware accelerators.

Also, GPUs are reasonable options for in- and near-memory computing as they often perform highly data-intensive tasks. Zhang et al. showed that viable in-memory GPU-solutions could have an advantageous energy efficiency of 76% compared to mainstream GPU systems [146]. Facing limitations of 3D-stacked combinations of memory and GPUs, Pattnaik et al. [109] alternatively use 2.5D integration techniques. This relaxes both the thermal constraint

within 3D-stacked devices as well as the limited memory capacity stacked in the processor's area, by combining memory and GPU horizontally on one silicon interposer. Nevertheless, they still achieved 4x lower memory latencies compared to ordinary GPUs.

Besides CPUs and GPUs, also reconfigurable logic like FPGA systems can be stacked on memory dies, as simulated by Gao et al. [52]. This approach can be seen as a compromise between performance, efficiency, and flexibility. Farmahini-Farahani et al. combined coarse-grained reconfigurable arrays with DRAM modules. This implies only minimal changes to the architecture of the DRAM [49]. Also, for prototyping purposes, FPGAs are a feasible solution to provide near-memory computing solutions. Jun et al. [71] implemented both the memory controller and the near-memory processor on an FPGA platform. As a cheaper alternative to memory clouds that combine tens or hundreds of accommodated DRAM modules, they developed a 20-node flash-based system that handles data sets of up to 20 TB efficiently.

4.3 Roofline Model for Near-Memory Units

The introduction of near-memory computing has significantly impacted the system's behavior in several aspects. In Figure 6, we analyze the implications of near-memory computing by extending the standard Roofline model. We differentiate between programmable cores and dedicated hardware accelerators. Both implementations profit from higher memory bandwidth, thus shifting the diagonal roof to the left in step ①. As we assume the operational intensity to remain constant, higher memory bandwidth results in an increase of effective compute performance, thus shifting the operating point in step ② vertically (cf. Section 3.3). Furthermore, near-memory units more frequently access the memory as they typically have smaller caches or scratchpads. Step ③ depicts the corresponding decrease of operational intensity, resulting in a left shift of the operating point. Compared to programmable cores, dedicated near-memory hardware accelerators generally enhance both the nominal and effective compute performance. Thus, the horizontal roof and the operating point shift upwards, as indicated in step ④.

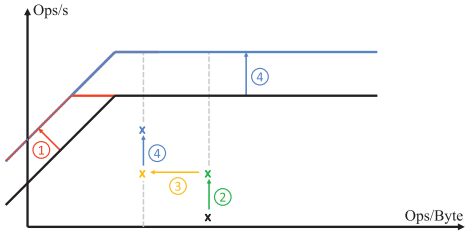

Figure 6: Combined Roofline model for near-memory cores (red) and accelerators (blue) with individual roofs.

4.4 Memory-Centric Programming

Despite plenty of architectural solutions for memory-centric computing systems, the question remains how these architectures can be efficiently programmed to exploit the nominal compute performance and especially the available memory bandwidth.

4.4.1 Programming Paradigm. The wide range of architectures employing the HMC is represented in Figure 5. Many solutions leveraging the HMC are standalone custom-made designs (i.e., logic-enhanced memories) that process application-specific workloads (cf. Figure 5a). Often API calls are used to communicate with the logic layer of the HMC [126]. This puts a tremendous burden on the developer and programmer.

A second widespread approach is to use the HBM or HMC as high bandwidth memory in a so-to-say compute-centric manner, i.e., as better memory technology, what is conceptionally shown in Figure 5b. The logic layer of the HMC is then used as a compatible memory controller without custom near-memory processing logic, e.g., as MCDRAM in the Intel Knights Landing architecture, or connected to the Intel Stratix 10 MX FPGA [29, 127]. In these cases, no changes to the programming model are required.

A third alternative is to integrate in- or near-memory processing units as co-processors into existing programming models in order to enable seamless cooperation with the host processor (cf. Figure 5c). This leads to heterogeneous architectures and requires an appropriate programming paradigm to be able to exploit the memory-centric architecture fully. Like CUDA and OpenGL are used for GPUs, OpenCL, AMDs HSA, or even OpenMP are viable candidates for near-memory computing. However, no dedicated programming model for memory-centric architectures is present to date [125].

One could envision a dynamic, system- and utilization-aware solution to distribute tasks between suitable processing nodes that burdens the programmers as little as possible. Hsieh et al. proposed a transparent offloading and mapping mechanism for near-memory GPU systems that identify candidates of near-memory offloading at compile-time and decides then dynamically which of these code blocks are actually offloaded [63]. Another attempt is to offload library or operating system routines such as inter-process communication to near-memory units. This would profit a wide range of applications, and the programmer is not burdened with scheduling near-memory operations and code changes of the application are avoided [116]. Faraboschi et al. went one step further and addressed the necessity to rethink traditional operating systems entirely as they do not target the characteristics of complex memory-centric architectures. They proposed a vision of a memory-centric OS, which relieves traditional compute nodes from typical OS routines, such as memory allocation or synchronization, and shifts them closer to the memory instead [48].

Thus, in- and near-memory approaches are utilized to accelerate application-specific tasks and also operating system routines (e.g., for inter-process communication or synchronization). While the former are very specific and lead to a wide variety of architectures, the latter, since being more or less opaque to the programmer, are prone to profit a wide range of applications and systems.

4.4.2 Cache Coherence. Besides the problems of architecture exploitation and task scheduling mentioned above, also cache coherence and memory consistency between host processors and in- or near-memory units have to be addressed. This is challenging as the latter are typically positioned behind the last level cache. Thus, all operations performed by near-memory processing units bypass the cache hierarchy of the host processor. There exist several approaches with different implications on system performance and ease of use.

In- or near-memory units can act like coherent processors and issue all writeback, lookup, and invalidation commands automatically

with hardware support in a fine-grained manner. While program code changes can be kept to a minimum, this would potentially result in a considerable amount of coherence traffic in the system before, during, and after the near-memory unit is operating. Besides the energy perspective, this solution also requires a significant amount of hardware development [66]. In case the system provides no hardware coherence between processors and near-memory units, this has to be even managed explicitly by the programmer. What works fine for simple data sets, becomes complicated for irregular data structures, e.g., highly pointer-based data structures. This can result in a vast number of memory loads and cache evictions [66].

Other approaches restrict near-memory units to operate on non-cacheable data, which is reasonable if near-memory units use data almost exclusively [49, 146].

Furthermore, Boroumand et al. presented their coarse-grained LazyPIM approach in 2016 [17]. This alternative to fine-grained coherency messages speculatively performs near-memory operations. After the execution is finished, one batched coherence message is sent to the corresponding processor. In case conflicts occurred during the speculative execution, the near-memory unit rolls back its execution, updates all conflicting cache lines, and starts its execution again from the beginning. This avoids coherence messages during concurrent execution and reduces the overall amount of coherence traffic significantly. This approach has similarities to the idea of transactional memory.

4.5 Discussion of Memory-Centric Systems

Near-memory processing units, both accelerators or programmable cores, have huge advantages in terms of processing efficiency and power consumption. A major difference between various designs is the way they are integrated into the system architecture – physically inside the memory or close to the memory. Figure 7 compares in- and near-memory processing architectures with respect to the key system characteristics defined in Section 3.5. The compute performance, interconnect latencies and power consumption are an order of magnitude better in case of in-memory solutions, due to much lower access latencies between processing unit(s) and memory [79]. However, in-memory computing designs cannot be integrated as easy as near-memory processing elements in various architectures, which limits their flexibility. In terms of programmability, both do not differ significantly – either implementation requires much programming effort to offload memory-intensive kernels sufficiently, to handle the cache coherence, and establish the communication between the host CPU and in-/near-memory processing units.

