Survey Paper

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2026, Vol. 0(0) 1–71

DOI: 10.1177/10943420251347461
journals.sagepub.com/home/hpc
Sage

Technology trends in computing hardware and their impacts on high-performance scientific computing
Part II: Memory systems, interconnects, and system integration

Bagus Hanindhito[1,*], Arash Fathi[2*], Dimitrios Gourounas[1], Dimitar Trenev[2], Andreas Gerstlauer[1,3] and Lizy K. John[1,3]

Abstract

In Part I of this paper (Hanindhito et al., 2026), we argued understanding technology trends in computing hardware is necessary for designing next-generation algorithms for scientific computing. Using a language that is accessible to a general computational or data scientist, or applied mathematician, we aim to help our targeted readers better understand how technology trends in computing hardware are going to impact their computations, and what characteristics algorithms should exhibit to best harness modern hardware. In Part I, we covered background material, general-purpose processors, and hardware accelerators. In Part II, we review memory systems, inter-device communication, heterogeneous computing and system integration, and why energy efficiency has become a central issue in hardware design, and why it cannot be ignored in cluster-sized computers. We conjecture how the above changes may impact scientific computing, and offer options for leveraging modern hardware for both old and modern scientific software.

Keywords

scientific computing, algorithms, computing hardware, Moore's law, SRAM, DRAM, HPC

Introduction

Motivation

Computing hardware has experienced transformative technological changes during the past two decades. Some of these changes include increased number of cores on processors, widespread adoption of hardware accelerators (e.g., GPUs), development of high-bandwidth memory technologies, and invention of new interconnects to alleviate bandwidth bottlenecks. Understanding these changes and their drivers is fundamental to effective harnessing of modern hardware, and making strategic decisions on how future algorithms and computational frameworks should be designed.

This paper reviews technology trends in computing hardware in a language that computational scientists and applied mathematicians find comprehensible. We also highlight how these changes impact scientific computing. When it comes to possibilities for designing future algorithms and workflows, as well as improving the performance of existing software, we discuss a range of options: from those that require limited resources (e.g., limited changes to an existing source code), which may yield marginal improvements to the performance, to those that need significantly more resources (e.g., more intrusive changes to a software system and possibly using a different hardware platform), which likely results in better performance improvements.

Outline and summary

Background material, general-purpose processors, and hardware accelerators are covered in Part I of this paper

[1]Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA
[2]ExxonMobil Technology and Engineering Company, Research, Spring, TX, USA
[3]Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX, USA

*Authors contributed equally.

Corresponding author:
Arash Fathi, ExxonMobil Technology and Engineering Company, Research, 22777 Springwoods Village Parkway, Spring, TX 77389, USA.
Email: arash.fathi@exxonmobil.com

[B. Hanindhito, A. Fathi, D. Gourounas, D. Trenev, A. Gerstlauer, and L. K. John, The International Journal of High Performance Computing Applications (2026)]. We suggest readers review the background material before reading the rest of the text. In Part II, we cover different memory systems, various interconnect technologies, system integration and heterogeneous computing, energy consumption of computing systems and their implications, and the impacts of these changes on high-performance scientific computing applications. Acronyms we used throughout the text are listed in Appendix A.

The *Memory Systems* section considers different memory technologies, and highlights why a whole host of different options, each having different bandwidth, latency, energy consumption, and cost, are needed in scalable computing systems. SRAM is a very fast, on-chip memory, and is typically used in registers and caches. It consumes more energy compared to alternatives, and uses space on the same precious silicon die that hosts the rest of a processor. SRAM is typically managed by hardware and compilers. Recently, some hardware accelerators have allowed users to manage a portion of SRAM, if they choose to do so; effective management by users often involves detailed familiarity with the underlying algorithms, and can result in performance gains. DRAM is off-chip memory, and typically, the primary source of providing memory capacity to applications. Compared to SRAM, it is slower, consumes less energy, provides more capacity in the same area, and is cheaper. Thanks to advanced packaging, modern processors can stack several layers of DRAM on the same package that they reside on to acquire high-bandwidth memory. This has now become common in high-end GPUs and started to appear on high-end CPUs as well. DRAM relies on capacitors to store information. Capacitor scaling is considerably more challenging than transistor scaling at advanced process nodes, and thus, it has slowed down progress in making more advanced DRAMs. Separation of memory and processing units, which characterizes the von Neumann architecture, is a major source of bottlenecks. Processing-in-memory combines the memory and compute units into a single integrated unit, which results in performance gains. Non-volatile memory technologies, such as flash memory, do not need continuous supply of energy to retain data. Compared to DRAM, they are slower, consume less energy, and are cheaper. These favorable properties have motivated their increased utilization in computing, such as integration of flash memory with DRAM to increase memory capacity and reduce power consumption.

We review technologies that are used for communication within a chip, as well as between different computing devices, such as processors, memory, and hardware accelerators in *Inter-device communication*. Communication, either within a single cluster node, or between cluster nodes, continues to be the primary bottleneck for many applications in scientific computing. This makes algorithms with a smaller communication footprint attractive to modern and emerging architectures. On-chip communication has greatly grown in complexity over the past decade, especially for Systems-on-Chip that comprise several different components that need to interact with each other. To this end, a wide variety of interconnect architectures and topologies have been suggested, each tailored to the optimization of different metrics, such as area, cost, performance and energy. Improvements in intra-node communication depend heavily on technological advancements in serial interfaces (*Part I: Architecture of communication interfaces* (Hanindhito et al., 2026)), such as NVLink, aiming at increasing the communication bandwidth. Inter-node communication is arguably the weakest link in many high-performance scientific computing applications. While inter-node communication technologies, such as InfiniBand, have steadily improved over the past decades, they trail advancements in modern microprocessors. Using more advanced signal modulation schemes improves bandwidth, and is expected to play an increasingly important role in the years to come. This will put more pressure on the host microprocessor. Data processing units are hardware accelerators that are specialized to offload this burden from the host microprocessor, and are becoming more common. Advanced technologies, such as optical interconnects for inter-node communication, can enable clustering of computing devices[1] in the future. This is referred to as disaggregated computing and will increase the efficiency of using computing resources for a diverse group of applications.

Integration aspects of contemporary heterogeneous systems are explored in the *System integration and heterogeneous computing* section. Heterogeneous computing involves leveraging various types of computing components to enhance performance and energy efficiency, based on specific application demands. This diversity is inherent not only in System-on-Chip devices but also in cluster nodes that may consist of different types of devices. Despite offering a broad spectrum of design options and high flexibility, optimizing the performance of all components in a heterogeneous system is a complicated endeavor. Numerous methods have been proposed to address the partitioning and scheduling problems that aim to determine the optimal allocation of tasks to resources. Distinct strategies are necessary for on-chip integration as well as intra- and inter-node system integration. While advanced tool support is limited and heavily reliant on the underlying architecture, it can greatly simplify this intricate task for application developers. We also provide a detailed example of system integration for the Anton specialized chip and computing system, which targets molecular dynamics simulations.

Supplying energy to supercomputing centers is becoming increasingly more challenging, as we discuss in the *Energy consumption of large computing centers and its implications*

section. While modern hardware strives to be more energy efficient, the quest to solve larger and more complex problems has led to clusters that are growing in size and total energy consumption. Some of today's largest clusters consume as much energy as a small town. This puts pressure on the power infrastructure, and can constrain or challenge upgrading a cluster into a more powerful machine. Upgrading the power infrastructure, or placing modular nuclear power plants close to large clusters of the future will likely become more common. These challenges highlight why energy efficiency heavily influences many hardware design decisions; a trend that is expected to continue.