In conclusion, memory-centric systems offer the following chances and challenges:

Chance MC+1: Memory-centric designs help to mitigate the locality wall. In- and near-memory computing tackles the von Neumann bottleneck as the overall data movement between memory and processor is reduced. Performing computations close to memory leads to an increased data-to-task locality and better utilization of the available memory bandwidth. Furthermore, the dependency on evermore complex cache hierarchies diminishes.

Chance MC+2: Performance and power efficiency are greatly improved. As data movement and off-chip accesses, being a significant contributor to energy consumption and communication latency, are reduced, the system's overall efficiency increases [106]. Additionally, the processors and the cache hierarchy are relieved by more efficient accelerators.

Chance MC+3: The application field of NMC is steadily increasing. The wide range of prototypes and commercial systems leveraging this architectural approach reveals its potential [126]. Inventions like the HMC and HBM can be either used standalone as logic-enhanced memory, as a heterogeneous system in conjunction with a host processor or seamlessly integrated as standard memory, i.e., as a high bandwidth memory technology.

Challenge MC-1: Not all technological limitations have been overcome. Especially 3D-stacked memories suffer from thermal issues and constraints as cooling capabilities are restricted inside the stack. Furthermore, it is still challenging to fabricate complex in-memory logic due to the conflicting technological optimizations of logic and memory cells. Besides, the resulting complexity of stacked memories implies further testing and yield challenges [70, 136].

Challenge MC-2: A programming paradigm specific to memory-centric systems is missing. The fact that the HBM is often used in a compute-centric manner as a fast and high bandwidth memory technology is a good indication that a real vision for memory-centric designs is still lacking. Many present systems are hand-crafted to reap the benefits of near-memory computing. Providing a standard and user-friendly programming model is still a very challenging topic. However, this is required to make a memory-centric system suitable for everyday use and to ease architecture exploitation. Major obstacles are: 1) the more complicated memory view of a heterogeneous system, 2) providing cache coherence and memory consistency between the host processors and the near-memory units, 3) integrating an MMU and virtual memory, and 4) the missing support of compilers, libraries, and programming languages.

Challenge MC-3: Multiple memories diminish the "near-memory" aspect. Compute-centric architectures avoid thermal and access hot-spots by physically distributing processing and memory nodes. Similar approaches to address the tremendous pressure on the main memory system have been successfully applied to memory-centric systems, e.g., an inter-memory network of HMCs with a 75.1% average memory access latency reduction and 22.1% total memory energy saving [145]. However, as many benefits of in- and near-memory computing depend on the locality inside a

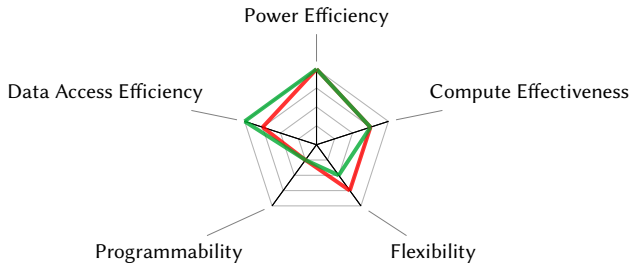

Figure 7: Comparison of near-memory computing (red) and in-memory computing (green)

memory, this may lead to an inter-memory bottleneck when several memories have to be accessed. Handling multiple memories is further impeded as most approaches do not use an MMU, virtual memory, or advanced memory scrambling technologies to extract the best off-chip bandwidth [125]. A vision for near-memory computing on distributed (shared) memory architectures is necessary.

Challenge MC-4: No general base NMC architecture is defined. The memory-centric community presents many custom and specialized designs for a particular use-case. This makes the standardization and ease of use more difficult as compared to e.g., multi-core architectures, GPU-systems, or mobile devices (cf. Section 5). Already a clear vision on what kind of operations are feasible for in- and near-memory computing [66] and how they will be integrated into a programming paradigm (via the API, libraries, or OS routines) would significantly improve the usability of the highly promising memory-centric research.

5 APPLICATION-CENTRIC SYSTEMS

While compute- and memory-centric approaches try to mitigate or altogether avoid the von Neumann bottleneck between data and tasks, respectively, application-centric designs follow a different path. Often driven by strict thermal and power constraints or high performance requirements, application-centric architectures consist of or make use of *application-specific* co-processors or accelerators. Prime examples of such heterogeneous systems are CPU-GPU, CPU-FPGA, or big.LITTLE architectures, mobile devices with integrated neural processors (NPU), or in general, the use of application-specific integrated circuits (ASIC). They are used when a general-purpose processor would not be performant and power-efficient enough. Digital signal processing, neural processing, and network processing are only a few examples.

As application-centric architectures are tailored to a specific task or application domain, many optimizations are possible that cannot be performed for a general-purpose system. Nevertheless, some of these specialized co-processors, like GPUs, have already become an essential part of today's compute-centric general-purpose systems. They are a promising role model for formerly specialized hardware that became mainstream.

In the following subsections, we provide and analyze the evolution of several application-centric systems, including their architecture and programming models. Due to the lack of research in this area [115], as well as the rapid evolution of mobile AI, we mainly focus on the mobile device architectures as prime examples of application-centric systems. Finally, we carve out the advantages and weaknesses of application-centric architectures.

5.1 Evolution of Application-Centric Systems

Throughout its history, the evolution of mobile devices has not solely been driven by technological innovations and advancements, but also by steadily growing user demands. The first portable cell phone was, in the most profound sense of the word, a truly application-specific architecture. It provided the means for communication by receiving and sending signals over radiofrequency. Since then, and especially in the past decade, the architecture of mobile devices has undergone a tremendous evolution. With each further generation, it is becoming more energy-efficient, compute-performant, and is acquiring numerous new features and capabilities. Mobile architectures have adopted the most crucial innovations from compute-centric architectures and further optimize them for operation under tight thermal, power, and energy constraints. Furthermore, the pace at which the mobile processors evolve is much higher than for other processors [8, 58].

In a study, Reddi et al. have identified two significant driving forces behind the ubiquity of mobile devices – Arm processors and the Android operating system (OS) [115]. From the hardware perspective, Arm itself does not produce the processors. Instead, they design and license CPU core's and other IP's designs. Thus, it is possible for vendors to radically reduce the development effort by merely customizing the existing designs for their purposes and application domain.

This trend continues today as Texas Instruments and Intel have ceased their mobile processor's development for smartphone and tablet markets in 2012 and 2016, respectively. The resurgence of Intel's Atom processors for mobile phones, however, has not been crowned with success – only a few smartphone models are equipped with new Atom processors. Similarly, Qualcomm and Samsung, having the prevalent market share, have also partially canceled their custom CPU development towards customization of *stock* Arm processors, straightening the retrieved resources to special-purpose cores. This tendency also applies to other processing units like GPUs.

As a result, processor customization has enabled the mobile system-on-chip designs to mitigate further the limitations coupled with the end of Dennard scaling and Moore's law [115]. Besides the micro-architectural improvements of mobile CPUs, extensively discussed in [58], the introduced Arm big.LITTLE design has provided the efficient means to deal with a set of power constraints and high compute performance requirements simultaneously [15, 55]. Arm's DynamIQ technology, the successor of big.LITTLE, extends the existing concept – it combines available *big*, high-performance, and *little*, power-efficient cores to build heterogeneous clusters of a much wider range of possible configurations, thus enhancing flexibility and performance scalability of a system [41].

Another application-centric domain – gaming consoles – undergoes a similar development. For instance, the more complex IBM Cell Broadband Engine [72] processor in the PlayStation 3 has been replaced in its successor (PlayStation 4) by a semi-custom AMD Accelerated Processing Unit (APU), representing an on-die CPU-GPU application-specific heterogeneous system. The unveiled technical specification of the next-generation PlayStation 5 even further proves the trend towards customization of already existing solutions, being built upon a discrete CPU-GPU system with an emphasis on gaming and multimedia processing [128].

A further representative of application-centric architectures, wearables, has followed an almost identical evolutionary path. To meet ever-growing application requirements while improving the end-user satisfaction at the same time, wearables have evolved from solely collecting and preprocessing the sensor data to compute-performant, energy-efficient, smartphone-independent devices by customizing and optimizing processing units of mobile SoCs [134]. Nonetheless, the arising IoT application demands cannot be met by the existing wearable architectures. However, increasing the

core count by utilizing traditional many-core architectures is restricted by tight power constraints, and the development of ASIC accelerators implies high nonrecurring engineering (NRE) costs. Based on these facts, a shift towards many-core architectures as the next evolutionary step of compute-centric architectures has been adopted for the application-centric domain in some academic projects, such as LOCUS and Stitch [133, 134].

All of these developments are enablers of another leading trend in the past decade: machine learning. No type of processing unit has been left untouched: general-purpose CPUs, GPUs, FPGAs, and even custom-designed ASICs are all workhorses in the deep learning era [65, 132]. However, the tremendous development of mobile AI accelerators is matchless, especially when considering the tight power and thermal constraints [65, 147]. With the development having started roughly 20 years ago, the first rudimentary and handmade models for smart data processing were run on single-core mobile CPUs and very limited RAM. However, the advent of smartphones and the introduction of mobile multi-core processors and mobile GPUs required and enabled the rapid development of mobile deep learning. The increased computational performance and the improvements of deep learning techniques have further changed the design of mobile devices. Nowadays, every high-end smartphone is equipped with several neural processing units (NPU) alongside multi-core CPUs, GPUs, and DSPs.

5.2 Contemporary Mobile Device Architectures

Today's mobile SoCs are the result of previously discussed technological innovations, trends, and ever-growing amounts of data to be processed. Figure 8 shows an abstract view of a typical modern smartphone's architecture. It is based on the analysis of the newest flagship SoCs of leading semiconductor companies, such as Snapdragon 865 (Qualcomm) [98], Exynos 990 (Samsung) [1], Kirin 990 5G (HiSilicon, Huawei) [76, 122] and Dimensity 1000 (MediaTek) [123] – all of them supporting 5G. Primary processing units are illustrated at the top of the block diagram. They are connected to memory, I/Os, and diverse hardware accelerators, which are essential for the mobile domain, through an interconnect hierarchy (separate interconnect fabrics are not shown in Figure 8).

Mobile CPUs. CPUs have always served the purpose of system orchestrator and general-purpose computing. As previously mentioned, microprocessors exploit asymmetric processing in the vast majority of modern mobile SoCs. Despite Arm's big.LITTLE heterogeneous design being omnipresent, there also exist other CPU design approaches.

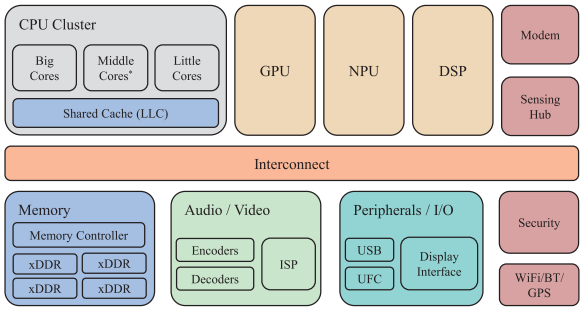

Figure 8: Mobile SoC of a contemporary high-end smartphone. Components are clustered by the type and application of a processing unit. Figure is not drawn to scale.

For instance, after analyzing the energy consumption of different scenarios, engineers of MediaTek have concluded that medium load tasks are of particular importance across all scenarios: their execution on *big* cores is energy-wasting, whereas *little* cores cannot meet the performance requirements, thus harming the user satisfaction. To address this, the "Tri-Gear" or "Tri-Cluster" concept has been introduced, splitting available cores into *three* clusters, each consisting of dedicated cores for tasks of a particular load [88].

Each flagship SoC discussed above incorporates CPUs built upon either the DynamIQ or Tri-Gear-like *technology*. Irrespective of configuration, the reviewed SoCs, except for the Exynos 990, integrate customized premium Arm Cortex-A7 series cores running at different frequencies to deliver maximum performance, whereas Cortex-A55 cores provide energy-efficiency in each of the considered SoCs. Exynos 990 is the last mobile SoC of Samsung, which still uses full-custom cores (Exynos M5) [5].

However, not only the technology scaling and increase of operating frequency of CPUs contribute to an enhancement of the general-purpose compute performance of the system. Also, the cache hierarchy, DRAM technology, as well as GPU performance, play a very crucial role. Nowadays, due to the emergence of novel processing units (e.g., NPUs), CPU cores occupy less than 25% of the entire die area, as it has been since 2015 [60, 124].

Mobile GPUs. The well-established CPU-GPU computing paradigm has also been successfully applied to the mobile domain. It is adapted and optimized for particular use-cases such as graphics, gaming, and in most recent models for the acceleration of machine learning. Considering the aforementioned mobile SoCs, most of them integrate customized Arm Mali-G76/77 GPUs, thus continuing and adopting the trend of mobile CPUs towards customization of existing IPs. Only the Snapdragon SoC incorporates the own product of Qualcomm – Adreno 650 GPU, which is moreover an essential part of the AI Engine along with the Kryo 585 (CPU), Hexagon 698 Processor (DSP) and Sensing Hub [98].

Mobile AI engines (NPUs). Also, AI processors have recently become a substantial part of modern mid-range and high-end smartphones. In contrast to state-of-the-art CPUs and GPUs, the implementation of AI processing units in the chipsets of each vendor is completely different. For instance, in the latest generation of the HiSilicon Kirin 990 5G, the previous NPU of Cambricon has been replaced by Huawei's internally developed Da Vinci architecture. According to Huawei, the Da Vinci NPU in the Kirin 990 5G consists of two *big* and one *tiny* core providing both high compute-performance and energy efficiency when necessary [87]. APU 3.0, the AI processing unit of the Dimensity 1000, borrowed a Tri-Gear configuration – consisting of two big, three small, and a single tiny core to better match the smartphone's AI needs.

Moreover, as previously mentioned, AI acceleration can also be performed within multiple various processing units, involving CPU, GPU, DSP on top of dedicated AI accelerators (illustrated as NPU in Figure 8). The Snapdragon 865 and Exynos 990 are the prime examples of this approach, utilizing the modified-for-AI-acceleration DSP. It is worth mentioning that the performance of AI processing units in mobile SoCs of a previous generation was

comparable with those of desktop CUDA-enabled Nvidia GPUs introduced in the past years [65].

Memory/Cache/Interconnect/Peripherals. Other contributors, such as an increase of the size of LLC in conjunction with the integration of new low-power DDR5 (LPDDR5) into the newest mobile SoCs, further improve the performance and power efficiency by providing better bandwidth and reducing costly DRAM accesses [2]. Cache-coherent interconnects, mostly vendor-specific, provide full cache coherence not only between processor clusters but also for other processing engines and I/O devices.

Today's mobile SoCs feature extremely high *heterogeneity*. Besides the processing units described earlier, a typical SoC of modern smartphones also contains a sophisticated modem with multimode support and WiFi/Bluetooth connectivity, an advanced image signal processor (ISP) providing excellent capture and editing capabilities, audio and video encoders and decoders. At the same time, data collected by sensors and external I/O devices have to be handled, and security has to be provided as well, complementing a typical mobile SoC, as depicted in Figure 8.

5.3 Roofline Model for Hardware Accelerators

To comply with tight constraints, application-centric designs employ specialized execution units for different processing tasks. When offloading subroutines to specific fixed-function units, the application code is divided into two parts. One fraction is executed on the accelerator, whereas the other remains on the host processor. Concerning the specific algorithms that can be executed on the accelerator, the compute performance of the host processor is typically significantly lower compared to a fixed-function accelerator. Also, the memory bandwidth of the accelerator can potentially vary compared to the processor, i.e., in case of near-memory units. Thus, similarly to Hill et al. [60], we compare different components in one diagram using individual roofs.