The *Impacts on high-performance scientific computing* section examines how the reviewed technology trends impact high-performance scientific computing. On the one hand, scientific computing is very diverse and relies on non-modular and old software for many applications. On the other hand, scientific computing has a smaller market size compared to competing computing markets, such as machine learning and artificial intelligence. At times, this makes it difficult to secure sufficient resources to modernize scientific computing software in order to effectively harness modern hardware. Moreover, a large group of research scientists who develop scientific software often prioritize productivity over performance. Furthermore, they may not be well-versed in programming alternative devices, such as GPUs. These factors exacerbate the adoption of specialized architectures in such groups. Modern hardware provides a wide range of possibilities to computational scientists, where productivity versus performance may be balanced according to specific needs. These include: careful selection of computing platforms for running old applications faster while making minimal changes to the software; making substantial changes to the code and revising algorithms to effectively harness modern hardware; or even designing specialized hardware and algorithms to maximize desired performance metrics.

The *Outlook* section attempts to envision the future of high-performance computing based on technology trends that are expected in the next decade and beyond. Chiplets will flourish since they reduce the cost of hardware design through modularization. Low-power processors allow more of them to be placed on a compute node, alleviating communication bottlenecks. Disaggregated computing will result in better utilization of computing resources. Hardware specialization for large workloads will gain traction, as it will become the only viable path for improving performance. Design cost of specialized hardware will likely decrease due to the availability of open-source and automated design tools. Algorithm specialization will become even more important, in order to fully utilize modern and emerging hardware. Hardware-algorithm co-design will become more common, since it can maximize performance gains. Once they become mature, exotic computing technologies (e.g., quantum computing) may be integrated with high-performance computing to accelerate certain workloads. Consequently, the future of computing is diverse. Moreover, since access to technology and cost of design may vary around the world, some companies may find it more economical to improve performance through using more advanced manufacturing technologies, whereas others may find hardware and algorithm customization as a more sensible approach, particularly when access to advanced technologies is regulated.

Finally, the *Frequently asked questions* section summarizes frequently asked questions that are often asked by researchers, practitioners, and decision makers. It provides a high-level summary of the possibilities in the coming years.

Memory systems

Computing performance is expected to improve due to new technologies, such as hardware accelerators (*Part I: Hardware accelerators* (Hanindhito et al., 2026)), chiplet packaging (*Part I: Advanced packaging technologies* (Hanindhito et al., 2026)), and heterogeneous computing (*System integration and heterogeneous computing*).

Computing memory systems should keep up with the above advances, which entails providing more bandwidth and lower latency.

A flat memory system[2], where only one memory technology is used, would simplify both hardware and software implementation (Agarwala et al., 2000; Jacob et al., 2007). However, no single memory technology has all the desired properties: low access latency, low energy consumption, high bandwidth, large capacity, and low cost per bit (Bolotin et al., 2015; Dunning et al., 2009). Therefore, a computer system usually has a hierarchical memory structure (Rhu et al., 2013; Wang et al., 2008), as shown on the left side of Figure 1, where each level of the hierarchy is implemented with a different memory technology, raising the complexity of software and hardware design (Guo et al., 2008; Tsai et al., 2018).

The top-half of the pyramid (Figure 1) uses volatile memory technologies (Valero et al., 2012), which require continuous supply of power to retain data (Dao et al., 2021). The bottom-half of the pyramid uses non-volatile memory technologies and can retain data in the absence of power. Volatile memory technologies directly impact performance and energy consumption (Li et al., 2019a), as they are used to store hot data[3]. They must keep up with demands from the processing units. Non-volatile memory technologies are typically used to store cold data[4] and usually have a smaller impact on the performance of the whole computer system.

In this section, we review key memory technologies, and describe their performance characteristics. Specifically, we review static random-access memory (SRAM), dynamic random-access memory (DRAM), and near-memory processing (NMP) and processing-in-memory (PIM). Finally,

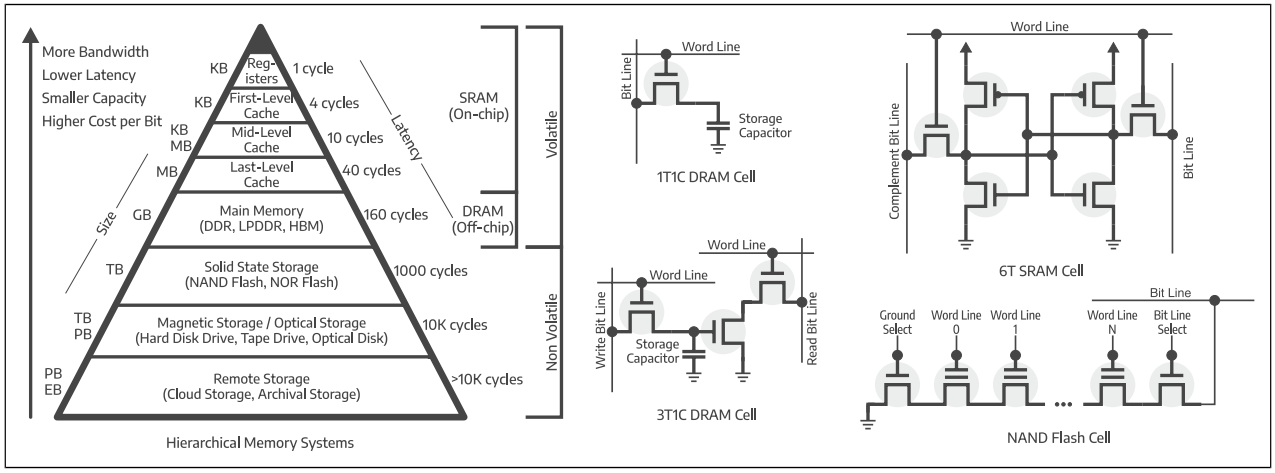

Figure 1. The hierarchy of memory systems consists of multiple memory technologies, both volatile and non-volatile. The top half of the hierarchy is volatile storage, which is implemented by using on-chip Static RAM (SRAM) and off-chip Dynamic RAM (DRAM). The registers provide very fast access, at the same level of the processing units, although, at a very limited size. DRAM provides significantly slower access, but at a significantly larger capacity. In between, multi-level cache systems are implemented to store recently-accessed data, and thus, reducing the frequency of accessing DRAM. SRAM is implemented by using 6 transistors (6T), while DRAM can be implemented either by using 3 transistors and 1 capacitor (3T1C), or 1 transistor and 1 capacitor (1T1C). The latter is preferred due to its higher density, but at the cost of a more complicated read mechanism. The bottom half of the hierarchy shows non-volatile storage technologies, consisting of flash-based storage, magnetic-based storage, and remote-based storage. The flash-based storage can be implemented by using NAND flash, consisting of transistors with a floating gate, resulting in a higher density than 1T1C DRAM. Flash has a limited number of write cycles due to endurance issues with the floating gate. The transistor-level operation details for DRAM, SRAM, and NAND/NOR Flash are not discussed; interested readers are suggested to consult Chapter 5 (SRAM) and Chapter 8 (DRAM) of Jacob et al. (2007), and Crippa et al. (2008) for NAND/NOR Flash.

non-volatile memory (NVM) and storage systems are also highlighted.

Static random access memory (SRAM)

SRAM is on-chip storage, and is a building block for registers and cache memory (Liang and Wang, 2016; Mahanta et al., 2022; Zhang et al., 2020a). It is implemented by employing similar transistors (and therefore, the same process node technology) that are being used for computing units (Huang et al., 2018; James, 2009; Lage et al., 1996). The standard form of SRAM consists of six transistors[5] (6T SRAM) (Guo et al., 2005; Margala, 1999; Weste and Harris, 2010), as shown in the right-most part of Figure 1. This structure allows for fast data reading and writing, while not requiring periodic refresh, hence named static RAM (Weste and Harris, 2010).

Registers and cache memory are key components that rely on SRAM technology, and will be discussed next. Using SRAM in hardware-accelerators (*Part I: Specialized and custom hardware* (Hanindhito et al., 2026)) is also popular due to the lower access latency and higher bandwidth it provides (at the expense of being less energy-efficient); integration of SRAM with the silicon die is also easier, compared to DRAM and HBM. We end this part by outlining different options for programming SRAM.