Considering *near-memory* accelerators as a specific example, *memory-intensive* tasks are performed by the accelerator, and *compute-intensive* tasks remain on the host processor. Evidently, the partitioning of compute kernels can vary for different scenarios, e.g., other heterogeneous architectures than near-memory accelerators. Figure 9 shows a combined Roofline model for both the host processor and near-memory accelerator. Individual roofs for both units are depicted. The processor is bound by the blue roofs, whereas the red roofs illustrate the boundaries for the fixed-function accelerators. As it has to share the bandwidth with the accelerator, the host processor has a slightly decreased memory bandwidth. Step ① illustrates the resulting shift of the diagonal roof. Furthermore, as the host processor performs significantly less memory accesses, its operational intensity in *Ops/Byte* increases ($I_{CT} > I_0$, step ②). In contrast, the accelerator benefits highly from increased memory bandwidth, and a higher nominal (but task-specific) compute performance. Thus, both the horizontal and the diagonal roofs shift upwards, which is shown in steps ③ and ④. Compared to the processor, the accelerator performs primarily memory-intensive operations (i.e., lower *Ops/Byte*), thus resulting in a low operational intensity ($I_{MT} < I_0$). Step 5 illustrates the operating point of the accelerator.

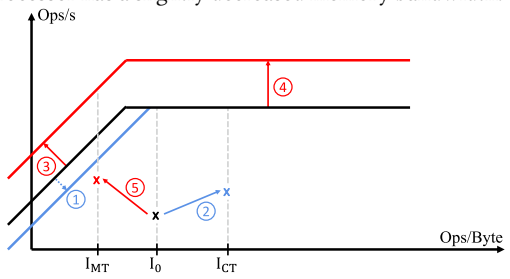

Figure 9: Roofline Model for heterogeneous architectures. Different threads are executed on the host processor and the near-memory accelerator as a particular example.

5.4 Application-Centric Programming Models

As previously discussed, the key advantage of application-centric architectures arises from their heterogeneity. Nonetheless, to achieve the improvements of potential orders of magnitude in performance and power efficiency, different types of applications (or different phases) have to be addressed to the best-suited processing units [84]. Moreover, strict task mapping to specific processing units and the resulting resource contention during simultaneous execution of parallel threads lead to another challenge – underutilization of idle resources [62].

In contrast to memory-centric architectures, there already exist well-established programming models and languages for each type of computation and processing unit. For instance, mobile GPUs can be utilized both for compute and graphics workloads. For the primary purpose of GPUs, graphics rendering applications, the industry's leading standard is OpenGL ES supporting 2D and 3D graphics [53]. In the case of computing workloads, OpenCL is widely used to enable general-purpose computing on GPUs [56].

The latest release of Vulkan, introduced by Khronos Group, is pursuing the idea of combining these two compute paradigms, providing a unified cross-platform API for graphics and compute acceleration [57]. The unveiled specifications of the upcoming game consoles, PlayStation 5 and Xbox Series X, point out to a custom AMD GPU built on the next generation of RDNA architecture. Besides the support of traditional graphic rendering languages, e.g., OpenGL, DirectX, or Vulkan, the new gaming consoles will be provided with the Unreal Engine, a software framework to ease the programmers' effort in creating games [3].

Furthermore, prior to the introduction of Android Neural Network API (NNAPI), GPUs have performed AI acceleration by using vendor-specific development tools. To cope with the programmer's effort and insufficient performance, NNAPI has been developed – it enables the hardware resources capable of accelerating deep learning models to be directly accessed through the Android OS [6]. Since then, each vendor has provided software development kits (SDKs), suitable for a particular SoC, to further improve and optimize the performance of trained neural networks by bridging different frameworks of corresponding processing units.

5.5 Discussion of Application-Centric Systems

In summary, a wide range of products, such as mobile communication devices or wearables, but also gaming consoles, meanwhile employ application-centric systems. Due to their high specialization on the intended purpose, these architectures significantly differ

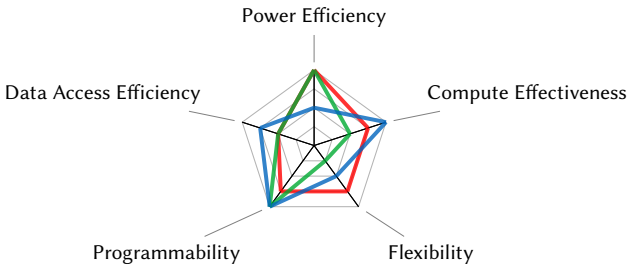

Figure 10: Comparison of different application-centric architectures: contemporary high-end mobile SoCs (red), wearables (green) and gaming consoles (blue).

in various aspects. Figure 10 provides a comparison of three representative application-centric architectures, discussed in previous subsections, to examine the influence of application constraints and requirements on the key system characteristics.

In comparison to gaming consoles and wearables, modern mobile devices exhibit higher flexibility by covering a wider spectrum of applications and use-cases. At the same time, modern mobile SoCs are significantly more power-efficient – building heterogeneous CPU clusters for particular workloads and utilizing a wide range of power-efficient application-specific accelerators. Based on the previous fact and the tight coupling between SoCs components, mobile devices exhibit high compute effectiveness as well – despite delivering an order of magnitude lower compute performance than gaming consoles [4, 128]. The synergy of an integrated non-volatile memory (SSD) and the latest generation of GDDR for RAM dramatically increase the data access efficiency in contemporary gaming consoles, which cannot be achieved in mobile SoCs due to the strict area and power limitations [128]. Finally, architecture-aware vendor-specific SDKs for all of the discussed application-centric architectures notably reduce the programming effort.

In conclusion, several chances and challenges of application-centric systems will be discussed.

Chance AC+1: Customization enables optimization. Unlike compute-centric systems, application-centric systems are primarily designed and customized to meet the requirements of a particular application domain. Mobile devices, gaming consoles and many other application-centric architectures are prime examples of a successful architecture customization, enabling optimizations that are unattainable for a general-purpose system [58, 65].

Chance AC+2: Heterogeneity is key. The availability of numerous processing units and dedicated hardware accelerators enables the application-specific architectures to operate under the tight power and area constraints efficiently. The introduced and well-established big.LITTLE design approach of CPUs in mobile SoCs can balance between performance and power efficiency, both being substantial for this domain. The appearance of novel processing elements like AI accelerators furthermore contributes to end-user satisfaction by improving the compute performance and providing additional capabilities.

Chance AC+3: An established ecosystem is present. The presence of an established ecosystem (e.g., for mobile devices) has multiple benefits for engineers as well: from the hardware perspective, the development effort can be drastically reduced by utilizing the Arm IPs for the primary processing elements, such as CPUs, GPUs, and even NPUs. From the software perspective, the existing open-source Android OS with comprehensive capabilities (e.g., NNAPI for the acceleration of neural networks) in conjunction with emerging programming models further ease the programmer's effort.

Chance AC+4: High architecture-awareness is provided. Beyond the existing ecosystem, each vendor provides architecture-aware SDKs for better exploitation of the system's capabilities while maintaining a user-friendly programming interface.

Challenge AC-1: An holistic SoC-level approach is needed. However, the characterization of the entire system performance of application-centric architectures is highly required yet still a very challenging task [147]. The performance of an application-centric architecture cannot be characterized by exclusively investigating processing units or applications executed on the system – it is highly affected by multiple IPs running in parallel [60].

Challenge AC-2: Tight performance and power constraints complicate the architectural design. Designers of application-centric architectures face a two-sided challenge – to deliver high performance while operating within strict thermal and energy constraints. The end of Dennard scaling has further severely impacted the existing power density issue [115]. Moreover, the battery density improvements cannot stand the pace of Moore's law implications [121].

Challenge AC-3: Underutilization of processing units is increasing with growing heterogeneity. Despite the heterogeneity of application-centric systems, a strict mapping policy might become a bottleneck. Mapping particular tasks to specific processing units can lead to resource contention, or, in contrast, their underutilization [62]. Thus, implementing a proper task partitioning policy poses a new challenge.

Challenge AC-4: Diversity of processing elements negatively impacts system programmability. Programmers of heterogeneous application-specific systems often have to deal with programmability issues. For instance, a cache coherence and memory consistency have to be explicitly handled in a system with multiple processing units of different types and programming standards. Moreover, the lack of programming standards for custom system configurations exacerbates the issue.

6 CONCLUSION AND INSIGHTS

So far, we have surveyed three different architectural design paradigms, including their strengths and weaknesses. However, the prime question of whether an architectural convergence of the presented domains or a further co-existence of distinct architectural design approaches will be presumed in the near future, needs to be answered.

Figure 11 illustrates and compares the key system characteristics of the individual domains. State-of-the-art compute-centric architectures satisfy user demands in many use-cases. As legacy and commoditized programming models can be applied without significant modifications to nearly all architectures, they provide a good user/programmer friendliness and flexibility. However, further performance advances by solely increasing the number of processing cores are hindered by scalability issues. Physically distributing memory and incorporating scalable interconnects meanwhile lead

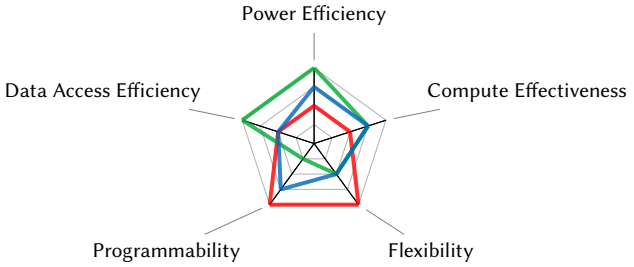

Figure 11: Comparison of different design techniques: Compute-Centric Architecture (red), Memory-Centric Architecture (green), Application-Centric Architecture (blue)

to new challenges such as data-to-task dis-localities and increased data movement. Due to the arising distributed memory hierarchies and the resulting spread of data structures, applying near-memory computing to the compute-centric domains is challenging.

Nevertheless, the problem of increasing amounts of data traffic between processor and memory is addressed by memory-centric designs. Instantiating of in- and near-memory processing units implies both the reduction of data traffic and utilization of higher memory bandwidth, thus contributing to a better power-efficiency and increased compute performance, respectively. Despite the high heterogeneity, application-centric designs are bound by tight thermal, power, and area constraints. Thus, they would highly benefit from 3D-stacking technology and near-memory computing, combined with the benefits of heterogeneity in a synergistic manner.

In addition to the ultimate diversity of specialized cores and hardware accelerators, some application-centric architectures feature an established ecosystem, like the presented mobile domain. In conjunction with the customization of existing IPs, this significantly reduces the effort of both hardware and software development. Moreover, the provided architecture-aware programming models enable the efficient exploitation of system resources, which is still missing in legacy programming models of compute-centric architectures. We believe that also the memory-centric domain would benefit from such an ecosystem, including a standardized base architecture as well as a unified powerful programming model.

Given the wide diversity in domain-specific requirements and constraints, this survey yields the insight that a one-size-fits-all architecture approach seems not in sight. However, we conclude that performant, programmable, and future-proof computer architectures would highly profit from established ecosystems, 3D-stacked logic-enhanced memory devices, as well as commoditized architecture-aware programming models.

ACKNOWLEDGEMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 146371743 – TRR 89: Invasive Computing. We thank Lars Nolte and Tim Twardzik from TUM for their valuable comments.