Registers. Registers are very fast storage elements, close to the functional units[6] of a microprocessor (Balasubramonian et al., 2001; Cruz et al., 2000a). Registers can be as fast as the microprocessor, being able to read and write in one clock cycle (Cruz et al., 2000b; Kim and Mudge, 2003), and are responsible for storing the operands and intermediate results of instructions that are being executed. While registers are fast and favorable, microprocessors can only have a limited number of them due to limited available space around the functional units and its latency constraints[7] (Kondo and Nakamura, 2005; Mittal, 2017). GPUs have a larger size of registers due to supporting a massive number of threads[8] (Gebhart et al., 2012), where each thread has its own register allocation[9]. Part I Tables 6 and 7 (Hanindhito et al., 2026) show trends of register size in NVIDIA datacenter GPUs, where it has stayed flat at 256 kB per Streaming Multiprocessor (SM) since 2013. With this limited number of registers, applications that have a large number of intermediate results do experience register spilling (Chaitin, 2004; Nuth and Dally, 1995). This results in data being moved back-and-forth between the registers and the first level of cache (Li et al., 2016; Vizitiu et al., 2014), causing delays.

Caches

Fundamental functions. Cache is a fast, hardware-managed, on-chip memory. It is used to store commonly-

used data to reduce off-chip memory access latency and bandwidth demand. Cache memory operates according to temporal and spatial locality (Lee et al., 2000). While cache operations are typically managed by hardware, understanding this process helps computational scientists develop hardware-aware algorithms[10], leading to improved performance (Christiaens et al., 1999; Cucchiara et al., 1999; Günther et al., 2006).

Data needed by a microprocessor for the first time should be accessed from off-chip memory, which incurs high access latency and energy consumption[11]. The data will then likely reside in cache for a short period of time in case the microprocessor needs to access that data again. When the data is not used for a while, it will be evicted to a higher-level cache, and, eventually, to off-chip memory[12] (Ghandeharizadeh et al., 2015). If the microprocessor needs to access this data again, it has to be fetched from higher-level caches or off-chip memory again[13]. Temporal locality relies on the idea that, most likely, recently-accessed data will need to be accessed again. Therefore, storing that data in cache saves energy, and improves performance. Spatial locality in caching suspects neighboring data of a recently-accessed data likely need to be accessed soon (Gu et al., 2009). For instance, when an element of an array is accessed, elements adjacent to the aforementioned element likely need to be accessed in the near future, and thus are brought to the cache as well.

Algorithms that exhibit temporal and spatial[14] locality (Wolf and Lam, 1991) can lead to higher performance (Kandemir et al., 1999), and lower energy consumption (Sardashti and Wood, 2013), since they need to access off-chip memory less frequently.

Hierarchical structure. Modern microprocessors employ multiple levels of cache (Liu, 1994). Compared to the registers, the first-level (L1) cache is located farther away from the functional units; therefore, it enjoys larger capacity at a slightly higher access latency (Torres et al., 2004). L1 cache is usually private to each core (Maurice et al., 2015) and stores the most-recently-used data, as well as data due to register spilling (Li et al., 2016). General-purpose microprocessors have around 64 kB of dedicated L1 cache per core, while GPUs typically have around 256 kB of L1 cache per Streaming Multiprocessor (SM).

In addition to limits posed by physical space, the L1 cache size is kept small since a larger cache entails larger access latency[15] (Hijaz et al., 2013). It is preferred to have a fast L1 cache, instead of a large L1 cache, since it should keep up with the speed of the registers (Huang and Nagarajan, 2014).

The last-level cache (LLC) is usually shared between cores of a microprocessor (Cataldo et al., 2016), and thus, is located outside of the core complex. It interfaces directly with off-chip memory[16] (Chaudhuri et al., 2019), provides a data-sharing mechanism between cores (Albericio et al., 2013), and can have a capacity of a few hundred megabytes, albeit at a higher access latency[17]. Between the L1 cache and LLC, there can be multiple mid-level caches (e.g., L2), which balance capacity and latency between the L1 cache and LLC (Chishti et al., 2005; Wang and Lee, 2008).

Specialty cache. Microprocessors may feature specialized caches whose structure is optimized for particular memory access patterns. In general-purpose microprocessors (*Part I: General-purpose microprocessors* (Hanindhito et al., 2026)), a separate instruction cache (e.g., L1-I) is used to cache the program instructions in addition to the first-level data cache (e.g., L1-D) that is used to store most-recently-used data. The instruction cache is read-only, while the data cache is capable of read and write. The split design has some advantages: (a) it doubles the aggregate bandwidth of the first-level cache, since there are two physical caches (McFarling, 1989; Smith, 1982); (b) it lowers the access latency, since the instruction cache can be physically placed near the instruction-fetch-and-decode unit, whereas the data cache can be placed near the memory unit (Smith, 1982); and (c) most importantly, it avoids interference[18] between instructions and data, since they have different access patterns (Racunas and Patt, 2003; Trancoso, 2005).

In GPUs (*Part I: GPU memory system* (Hanindhito et al., 2026)), in addition to the data caches (i.e., L1 and L2), there exists read-only texture cache, and read-only constant memory cache. Texture cache is optimized to store large amounts of data with spatial locality, with support for hardware filtering and interpolation, which is mostly beneficial for graphics applications. The constant memory cache is used to store small amounts of constant data (e.g., pre-computed constants), and provides lower access latency than texture cache.

Coherency. Coherency among the cache hierarchies needs to be maintained: if data is changed in the lower-level cache, it should be reflected in the higher-level cache, as well as in the off-chip memory (Ros et al., 2015). Consider a core modifying data in its private lower-level cache; if another core needs to access this modified data, it can obtain the data from the shared last-level cache, which should contain the correct version of the data after being modified by another core.

Cache coherency becomes more complex in multi-core microprocessors and massively-parallel architectures (e.g., GPUs) (Joshi and Ramasubramanian, 2015; Martin et al., 2012; Parvathy et al., 2016): it requires expensive hardware structures that consume additional area and power to track vast amounts of in-flight coherence requests, introduces excessive coherence traffic overheads that degrade performance, and complicates program execution through additional transient states and communication classes (Keckler et al., 2011; Singh et al., 2013). Accordingly, massively-parallel architectures do not support cache coherency[19], and this responsibility is delegated to the programmer.

User-managed vs. compiler-managed scratchpad memory. While general-purpose processors rely on hardware to manage SRAM in the form of cache, several

hardware architectures allow users or compilers to manage the on-chip SRAM explicitly. Some algorithms may have memory access patterns that cause cache trashing (Jaleel et al., 2010; Seshadri et al., 2012), reducing the effectiveness of the cache, and resulting in significant performance degradation. In these situations, explicit management of on-chip memory may improve memory access performance.

User-managed shared memory in GPUs provides this opportunity (*Part I: GPU memory system* (Hanindhito et al., 2026)). Some GPU architectures[20] have both L1 cache and shared memory implemented as unified on-chip memory. This provides flexibility to the users in sizing the shared memory, leaving the rest for L1 cache. Therefore, using shared memory effectively reduces the size of the L1 cache (Part I Tables 6 and 7 (Hanindhito et al., 2026)).

Management of the on-chip SRAM by a programmer often requires a deep understanding of the underlying algorithm and its memory-access pattern. If not done correctly, it may lead to performance loss, especially in unified architectures such as GPUs, where it affects the smaller L1 cache, which may outweigh performance gains achieved through using the shared memory.

Using compilers to manage SRAM, e.g., on hardware accelerators, is becoming more common. For instance, the only available memory on Cerebras' Wafer Scale Engine (WSE) is SRAM[21] (Lauterbach, 2021; Lie, 2022) (*Part I: Specialized and custom hardware* (Hanindhito et al., 2026)), which consumes 50% of the total chip area. WSE relies on a compiler to optimally distribute the data across the SRAM.