REFERENCES

[1] Samsung Exynos 990. 2019. https://www.samsung.com/semiconductor/global.semi.static/minisite/exynos/file/solution/MobileProcessor-990.pdf
[2] Shweta Aladakatti et al. 2019. Battery life optimization techniques for ultra-low power SOCs. *EAI Endorsed Transactions on Cloud Systems* 5, 16 (11 2019). https://doi.org/10.4108/eai.5-11-2019.162591
[3] AMD. 2019. Introducing RDNA architecture. https://www.amd.com/system/files/documents/rdna-whitepaper.pdf
[4] AnandTech. 2019. The Snapdragon 865 Performance Preview: Setting the Stage for Flagship Android 2020. https://www.anandtech.com/show/15178/qualcomm-announces-snapdragon-865-and-765-5g-for-all-in-2020-all-the-details/2
[5] AnandTech. 2020. The Exynos 990 SoC: Last of Custom CPUs. https://www.anandtech.com/show/15603/the-samsung-galaxy-s20-s20-ultra-exynos-snapdragon-review-megalomania-devices/4
[6] Android Neural Networks API. 2017. https://developer.android.com/ndk/guides/neuralnetworks
[7] Intel Many Integrated Core Architecture. 2012. https://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture/
[8] Arm. 2018. Accelerating mobile and laptop performance: Arm announces Client CPU roadmap. https://www.arm.com/company/news/2018/08/accelerating-mobile-and-laptop-performance
[9] Oliver Arnold et al. 2010. Power aware heterogeneous MPSoC with dynamic task scheduling and increased data locality for multiple applications. In *Proceedings of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2010), Samos, Greece, July 19-22, 2010*, Fadi J. Kurdahi and Jarmo Takala (Eds.). IEEE, 110–117. https://doi.org/10.1109/ICSAMOS.2010.5642075
[10] John Backus. 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Commun. ACM* 21, 8 (1978), 613–641.
[11] Evgenij Belikov et al. 2013. A Survey of High-Level Parallel Programming Models.
[12] Shane Bell et al. 2008. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *2008 IEEE International Solid-State Circuits Conference, ISSCC 2008, Digest of Technical Papers, San Francisco, CA, USA, February 3-7, 2008*. IEEE, San Francisco, CA, USA, 88–89. https://doi.org/10.1109/ISSCC.2008.4523070
[13] M. Berezecki et al. 2011. Many-Core Key-Value Store. In *Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC '11)*. IEEE Computer Society, USA, 1–8. https://doi.org/10.1109/IGCC.2011.6008565
[14] Keren Bergman et al. 2008. Exascale computing study: Technology challenges in achieving exascale systems. (2008).
[15] Arm big.LITTLE. 2011. https://www.arm.com/why-arm/technologies/big-little
[16] G. Blake et al. 2009. A survey of multicore processors. *IEEE Signal Processing Magazine* 26, 6 (2009), 26–37.
[17] Amirali Boroumand et al. 2017. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE Comput. Archit. Lett.* 16, 1 (2017), 46–50. https://doi.org/10.1109/LCA.2016.2577557
[18] A. Branover et al. 2012. AMD Fusion APU: Llano. *IEEE Micro* 32, 2 (Mar 2012), 28–37. https://doi.org/10.1109/MM.2012.2
[19] Doug Burger et al. 1996. Memory Bandwidth Limitations of Future Microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture, Philadelphia, PA, USA, May 22-24, 1996*, Jean-Loup Baer (Ed.). ACM, 78–89. https://doi.org/10.1145/232973.232983
[20] Thomas Burger. 2005. Intel Multi-Core Processors: Quick Reference Guide. https://software.intel.com/en-us/articles/intel-multi-core-processors-quick-reference-guide
[21] Lei Chai et al. 2007. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), 14-17 May 2007, Rio de Janeiro, Brazil*. IEEE Computer Society, 471–478. https://doi.org/10.1109/CCGRID.2007.119
[22] T. Chen et al. 2007. Cell Broadband Engine Architecture and its first implementation—A performance view. *IBM Journal of Research and Development* 51, 5 (Sep 2007), 559–572. https://doi.org/10.1147/rd.515.0559
[23] Nagabhushan Chitlur et al. 2012. QuickIA: Exploring heterogeneous architectures on real prototypes. In *18th IEEE International Symposium on High Performance Computer Architecture, HPCA 2012, New Orleans, LA, USA, 25-29 February, 2012*. IEEE Computer Society, 433–440. https://doi.org/10.1109/HPCA.2012.6169046
[24] Byn Choi et al. 2011. DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism. In *2011 International Conference on Parallel Architectures and Compilation Techniques, PACT 2011, Galveston, TX, USA, October 10-14, 2011*, Lawrence Rauchwerger and Vivek Sarkar (Eds.). IEEE Computer Society, 155–166. https://doi.org/10.1109/PACT.2011.21
[25] George Chrysos. 2014. *Intel® Xeon Phi™ Coprocessor — the Architecture.* Technical Report.
[26] Hybrid Memory Cube Consortium. 2014. *Hybrid Memory Cube Specification 2.1.* Technical Report.
[27] UPC Consortium. 2005. UPC Language Specifications V1.2. (5 2005). https://doi.org/10.2172/862127
[28] Intel Corporation. 2018. *Intel Arria 10 Device Overview.* Technical Report.
[29] Intel Corporation. 2020. *Intel Stratix 10 MX (DRAM System-in-Package) Device Overview.* Technical Report.
[30] L. Dagum et al. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5, 1 (Jan 1998), 46–55. https://doi.org/10.1109/99.660313
[31] William J. Dally. 2006. Computer Architecture in the Many-Core Era. In *24th International Conference on Computer Design (ICCD 2006), 1-4 October 2006, San Jose, CA, USA*. IEEE, 1. https://doi.org/10.1109/ICCD.2006.4380784
[32] Satish Damaraju et al. 2012. A 22nm IA multi-CPU and GPU System-on-Chip. In *2012 IEEE International Solid-State Circuits Conference, ISSCC 2012, San Francisco, CA, USA, February 19-23, 2012*. IEEE, 56–57. https://doi.org/10.1109/ISSCC.2012.6176876
[33] Mattias De Wael et al. 2015. Partitioned Global Address Space Languages. *ACM Comput. Surv.* 47, 4, Article 62 (May 2015), 27 pages. https://doi.org/10.1145/2716320
[34] Robert H Dennard. 1968. Field-effect transistor memory. US Patent 3,387,286.
[35] M. Deo et al. 2016. *Intel Stratix 10 mx devices with Samsung HBM2 solve the memory bandwidth challenge.* Technical Report.
[36] J. Diaz et al. 2012. A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era. *IEEE Transactions on Parallel and Distributed Systems* 23, 8 (2012), 1369–1386.
[37] Chapel Documentation. 2020. https://chapel-lang.org/docs/index.html
[38] Xiangyu Dong et al. 2008. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Proceedings of the 45th Design Automation Conference, DAC 2008, Anaheim, CA, USA, June 8-13, 2008*, Limor Fix (Ed.). ACM, 554–559. https://doi.org/10.1145/1391469.1391610
[39] Alejandro Duran et al. 2011. Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Process. Lett.* 21, 2 (2011), 173–193. https://doi.org/10.1142/S0129626411000151
[40] Yves Durand et al. 2014. EUROSERVER: Energy Efficient Node for European Micro-Servers. In *17th Euromicro Conference on Digital System Design, DSD 2014, Verona, Italy, August 27-29, 2014*. IEEE Computer Society, 206–213. https://doi.org/10.1109/DSD.2014.15
[41] Arm DynamIQ. 2017. https://www.arm.com/why-arm/technologies/dynamiq
[42] Ryusuke Egawa et al. 2013. Vertically integrated processor and memory module design for vector supercomputers. In *2013 IEEE International 3D Systems Integration Conference (3DIC)*. 1–6.
[43] Duncan G Elliott et al. 1992. Computational RAM: A memory-SIMD hybrid and its application to DSP. In *Custom Integrated Circuits Conference*, Vol. 30. 1–30.
[44] Tetsuo Endoh et al. 2016. An Overview of Nonvolatile Emerging Memories - Spintronics for Working Memories. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 6, 2 (2016), 109–119. https://doi.org/10.1109/JETCAS.2016.2547704
[45] Babak Falsafi et al. 2016. Near-Memory Data Services. *IEEE Micro* 36, 1 (2016), 6–13. https://doi.org/10.1109/MM.2016.9
[46] Intel Core Processor Family. 2020. https://www.intel.com/content/www/us/en/products/processors/core/
[47] POSIX 1003.1 FAQ. 2011. http://www.opengroup.org/austin/papers/posix_faq.html
[48] Paolo Faraboschi et al. 2015. Beyond Processor-centric Operating Systems. In *15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015*, George Candea (Ed.). USENIX Association. https://www.usenix.org/conference/hotos15/workshop-program/presentation/faraboschi
[49] Amin Farmahini Farahani et al. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. IEEE Computer Society, 283–295. https://doi.org/10.1109/HPCA.2015.7056040
[50] Wu-chun Feng et al. 2009. Tools and Environments for Multicore and Many-Core Architectures. *IEEE Computer* 42, 11 (2009), 26–27. https://doi.org/10.1109/MC.2009.412
[51] Intel FPGAs for Deep Learning. 2019. https://press3.mcs.anl.gov//atpesc/files/2019/08/ATPESC_2019_Track-1_8_7-29_330pm_Moawad_Nash-FPGAs.pdf
[52] Mingyu Gao et al. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*. IEEE Computer Society, 126–137. https://doi.org/10.1109/HPCA.2016.7446059
[53] Dan Ginsburg et al. 2014. *OpenGL ES 3.0 programming guide.* Addison-Wesley Professional.
[54] Gregory Ray Goslin. 1996. Guide to using field programmable gate arrays (FPGAs) for application-specific digital signal processing performance. In *High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic*, John Schewel, Peter M. Athanas, V. Michael Bove Jr., and John Watson (Eds.), Vol. 2914. International Society for Optics and Photonics, SPIE, 321 – 331. https://doi.org/10.1117/12.255830
[55] Peter Greenhalgh. 2011. *Big.LITTLE Processing with ARM CortexTM-A15 & Cortex-A7.* Technical Report.