Dynamic random access memory (DRAM)

DRAM is an off-chip[22] memory that provides more capacity, and is more energy-efficient, compared to on-chip SRAM (Hassan, 2018), at the expense of being slower. DRAM is implemented with a different technology than the logic circuits that implement the processing units (Iyer and Kalter, 1999) and is connected to the microprocessor through an external bus. Classic DRAM was implemented by using three transistors and one capacitor (3T1C), while the most common implementation uses one transistor and one capacitor (1T1C) (Gong and Chung, 2016) (middle of Figure 1). The presence or absence of a charge stored in the capacitor represents bit 0 or bit 1. This structure allows DRAM to have a significantly higher bit density compared to SRAM, and thus lowers the cost per bit (Si et al., 2021).

Unlike SRAM, since the charge in the capacitors fades over time (Gong and Chung, 2016), DRAM needs to be refreshed periodically to maintain data integrity (Nair et al., 2014). Hence, it is named dynamic RAM. The periodic refresh occurs every few microseconds, depending on the manufacturing technology and bit density[23] (Nguyen et al., 2019). During the refresh, the data is read by the sense amplifier (Blalock and Jaeger, 1992) through sensing the charge in the capacitor. Next, the same data is written back, by putting the correct amount of charge in the capacitor. As the bit density and speed of DRAM increase, so do the negative impacts of periodic refresh on performance and power consumption[24] (Baek et al., 2014; Nguyen et al., 2019). In addition, read operations in DRAM are destructive: reading a memory cell destroys its content, and thus, rewriting after reading is required. It is worth mentioning that 3T1C implementation of DRAM[25] does not exhibit this issue (Jacob et al., 2007). Nevertheless, the 1T1C implementation is preferred due to its higher bit density (Yin et al., 2019).

In the remainder of this part, we review how DRAM evolved through the years, graphics DRAM, High-Bandwidth Memory, and the main technological challenge DRAM faces.

Classic DRAM. The first commercially-available DRAM chip was Intel 1103 (1970) (Dennard, 2018), with 1024 bits capacity[26] on a 10 mm^2 die size (Klein, 2016). It cost a penny per bit (Santo, 1988), the same as the magnetic-core memory, which was a technology it replaced. At Intel 1103 capacity, the magnetic-core memory would have had a square-foot footprint, and a pound of weight (Lojek, 2007). DRAM's success in replacing the magnetic-core memory fueled the development of larger and higher-speed DRAMs (Figure 2). The top half of Figure 2 shows trends related to important metrics in DRAM: (a) the process node with which the DRAM chip is manufactured; (b) the chip bit capacity; and (c) the pin transfer rate, which measures how fast the DRAM chip transfers data through each of its pins. DRAM chip capacity has increased through the years: it roughly doubled every 2.5 years due to smaller process nodes. However, its growth slowed down in the late 1990s due to the technological challenges highlighted in *Main technology challenges*. The data transfer rate, expressed as the pin transfer rate, has steadily increased to satisfy the need for even higher bandwidth; however, there is a trade-off between chip capacity and pin transfer rate, as discussed in *Graphics DRAM*. The bottom half of Figure 2 shows the progression of DRAM standards[27] in industry.

Fast page mode and extended data out DRAM. During the mid-1980s, the DRAM interface, which connects a microprocessor into the DRAM modules containing multiple DRAM chips, could not keep up with the demands of microprocessors (Jacob et al., 2007). Performance improvement of microprocessors (Part I Figure 5 (Hanindhito et al., 2026)) significantly outpaced that of DRAM. This led to several innovations in DRAM chip development, aimed at reducing latency, and increasing bandwidth (Jacob et al., 2007).

The Fast Page Mode (FPM) DRAM, and the Extended Data Out (EDO) DRAM, improved bandwidth over classic DRAM implementations. In classic DRAM, to request specific data, a row[28] in the memory array is selected ("opened") by using the row address. The data represented as electrical charges are then propagated to the sense amplifiers, where they are translated to digital (binary) data. Then,

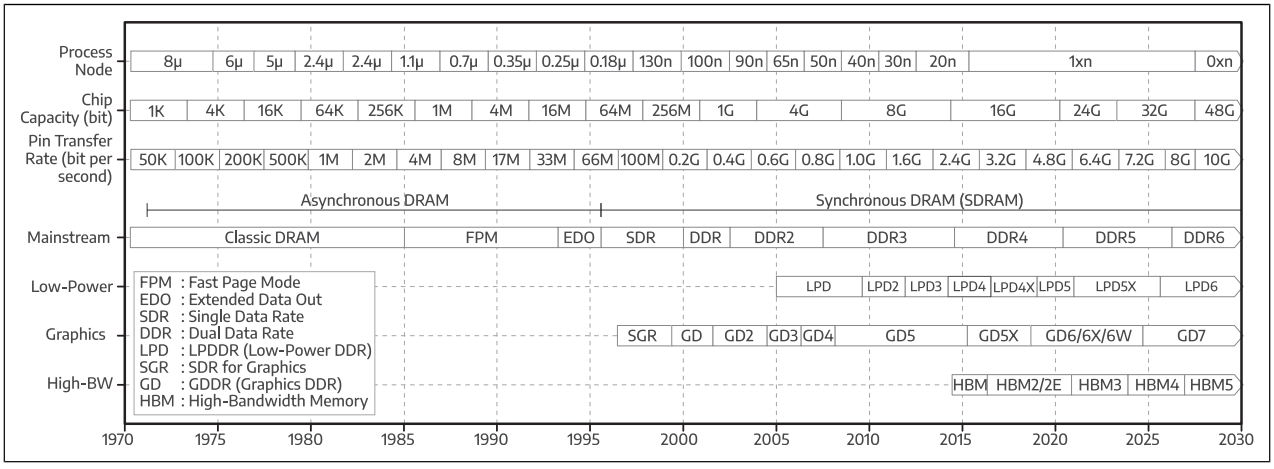

Figure 2. Evolution of Dynamic Random Access Memory (DRAM) since the 1970s, with projections until 2030.

through the column address, the requested data is selected and put into the output pins. This lengthy process must be repeated, even if the next requested data is located on the same row. FPM DRAM[29] and EDO DRAM[30] made DRAM transactions more efficient. We refer to (Jacob et al., 2007) for details. These improvements required small modifications to the structure of DRAM. Nevertheless, they improved system performance by as much as 30% (Cuppu et al., 1999a, 2001).

Mainstream synchronous DRAM. Classic DRAM, FPM DRAM, and EDO DRAM, used asynchronous interfaces between the DRAM chips and microprocessors. The asynchronous design has been primarily motivated by analog circuits (i.e., sense amplifier) in DRAM. Since the duration of operations in DRAM varies (based on different designs, process variations, and specific manufacturer), they are measured in nanoseconds, instead of number of cycles. Therefore, at the time, it was more challenging to tie DRAM's operations with the processor's clock. This asynchronous interface caused control signals of the memory controller[31], and the requested data from the DRAM chips, to arrive at DRAM pins and microprocessor pins at non-deterministic times. Accordingly, transaction events between the memory controller and DRAM chips were temporally unpredictable, making bandwidth and latency improvements challenging. Specifically, lack of synchronization and not using a common-time-reference, on the interface between the microprocessor and DRAM, makes maintaining the correctness of the transaction difficult. The microprocessor has to "wait" for the previous instruction to be completed by the DRAM chips, before issuing another instruction. The window in which the microprocessor needs to wait, typically expressed in nanoseconds, varies greatly[32], which incurs a performance penalty, limiting the achievable bandwidth and latency. Synchronous interfaces tried to address these challenges.

During the mid-1990s, DRAM started to use synchronous interfaces (Cosoroaba, 1995), where a clock signal is used to control the timing of the transactions (i.e., through the use of state machines), leading to predictability. Accordingly, DRAM speed was expressed through clock cycles, as opposed to nanoseconds (Cuppu et al., 1999a). In a synchronous interface, the transaction time is known due to using a common time reference[33], allowing the microprocessor to issue another instruction before the completion of the previous instruction[34]. The microprocessor may also issue multiple instructions to different DRAM modules[35], provided that it obeys the timing requirements of the DRAM chips (Jacob et al., 2007).

Moving from asynchronous to synchronous interfaces in the early version of synchronous DRAM (SDRAM) incurred significant implementation costs, with almost no performance gains, compared to EDO DRAM (Jacob et al., 2007). However, it provided a strong foundation for future SDRAM developments for decades to come (Table 1). We highlight the main synchronous DRAM technology developments next.

Single data rate (SDR). Single Data Rate (SDR) SDRAM (1993) was the first generation of SDRAM, where the DRAM chips operated at a voltage of 3.3 V, and had a bus clock frequency of 66 MHz (PC66), 100 MHz (PC100), or 133 MHz (PC133) (Davis et al., 2000; Jahed, 1995). With this bus clock frequency, SDR could carry data at a rate of 66 Mb/s to 133 Mb/s for each module pin.

An SDR module, called Dual In-Line Memory Module (DIMM) (Cuppu et al., 1999b; Rixner, 2004), had 64 data lanes, which were connected to the microprocessor's memory controller. Hence, it could send a word of 64 bits (8 bytes of data) in each clock cycle, translating to a module bandwidth of 533 MB/s[36] to 1066 MB/s.

To increase memory bandwidth, a microprocessor can have more than one memory channel, where each channel can have a dedicated memory controller (Jacob, 2003; Zhu et al., 2002). To increase memory capacity, more than one memory module (DIMM) per memory channel is often used (Lee et al., 2010a). For instance, a microprocessor that has two memory channels, with two DIMMs per channel, each supporting 133 MHz SDR

Table 1. Comparison of mainstream synchronous DRAM technologies.

Year	1993	1998	2003	2007	2014	2020	2026+
Technology	SDR	DDR	DDR2	DDR3	DDR4	DDR5	DDR6[b]
Voltage (V)	3.3	2.5–2.6	1.8	1.35–1.5	1.2	1.1	TBD
Chip capacity (bit)	16 M–256 M	64 M–512 M	256 M–4 G	512 M–8 G	4 G–32 G	8 G–64 G	TBD
Module size (byte)	8 M–512 M	32 M–1 G	128 M–8 G	256 M–16 G	2 G–64 G	2 G–128 G	TBD
Prefetch (word)	1	2	4	8	8	16	TBD
Internal clock (MHz)	66–133	100–200	100–266	100–266	200–400	250–550	TBD
Internal:Bus clock ratio	1:1	1:1	1:2	1:4	1:4	1:8	TBD
Bus clock (MHz)	66–133	100–200	200–533	400–1066	800–1600	2000–4400	TBD
Bus width (bit)	64	64	64	64	64	2 × 32	4 × 16
Pin transfer rate (Mb/s)[a]	66–133	200–400	400–1066	800–2133	1,600–3200	4,000–8,800	8,800–21,000
Module bandwidth (MB/s)[a]	533–1066	1600–3200	1,600–8,533	3,200–17,066	12,800–25,600	32,000–70,400	70,400–167,000

[a]Data is based on the JEDEC standard; the manufacturers may produce overclockable memory modules that exceed the standard.
[b]Based on publicly available preliminary information and may change towards the finalization of the standard by JEDEC.

modules, can achieve 2132 MB/s[37] aggregate memory bandwidth, with 1 GB of total system memory (Table 1).

Double data rate (DDR). Succeeding SDR technologies focused on improving the interface bandwidth. Instead of transferring 64 bits (8 bytes) of data at each clock cycle, Double Data Rate (DDR) SDRAM (Cosoroaba, 1997) was able to read and write two words of 64 bits (16 bytes of data) during each clock cycle (Davis et al., 2000). This was realized by using both the rising and falling of the clock edges, which effectively doubled the bandwidth.

In this generation, DRAM chips had a lower voltage of 2.5 V (Yoon et al., 1999b), allowing for reduced power consumption per bit, and higher bit capacity per chip within the same power envelope[38], with up to 1 GB of module size.

With the double prefetch[39] length, at 200 MHz of bus clock, and 64-bit module interface, the DDR module can achieve 3200 MB/s[40] of memory bandwidth, which is twice that of SDR (Yahata et al., 2000).

DDR2. Compared to DDR, DDR2 (Kyung et al., 2005) doubled the bus clock, without doubling the internal clock of the DRAM chips, which, effectively, doubled the bandwidth, but did not improve latency. To do so, DDR2 doubled the prefetch into four words (Shuang-yan et al., 2005). Since the internal clock is half the frequency of the bus clock (Prince, 2003), the latency[41] of the DDR2 module was higher than that of the DDR[42].

The voltage was reduced to 1.8 V, with a significant increase in DRAM chip bit capacity, which was enabled by using advanced process nodes (Prince, 2003). This allowed implementation of 8 GB DIMMs, with expected module bandwidth that could reach 8533 MB/s[43].

DDR3. To further increase bandwidth, a similar approach was followed by DDR3 (Fujisawa et al., 2007; Park et al., 2005), which doubled the prefetch bandwidth to 8 words, reduced the voltage[44] to 1.5 V, and doubled the chip bit capacity to allow 16 GB of DIMM size (Cui et al., 2008; Fujisawa et al., 2007).

DDR4. DDR4 (Koo et al., 2012; Shim et al., 2018) supports even higher DRAM chip bit capacity, at higher bus clock frequency (Lingambudi et al., 2016), while retaining the same prefetch width as DDR3 (Islam et al., 2014). Therefore, the DRAM chips need to interleave read and write from several bank groups to keep the bus busy (Islam et al., 2014; Sohn et al., 2013).

DDR5. The latest standard, DDR5 (Kim et al., 2019; Winterberg et al., 2023), doubles the prefetch width to 16 words (Kim et al., 2020) to reach higher bandwidth while maintaining internal clock frequency around 250 MHz to 550 MHz. It splits the bus into two 32-bit sub-channels to increase parallelism (Liu et al., 2023) and achieves up to 70.4 GB/s of module bandwidth.

DDR6. DDR6 is the successor to DDR5, slated to launch between 2026 and 2027 or even later. It doubles the number of sub-channels to four 16-bit subchannels and is expected to double the bandwidth provided by DDR5 at lower voltage. Although it was rumored to use PAM-4 modulation, JEDEC most likely will keep using the NRZ due to the complexity of the PAM-4 (*Part I: Signal coding and modulation* (Hanindhito et al., 2026)). In addition, a new module form factor called Compressed-Attached Memory Module (CAMM) will be introduced to replace the aging DIMM, accommodating the signal integrity challenge with the anticipated increase in the bus clock. JEDEC standardized CAMM for DDR5 in 2023 with limited adoption.

Mobile and low power DRAM. Mobile devices use the low-power version of DRAM chips, referred to as Low-Power DDR-SDRAM (LPDDR). The LPDDR chip is usually

placed very close to the processor (Hollis et al., 2019), either by soldering the chip closer to the microprocessor, or by putting the LPDDR chip on top of the microprocessor package (i.e., package-on-package (Hsieh, 2016; Lin et al., 2014)). The connection between LPDDR and the processor has a small bus width[45]. This integration technique, and the smaller bus width, reduces the wire resistance, which reduces power consumption.

To further reduce power consumption, LPDDR operates on a lower voltage (Hajkazemi et al., 2015), compared to the standard DRAM: 1.8 V on LPDDR, 1.2 V on LPDDR2 (2009) and LPDDR3 (2012), 1.1 V on LPDDR4 (2014), and 0.6 V on LPDDR5 (2019) and LPDDR5X (2021). Moreover, the periodic refresh operations are optimized by applying multiple techniques that control power consumption aggressively (Baek et al., 2014; Hemani and Klapproth, 2006). LPDDR's lower power consumption is attractive to data centers and high-performance computing clusters. For instance, LPDDR5X is used for the Grace CPU memory in NVIDIA Grace-Hopper (GH200) and Grace-Blackwell (GB200) (Tirumala and Wong, 2024) CPU-GPU heterogeneous platforms.