[56] Khronos Group. 2009. OpenCL Overview. https://www.khronos.org/opencl
[57] Khronos Group. 2020. Vulkan Overview. https://www.khronos.org/vulkan/
[58] M. Halpern et al. 2016. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 64–76.
[59] J. Henkel et al. 2012. Invasive manycore architectures. In *17th Asia and South Pacific Design Automation Conference*. 193–200.
[60] Mark D. Hill et al. 2019. Gables: A Roofline Model for Mobile SoCs. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 317–330. https://doi.org/10.1109/HPCA.2019.00047
[61] J. Howard et al. 2010. A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*. 108–109. https://doi.org/10.1109/ISSCC.2010.5434077
[62] C. Hsieh et al. 2019. The Case for Exploiting Underutilized Resources in Heterogeneous Mobile Architectures. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1265–1268.
[63] Kevin Hsieh et al. 2016. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 204–216. https://doi.org/10.1109/ISCA.2016.27
[64] IBM IDC. 2017. The transformation of High Performance Computing: Simulation and Cognitive Methods in the Era of Big Data. https://www.slideshare.net/insideHPC/the-transformation-of-hpc-simulation-and-cognitive-methods-in-the-era-of-big-data
[65] Andrey Ignatov et al. 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. *CoRR* abs/1910.06663 (2019). arXiv:1910.06663 http://arxiv.org/abs/1910.06663
[66] Gabriel H. Loh Nuwan Jayasena Mark H. Oskin Mark Nutter Da Ignatowski. 2013. A Processing-in-Memory Taxonomy and a Case for Studying Fixed-function PIM.
[67] Advanced Micro Devices Inc. 2015. *High-Bandwidth Memory (HBM) Reinventing Memory Technology*. Technical Report.
[68] Intel. [n.d.]. *NVIDIA's Next Generation CUDA Compute Architecture*. Technical Report. https://www.nvidia.de/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
[69] Nuwan Jayasena. 2018. Memory-centric Accelerators in High-performance Systems. In *55th Design Automation Conference (DAC 2018), 24-28 June 2018, San Francisco, CA USA, Special Session on "Memory-centric Architectures: Industry Perspective from Embedded Systems to High Performance Computing"*.
[70] L. Jiang et al. 2010. Yield enhancement for 3D-stacked memory by redundancy sharing across dies. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 230–234.
[71] Sang Woo Jun et al. 2015. BlueDBM: an appliance for big data analytics. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 1–13. https://doi.org/10.1145/2749469.2750412
[72] J. A. Kahle et al. 2005. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development* 49, 4.5 (2005), 589–604.
[73] Henry Kasim et al. 2008. Survey on Parallel Programming Model. In *Network and Parallel Computing*, Jian Cao, Minglu Li, Min-You Wu, and Jinjun Chen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 266–275.
[74] Mushfique Junayed Khurshid et al. 2013. Data compression for thermal mitigation in the Hybrid Memory Cube. In *2013 IEEE 31st International Conference on Computer Design, ICCD 2013, Asheville, NC, USA, October 6-9, 2013*. IEEE Computer Society, 185–192. https://doi.org/10.1109/ICCD.2013.6657041
[75] W. I. Kinney et al. 1987. A non-volatile memory cell based on ferroelectric storage capacitors. In *1987 International Electron Devices Meeting*. 850–851.
[76] HiSilicon Kirin. 2019. http://www.hisilicon.com/en/Products/ProductList/Kirin
[77] D. Kirk et al. 2013. *Programming Massively Parallel Processors: A Hands-on Approach (Second Edition)*. Morgan Kaufmann.
[78] Peter Kogge. 2017. Memory Intensive Computing, the 3rdWall, and the Need for Innovation in Architecture. https://memsys.io/wp-content/uploads/2017/12/The_Wall.pdf
[79] Peter M Kogge. 1994. EXECUBE-a new architecture for scaleable MPPs. In *1994 International Conference on Parallel Processing Vol. 1*, Vol. 1. IEEE, 77–84.
[80] Peter M Kogge et al. 1997. Processing in memory: Chips to petaflops. In *Workshop on Mixing Logic and DRAM: Chips that Compute and Remember at ISCA*, Vol. 97. Citeseer.
[81] David A. Kranz et al. 1993. Integrating Message-Passing and Shared-Memory: Early Experience. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP), San Diego, California, USA, May 19-22, 1993*, Marina C. Chen and Robert Halstead (Eds.). ACM, 54–63. https://doi.org/10.1145/155332.155338
[82] Ronny Krashinsky et al. 2020. *NVIDIA Ampere Architecture In-Depth*. Technical Report.
[83] Rakesh Kumar et al. 2005. Heterogeneous Chip Multiprocessors. *Computer* 38, 11 (Nov. 2005), 32–38. https://doi.org/10.1109/MC.2005.379
[84] R. Kumar et al. 2004. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.* 64–75. https://doi.org/10.1109/ISCA.2004.1310764
[85] E Scott Larsen et al. 2001. Fast matrix multiplies using graphics hardware. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*. 55–55.
[86] Dong Uk Lee et al. 2014. 25.2 A 1.2 V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 432–433.
[87] Heng Liao et al. 2019. DaVinci: A Scalable Architecture for Neural Network Computing. In *2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, August18-20, 2019*. IEEE, 1–44. https://doi.org/10.1109/HOTCHIPS.2019.8875654
[88] Tsung-Yao Lin et al. 2016. Helio X20: The first tri-gear mobile SoC with CorePilot™ 3.0 technology. In *2016 IEEE Hot Chips 28 Symposium (HCS), Cupertino, CA, USA, August 21-23, 2016*. IEEE, 1–24. https://doi.org/10.1109/HOTCHIPS.2016.7936204
[89] Pejman Lotfi-Kamran et al. 2012. Scale-out Processors. In *Proceedings of the 39th Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '12)*. IEEE Computer Society, USA, 500–511.
[90] Luxin Yan et al. 2006. A DSP/FPGA - Based Parallel Architecture for Real-time Image Processing. In *2006 6th World Congress on Intelligent Control and Automation*, Vol. 2. 10022–10025.
[91] Milo M. K. Martin et al. 2012. Why on-chip cache coherence is here to stay. *Commun. ACM* 55, 7 (2012), 78–89. https://doi.org/10.1145/2209249.2209269
[92] K. Matsuyama et al. 1997. Low current magnetic-RAM memory operation with a high sensitive spin valve material. *IEEE Transactions on Magnetics* 33, 5 (1997), 3283–3285.
[93] T.G. Mattson et al. 2014. *Patterns for Parallel Programming*. Addison-Wesley Professional.
[94] RC Minnick et al. 1966. *CELLULAR ARRAYS FOR LOGIC AND STORAGE*. Technical Report. STANFORD RESEARCH INST MENLO PARK CALIF.
[95] Sparsh Mittal et al. 2019. A Survey on Evaluating and Optimizing Performance of Intel Xeon Phi. (May 2019).
[96] Sparsh Mittal et al. 2015. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.* 47, 4, Article 69 (July 2015), 35 pages. https://doi.org/10.1145/2788396
[97] Sparsh Mittal et al. 2015. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.* 47, 4 (2015), 69:1–69:35. https://doi.org/10.1145/2788396
[98] Qualcomm Snapdragon 865 5G mobile platform. 2019. https://www.qualcomm.com/media/documents/files/qualcomm-snapdragon-865-5g-mobile-platform-product-brief.pdf
[99] Hadi Asghari Moghaddam et al. 2016. Near-DRAM Acceleration with Single-ISA Heterogeneous Processing in Standard Memory Modules. *IEEE Micro* 36, 1 (2016), 24–34. https://doi.org/10.1109/MM.2016.8
[100] Manuel Mohr et al. 2017. Pegasus: Efficient data transfers for PGAS languages on non-cache-coherent manycores. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, David Atienza and Giorgio Di Natale (Eds.). IEEE, 1781–1786. https://doi.org/10.23919/DATE.2017.7927281
[101] Valentin Mena Morales et al. 2014. Energy-efficient FPGA implementation for binomial option pricing using OpenCL. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, Gerhard P. Fettweis and Wolfgang Nebel (Eds.). European Design and Automation Association, 1–6. https://doi.org/10.7873/DATE.2014.221
[102] M. Motoyoshi. 2009. Through-Silicon Via (TSV). *Proc. IEEE* 97, 1 (2009), 43–48.
[103] Robert W. Numrich et al. 1998. Co-Array Fortran for Parallel Programming. *SIGPLAN Fortran Forum* 17, 2 (Aug. 1998), 1–31. https://doi.org/10.1145/289918.289920