Graphics DRAM. GPUs that rely on high-bandwidth memory often need different approaches for designing the DRAM chips, along with tighter integration of memory with the device.

Higher bandwidth for memory can be achieved through: (a) constructing a wider memory bus (Kim et al., 2014; Li et al., 2018); (b) increasing the bus clock frequency (Cho et al., 2012); (c) running DRAM chips at a higher internal clock frequency (Woo, 2010); and (d) utilizing denser signal modulation (*Part I: Signal coding and modulation* (Hanindhito et al., 2026; Horowitz et al., 1998; Wang and Buckwalter, 2011) between the microprocessor and DRAM chips. In what follows, we review how the memory bandwidth of GPUs has increased over the years through the evolution of various technologies.

Evolution of graphics memory technology. Earlier DRAM for GPUs was constructed by using video RAM (VRAM) to satisfy the bandwidth requirements at the time (Prince, 1999). VRAM is an ancestor of SDRAM, which comprised multi-ported[46] asynchronous DRAM, and serial access memory (SAM). This allowed VRAM to operate as an asynchronous DRAM in one port, while having a synchronous serial memory interface in another port.

In 1997, GPUs started to use Synchronous Graphics Random Access Memory (SGRAM), which was derived from the Synchronous DRAM (SDRAM), eliminating the need for more expensive, multi-port asynchronous DRAM (Prince, 1999).

The successor of SGRAM is Graphics Double Data Rate SDRAM (GDDR-SDRAM, which was initially known as DDR-SGRAM (Foss, 1997; Prince, 1999)). It was based on the Double Data Rate SDRAM (DDR-SDRAM) (Cosoroaba, 1997). The GDDR-SDRAM chip is specifically designed to run at higher internal clock frequencies, compared to mainstream DRAM. To do so, it sacrifices bit capacity per chip[47] (Dunning et al., 2009), by adding more periphery components, which facilitate faster memory transactions. The increase in internal clock frequency of GDDR chips requires more cooling, as they dissipate more heat compared to mainstream DRAM chips.

Table 2 shows the evolution of GDDR-SDRAM, which closely resembles the developments of mainstream DRAM (Table 1): GDDR (1999) and GDDR2 (2002)[48] were derived from DDR. GDDR3 (2004) (Lee et al., 2006) was developed from DDR2, and GDDR4 (2005) (Bae et al., 2008) and GDDR5 (2007) (Kho et al., 2010) were developed from DDR3. Their successors, GDDR5X (2016) (Brox et al., 2018)

Table 2. Comparison of graphics synchronous DRAM technologies.

Year	2004	2007	2017
Technology	GDDR3	GDDR5	GDDR6
Voltage (V)	1.8	1.5 V	1.35 V
Chip layer (#)	1	1	1
Layer capacity (bit)	256 M–1 G	2 G–8 G	8 G–32 G
Chip capacity (bit)	256 M–1 G	2 G–8 G	8 G–32 G
Chip I/O width (bit)	32	32	2 × 16
Bus clock (MHz)	200–1242	500–2257	1250–3000
Pin transfer rate (Mb/s)	400–2484	2000–9028	10,000–24,000
Chip bandwidth (MB/s)	3200–9936	8,000–36,112	40,000–96,000
Device bandwidth (GB/s)			
64-bit (e.g., 2 chips)	3.2–19.9	16–72.2	80–192
128-bit (e.g., 4 chips)	6.4–39.7	32–144.4	160–384
256-bit (e.g., 8 chips)	12.8–79.5	64–288.9	320–768
384-bit (e.g., 12 chips)	19.2–119.2	96–433.3	480–1152
512-bit (e.g., 16 chips)	25.6–159.0	128–577.8	—

and GDDR6 (2017) (Kim et al., 2021c), use Quad Data Rate[49] (QDR) (Kim et al., 2016) to achieve even higher bandwidth.

The GDDR6X (2020) (Hollis et al., 2022) can[50] provide triple the bandwidth of GDDR5[51] through using the same bus width, but by using a higher bus clock frequency, and denser signal modulation[52]. With their higher internal clock frequency, denser signal modulation, and higher bus clock frequency, GDDR6X chips run at junction temperatures[53] exceeding 100°C. The high temperature is concerning to some users, since it may impact the longevity of their products[54]. Unlike GDDR6X, next-generation GDDR7 uses PAM-3 instead of PAM-4 modulation to reduce costs, complexity, and power while delivering a meaningful increase in pin data rate over GDDR6. GDDR7 was launched in early 2025 with the launch of consumer-class NVIDIA Blackwell GPUs.

Integration of memory with GPUs. GDDR chips are closely integrated with GPU (Kim et al., 2016; Li et al., 2018) (Figure 3). This allows GDDR chips to run at higher bus clock frequencies, and use denser signal modulation. However, parasitic capacitance from the printed circuit board (PCB) materials[55], and interference between adjacent wires[56], become more severe at higher frequencies and longer distances (*Part I: Wiring, connectivity, and signal integrity* (Hanindhito et al., 2026)). Therefore, to achieve higher bus clock frequencies, and to use denser signal modulation, the distance between the GPU chip and GDDR chips must be minimized. Typical placement of a GPU with a 384-bit memory bus, when connected to GDDR5 chips, is shown on the left-side of Figure 3, whereas the typical placement of a GPU with 384-bit memory bus, when connected to GDDR6X chips is shown in the middle of Figure 3.

Increasing the memory bandwidth can also be realized by widening the bus (Mahapatra and Venkatrao, 1999), through adding more GDDR chips, and more memory channels. For instance, a low-end GPU can have a 64-bit bus-width, consisting of two GDDR chips, whereas a high-end GPU can have a 512-bit bus, consisting of as many as 16 GDDR chips.

Manufacturing a wider bus is challenging, since it requires routing more signal paths on a PCB (Na et al., 2017; Nitin et al., 2018). Each bit line on the memory bus becomes a single wire on the PCB. Each wire carries a high-frequency signal, and thus, can interfere with neighboring wires, which then affects data integrity (*Part I: Wiring, connectivity, and signal integrity* (Hanindhito et al., 2026)). Therefore, to realize a wider memory bus, new technologies that overcome the physical limitation of PCB are needed.

High-bandwidth memory (HBM) and variants. Advances in packaging technologies (Part I Figure 6 (Hanindhito et al., 2026)) allow DRAM dies to be stacked on top of each other, and then be placed on the same package that a GPU resides (the right-side of Figure 3) (Loh et al., 2015). Instead of using wires that are implemented on PCB to connect a GPU to the stacked DRAM chips, a silicon interposer (*Part I: Advanced packaging technologies* (Hanindhito et al., 2026)) is used (Cho et al., 2015; Lee et al., 2015b). Each stack of the