[104] P. Pacheco. 1996. *Parallel Programming with MPI*. Morgan Kaufman Publishers.
[105] J. Parkhurst et al. 2006. From Single Core to Multi-Core: Preparing for a new exponential. In *2006 IEEE/ACM International Conference on Computer Aided Design*. 67–72. https://doi.org/10.1109/ICCAD.2006.320067
[106] David Patterson et al. 1997. Intelligent RAM (IRAM): Chips that remember and compute. In *1997 IEEE International Solids-State Circuits Conference. Digest of Technical Papers*. IEEE, 224–225.
[107] David A. Patterson. 2004. Latency Lags Bandwidth. *Commun. ACM* 47, 10 (Oct. 2004), 71–75. https://doi.org/10.1145/1022594.1022596
[108] David A. Patterson et al. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (1997), 34–44. https://doi.org/10.1109/40.592312
[109] Ashutosh Pattnaik et al. 2016. Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*, Ayal Zaks andBilha Mendelson, Lawrence Rauchwerger, and Wen-mei W. Hwu (Eds.). ACM, 31–44. https://doi.org/10.1145/2967938.2967940
[110] J. T. Pawlowski. 2011. Hybrid memory cube (HMC). In *2011 IEEE Hot Chips 23 Symposium (HCS)*. 1–24.
[111] Russell J. Petersen et al. 1995. An assessment of the suitability of FPGA-based systems for use in digital signal processing. In *Field-Programmable Logic and Applications*, Will Moore and Wayne Luk (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 293–302.
[112] AMD Ryzen Threadripper 3990X Processor. 2020. https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3990x/
[113] AMD Ryzen Desktop Processors. 2020. https://www.amd.com/en/ryzen
[114] Seth H. Pugsley et al. 2014. NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*. IEEE Computer Society, 190–200. https://doi.org/10.1109/ISPASS.2014.6844483
[115] V. J. Reddi et al. 2018. Two Billion Devices and Counting. *IEEE Micro* 38, 1 (2018), 6–21.
[116] Sven Rheindt et al. 2019. NEMESYS: near-memory graph copy enhanced system-software. In *Proceedings of the International Symposium on Memory Systems, MEMSYS 2019, Washington, DC, USA, September 30 - October 03, 2019*. ACM, 3–18. https://doi.org/10.1145/3357526.3357545
[117] NVIDIA Titan RTX. 2018. https://www.nvidia.com/en-us/deep-learning-ai/products/titan-rtx/
[118] M. M. Sabry Aly et al. 2015. Energy-Efficient Abundant-Data Computing: The N3XT 1,000x. *Computer* 48, 12 (2015), 24–33.
[119] Vijay Saraswat et al. 2019. X10 Language Specification. http://x10.sourceforge.net/documentation/languagespec/x10-latest.pdf
[120] Ashley Saulsbury et al. 1996. Missing the Memory Wall: The Case for Processor/Memory Integration. *SIGARCH Comput. Archit. News* 24, 2 (May 1996), 90–101. https://doi.org/10.1145/232974.232984
[121] Fred Schlachter. 2013. No Moore's Law for batteries. *Proceedings of the National Academy of Sciences* 110, 14 (2013), 5273–5273. https://doi.org/10.1073/pnas.1302988110 arXiv:https://www.pnas.org/content/110/14/5273.full.pdf
[122] Huawei Kirin 990 Series. 2019. https://consumer.huawei.com/en/campaign/kirin-990-series/
[123] MediaTek Dimensity 1000 Series. 2019. https://www.mediatek.com/products/smartphones/dimensity-1000-series
[124] Yakun Sophia Shao et al. 2015. The Aladdin Approach to Accelerator Design and Modeling. *IEEE Micro* 35, 3 (2015), 58–70. https://doi.org/10.1109/MM.2015.50
[125] Patrick Siegl et al. 2016. Data-Centric Computing Frontiers: A Survey On Processing-In-Memory. In *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Alexandria, VA, USA, October 3-6, 2016*, Bruce Jacob (Ed.). ACM, 295–308. https://doi.org/10.1145/2989081.2989087
[126] Gagandeep Singh et al. 2019. Near-memory computing: Past, present, and future. *Microprocessors and Microsystems* 71 (2019), 102868.
[127] Avinash Sodani et al. 2016. Knights landing: Second-generation intel xeon phi product. *Ieee micro* 36, 2 (2016), 34–46.
[128] Sony. 2020. Unveiling New Details of PlayStation 5: Hardware Technical Specs. https://blog.playstation.com/2020/03/18/unveiling-new-details-of-playstation-5-hardware-technical-specs/?ref-cat=254013
[129] Euripides Sotiriades et al. 2007. A General Reconfigurable Architecture for the BLAST Algorithm. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology* 48, 3 (01 Sep 2007), 189–208. https://doi.org/10.1007/s11265-007-0069-2
[130] Akshay Srivatsa et al. 2017. Region based cache coherence for tiled MPSoCs. In *30th IEEE International System-on-Chip Conference, SOCC 2017, Munich, Germany, September 5-8, 2017*, Massimo Alioto, Hai Helen Li, Jürgen Becker, Ulf Schlichtmann, and Ramalingam Sridhar (Eds.). IEEE, 286–291. https://doi.org/10.1109/SOCC.2017.8226059
[131] Harold S Stone. 1970. A logic-in-memory computer. *IEEE Trans. Comput.* 100, 1 (1970), 73–78.
[132] Vivienne Sze et al. 2017. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference, CICC 2017, Austin, TX, USA, April 30 - May 3, 2017*. IEEE, 1–8. https://doi.org/10.1109/CICC.2017.7993626
[133] C. Tan et al. 2018. Stitch: Fusible Heterogeneous Accelerators Enmeshed with Many-Core Architecture for Wearables. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 575–587.
[134] Cheng Tan et al. 2017. LOCUS: Low-Power Customizable Many-Core Architecture for Wearables. *ACM Trans. Embed. Comput. Syst.* 17, 1, Article 16 (Nov. 2017), 26 pages. https://doi.org/10.1145/3122786
[135] The Economist. 2017. The world's most valuable resource is no longer oil, but data. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data
[136] E. I. Vatajelu et al. 2019. Challenges and Solutions in Emerging Memory Testing. *IEEE Transactions on Emerging Topics in Computing* 7, 3 (2019), 493–506.
[137] David Wentzlaff et al. 2007. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro* 27, 5 (2007), 15–31. https://doi.org/10.1109/MM.2007.89
[138] Samuel Williams et al. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785
[139] Martin S. Won. 2019. *Intel Agilex FPGAs Deliver a Game-Changing Combination of Flexibility and Agility for the Data-Centric World*. Technical Report.
[140] William A. Wulf et al. 1995. Hitting the memory wall: implications of the obvious. *SIGARCH Computer Architecture News* 23, 1 (1995), 20–24. https://doi.org/10.1145/216585.216588
[141] Yuan Xie. 2013. Future memory and interconnect technologies. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, Enrico Macii (Ed.). EDA Consortium San Jose, CA, USA / ACM DL, 964–969. https://doi.org/10.7873/DATE.2013.202
[142] Katherine A. Yelick et al. 1998. Titanium: A High-performance Java Dialect. *Concurrency - Practice and Experience* 10 (1998), 825–836.
[143] Salessawi Ferede Yitbarek et al. 2016. Exploring specialized near-memory processing for data intensive operations. In *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, Luca Fanucci and Jürgen Teich (Eds.). IEEE, 1449–1452. http://ieeexplore.ieee.org/document/7459537/
[144] Marcelo Yuffe et al. 2011. A fully integrated multi-CPU, GPU and memory controller 32nm processor. In *IEEE International Solid-State Circuits Conference, ISSCC 2011, Digest of Technical Papers, San Francisco, CA, USA, 20-24 February, 2011*. IEEE, 264–266. https://doi.org/10.1109/ISSCC.2011.5746311
[145] Jia Zhan et al. 2016. A unified memory network architecture for in-memory computing in commodity servers. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*. IEEE Computer Society, 29:1–29:14. https://doi.org/10.1109/MICRO.2016.7783732
[146] Dong Ping Zhang et al. 2014. TOP-PIM: throughput-oriented programmable processing in memory. In *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014*, Beth Plale, Matei Ripeanu, Franck Cappello, and Dongyan Xu (Eds.). ACM, 85–98. https://doi.org/10.1145/2600212.2600213
[147] Yuhao Zhu et al. 2018. Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective. *CoRR* abs/1801.06274 (2018). arXiv:1801.06274 http://arxiv.org/abs/1801.06274