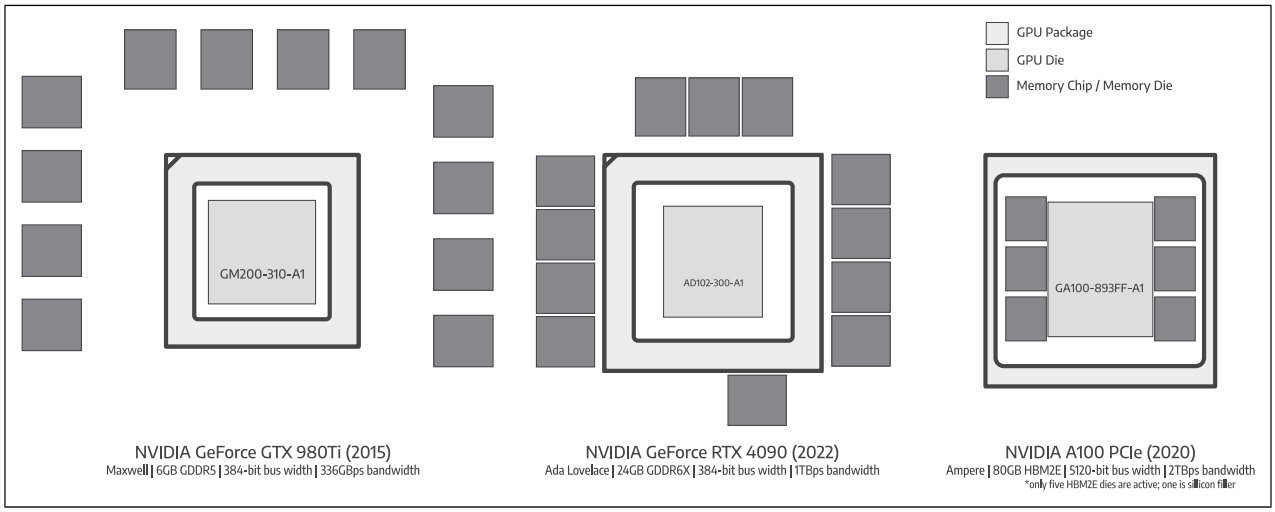

Figure 3. Evolution of DRAM interface on GPU, where providing higher bandwidth is the main design objective. NVIDIA GeForce GTX 980Ti has 338 GB/s memory bandwidth, implemented by using 12 chips of GDDR5 memory, each having a 32-bit bus width, resulting in a total of 384-bit bus width. The newer NVIDIA GeForce RTX 4090 uses 12 chips of GDDR6X memory with the same total bus width of 384-bit. With a significantly faster bandwidth of 1 TB/s coming from its higher effective transfer rate, the memory chips must be placed as close as possible to the GPU package to maintain signal integrity. Finally, High-Bandwidth Memory (HBM) provides a wider memory bus of 1024-bit per chip (i.e., HBM stack) to achieve significantly higher memory bandwidth. Implementing this wide memory bus on printed circuit board (PCB) is challenging. Therefore, HBM die is usually placed on the same package that the GPU resides on through using a silicon interposer (e.g., in NVIDIA A100 GPU), as explained in *Part I: Advanced packaging technologies* (Hanindhito et al., 2026).

DRAM chips can have a bus width as wide as 1024 bits (Kim, 2015; Martwick and Drew, 2015), which is 32 times wider than the bus width of GDDR chips. This makes implementation of a wider memory bus at lower interconnect power dissipation possible (Zhao et al., 2017).

There were two standards for memory technology that use three-dimensional stacks of DRAM dies along with silicon interposer for connectivity: Hybrid Memory Cube (HMC) and High-bandwidth Memory (HBM). HMC was a proprietary standard by Micron (Jeddeloh and Keeth, 2012), whereas HBM is standardized by JEDEC. In 2018, Micron discontinued HMC in favor of HBM due to low market adoption. Therefore, only HBM is discussed in this section. Table 3 shows the evolution of HBM-SDRAM.

The first-generation[57] HBM (2015) can stack up to 4 DRAM dies, enabling 8 Gb chip capacity (Lee et al., 2015a; Macri, 2015). HBM2 (2016) (Cho et al., 2018) and HBM2E (2018) (Chun et al., 2021; Lee et al., 2020b) increased the number of DRAM dies to 8, realizing 64 Gb of chip capacity, while maintaining the same 1024-bit bus interface. The latest generation, HBM3 (2021) (Park et al., 2023; Ryu et al., 2023), increases the number of stacked DRAM dies to 12, with the possibility of having 16 stacked DRAM dies in the future, pushing the chip bit capacity beyond 192 Gb. Its successor, HBM3E, was launched in 2024.

Stacking more DRAM dies into the HBM increases the chip memory capacity. However, the parasitic capacitance of thru-silicon vias (TSV), which connects each layer of the stack, becomes more problematic, as the stack size grows (Farmahini-Farahani et al., 2018; Kim et al., 2021e). Moreover, stacking more DRAM dies on the chip makes thermal dissipation of HBM chips more challenging, as the surface area of the chips remains the same (Kim et al., 2023; Lee et al., 2023b). Therefore, the next generation of HBM, HBM4 (2025+), is expected to have a vapor-chamber cooling system, which is a cooling technology that is often used for chips that need high thermal dissipation. Moreover, HBM5 (2027+) is expected to have a micro-channel cooling system.

Despite the above challenges, HBM has been successful in fulfilling the demand for memory bandwidth of high-end GPUs. However, high manufacturing costs and low production capacity (Jun et al., 2017; Abdennadher et al., 2018) limit the adoption of HBM to data center class products, leaving the consumer market with GDDR.

In addition to GPUs, HBM has been used in many other devices that need high-bandwidth memory. For instance, Intel Knights Landing, a manycore architecture (*Part I: Many-core processors* (Hanindhito et al., 2026)), features 16 GB Multi-Channel DRAM[58] (MCDRAM) (Pohl and Sattler, 2018; Sodani, 2015), which was derived from HMC (Jeddeloh and Keeth, 2012). In addition, Intel Xeon Max[59] features (on-package) 64 GB HBM2E (Biswas, 2021), providing considerable amount of memory bandwidth. This can especially benefit applications that fit into the HBM. In both cases, conventional (off-package) DDR5-SDRAM is still provided to make up for the limited capacity of HBM, which can be configured in multiple ways. HBM is also used in FPGAs (Holzinger et al., 2021; Shi et al., 2022), CGRAs (Kim et al., 2017), and ASICs (Jouppi et al., 2020, 2021) (*Part I: Specialized and custom hardware* (Hanindhito et al., 2026)).

Main technology challenges. As DRAM moves to more advanced process nodes, the chip bit density increases. This makes larger-capacity memory modules possible, and drives down cost-per-bit. However, DRAM is facing difficulties in moving to more advanced process nodes: it has stagnated in the

Table 3. Comparison of high-bandwidth synchronous DRAM technologies.

Year	2015	2016/2018	2021/2023
Technology	HBM	HBM2/2E	HBM3/3E
Voltage (V)	1.2	1.2	1.1
Chip layer (#)	4	4–8	4–12+
Layer density (bit)	2 G	8 G	16 G
Chip density (bit)	8 G	32 G–64 G	64 G–192 G+
Chip I/O width (bit)	8 × 128	8 × 128	16 × 64
Bus clock (MHz)	500	700–1600	1313–1600
Pin transfer rate (Mb/s)	1000	1400–3200	2626–3200
Chip bandwidth (MB/s)	128,000	179,200–409,600	336,128–409,600
Device bandwidth (GB/s)			
1024-bit (e.g., 1 stack)	—	179.2–409.6	—
2048-bit (e.g., 2 stacks)	—	358.4–819.2	—
4096-bit (e.g., 4 stacks)	512	716.8–1638.4	—
5120-bit (e.g., 5 stacks)	—	896.0–2048	1680.6–2048
8192-bit (e.g., 8 stacks)	—	1433.6–3276.8	2689–3276.8

1x nm process node[60] (Kang et al., 2014; Mellor, 2020; Shiratake, 2020) for almost a decade, and is expected to remain in this range for the foreseeable future (Chen et al., 2023b).

The structure of DRAM is the main cause of this difficulty: DRAM uses a capacitor to store a charge for representing bit 0 and bit 1. While more advanced process nodes have enabled transistor shrinking, capacitor shrinking remains challenging. As a capacitor becomes smaller, the electrical charge it can hold is also reduced (Chen et al., 2023b). Eventually, it becomes difficult for the sense amplifier to detect the charge (Shiratake, 2020). Additionally, due to the smaller charges that capacitors can hold, they need to be refreshed more frequently, reducing DRAM's performance (Khan et al., 2014; Liu et al., 2013). Moreover, as the bit density of DRAM becomes larger, the smaller charge has to travel through longer wires in DRAM chips, making it difficult to maintain data integrity (*Part I: Wiring, connectivity, and signal integrity* (Hanindhito et al., 2026)).

Due to these challenges, development of next-generation DRAM technologies is costly. Indeed, there are only three major players[61] in the DRAM industry, due to its reliance on large research and development budgets.

Near-memory processing (NMP) and processing-in-memory (PIM)

Centralized memory and compute units are separated from each other in the von Neumann architecture, which constitutes the vast majority of modern computing systems. The interface between the compute unit and off-chip memory has limited bandwidth, and often becomes the bottleneck in overall system performance. In addition, excessive data-movement consumes a lot of energy[62]. Near-memory processing (NMP) and Processing-in-memory (PIM) attempt to address the von Neumann bottleneck and reduce energy consumption by bringing the compute units closer to where data is stored (Khoram et al., 2017; Mutlu et al., 2019). With their promising performance and reduced energy consumption, NMP and PIM are attractive for data-intensive workloads. Nevertheless, these technologies are still in their infancy. A better software stack[63] is also critical for the adoption of these technologies by the end users (Ghose et al., 2019).

Near-memory processing (NMP). The NMP brings the compute units near the memory arrays. Accordingly, the compute units and memory arrays can be integrated at chip or package levels. The capability of the compute units depends on their size. Usually, when the compute unit is placed on the same die where the memory arrays are implemented, there is competition for space, which typically results in simpler compute units. Next, we highlight a few examples that use NMP.

Cerebras and SambaNova. Cerebras' Wafer Scale Engine and SambaNova's Reconfigurable DataFlow Unit (*Part I: Specialized and custom hardware* (Hanindhito et al., 2026)) distribute arrays of on-chip memory units (SRAM) and arrays of compute units throughout the chip. This design enables interaction of memory and compute units with very high bandwidth.

UPMem. UPMem (Devaux, 2019) develops NMP through integrating compute units within the same die that a DRAM memory array resides on, referred to as DRAM processing units (DPUs[64]). Since both the compute and memory units are implemented on the same die, the compute logic has to be manufactured by using the same process node technology that is used for DRAM cells. This leads to a suboptimal design, since the logic circuits could have been implemented with a smaller process node technology otherwise. Nevertheless, substantial performance improvements and energy savings are reported due to parallel operation of thousands of DPUs[65], as well as the availability of significantly high bandwidth between the memory and compute units. Furthermore, UPMem DPU modules are drop-in replacements for standard memory modules[66], allowing seamless transition from an existing memory technology to an NMP technology.

Samsung. Samsung released their version of NMP by integrating Programmable Computing Units (PCUs) into their High-Bandwidth Memory (HBM-PIM) and LPDDR5 (LPDDR5-PIM) (Kim et al., 2021b; 2022a). While they use PIM in naming their products, the associated technology that they use is still NMP. In addition, Samsung also released Acceleration DIMM (AxDIMM), which integrates reconfigurable logic to the standard DDR4 memory modules (Ke et al., 2022).

Processing-in-memory (PIM). Unlike NMP that relies on separate compute units placed near the memory arrays, PIM performs the computations directly in the memory arrays. The approach varies based on the utilized memory technology. However, it usually requires minimal changes to the memory array structures, and relies on altering the memory commands issued by the memory controller to perform the computations. PIM has been implemented in different memory technologies: SRAM (Fujiki et al., 2021c), DRAM (Fujiki et al., 2021a), and Non-volatile memory[67] (Fujiki et al., 2021b).

Based on its operation, PIM can be divided into two categories: analog (Feinberg et al., 2018) and digital (Imani et al., 2019a). Analog PIM leverages inherent electrical properties of the memory arrays to perform computations according to Kirchoff's law[68] (Zhang et al., 2020b). Analog circuits are more sensitive to noise, manufacturing variations, temperature changes, and voltage fluctuations. Moreover, inside SRAM-based and NVM-based PIM, the analog-to-digital and digital-to-analog converter blocks consume the majority of area and power of the memory chip (Talati et al., 2016). Digital PIM attempts to address this issue by performing digital logic[69] operations on the memory arrays. Through using logical operations, more complex arithmetic operations can be

performed. However, this makes arithmetic operations, such as addition and multiplication, have significantly longer latency since the operands are processed bit-by-bit. Although individual arithmetic operations take longer latency, significant performance improvements result from the massively parallel operations that memory arrays of digital PIM can perform.

Examples of PIM implemented in DRAM include Compute DRAM (Gao et al., 2019) and Ambit (Seshadri et al., 2017), whereas Neural Cache (Eckert et al., 2018) is an SRAM PIM. Non-volatile memory technologies investigated for PIM include phase-change memory (Hoffer et al., 2022), resistive RAM (Hanindhito et al., 2021; Imani et al., 2019b), spintronic RAM (Chowdhury et al., 2018), and NAND Flash (Gao et al., 2021).

Non-volatile memory (NVM)

Non-volatile memory technologies enable permanent storage of data in the absence of power. Flash-based NVM is becoming more popular, not only for storing cold data, but also for storing hot data due to its higher density compared to DRAM, especially for HPC and ML applications that deal with big data. Emerging NVM technologies, such as phase-change NVM, are actively being investigated as an alternative to Flash-based NVM, which has been approaching its scaling limit.

Flash-based memory. Flash memory is an early example of NVM, and was invented in 1984 (Masuoka et al., 1984). It can be constructed by using NAND[70] Flash or NOR[71] Flash. NOR Flash has a higher random access speed compared to NAND Flash (Wong, 2010), along with a higher cost-per-bit. A NOR Flash cell[72] is 2.5× larger than a corresponding NAND Flash cell (Van Houdt, 2006). Due to its high bit density, and low cost-per-bit, NAND flash is generally preferred for manufacturing flash-based storage[73] (Lu, 2012).

We review key characteristics and technology trends in NAND Flash, and highlight how they are being used close to microprocessors to augment DRAM-provided memory capacity.

Flash memory operations. NAND flash (right of Figure 1) uses two transistors to control read and write to the cells, along with several floating-gate transistors for storing the data. Data is stored by maintaining an electrical charge in the floating gate, which is surrounded by oxide layers as insulators[74] (Friederich, 2010). Unlike the capacitor inside DRAM, the floating gate transistor can hold its charge for years, without the need for periodic refresh.

Each write operation to the cell slightly damages the oxide layers due to the high voltage needed for either pushing an electron into the floating gate, or removing an electron from the floating gate (Zambelli et al., 2010). After a sufficient number of write cycles, the oxide layers are damaged to the point where the floating gate cannot hold the charges, making the cell unusable. Therefore, unlike a DRAM cell, which has virtually no degradation (Itoh, 2011), NAND Flash[75] has a limited number of write cycles. Total bytes written (TBW), and drive writes per day (DWPD), are used as metrics to represent endurance of flash memory products (Li et al., 2019b; Woo et al., 2020). To improve endurance, manufacturers usually provide more cells than the stipulated capacity, and use them for over-provisioning to substitute failing cells with spares (Li, 2020). The flash memory controller is responsible to perform wear-leveling (Dharamjeet et al., 2022; Liao et al., 2015), by managing the use of the cells in such a way that all of them degrade with the same rate.

Miniaturization challenges. Increasing the bit density of flash memory can be achieved through transistor miniaturization. However, shrinking the size of the transistors reduces their endurance, since the oxide layers become thinner, making them more susceptible to damage (Koh, 2009). The use of floating gate transistors limits[76] future scaling of flash memory, and thus, a new cell structure, referred to as charge trap, is being used (Lu, 2012). Instead of using semiconductors as storage elements[77], the charge trap uses dielectric layers[78] to hold the charge (Grossi, 2010). Compared to the floating gate, charge trap has lower reliability. However, due to advances in processing materials, charge trap is now preferred, especially for 3D NAND (Advani, 2016).

Multi-level cell. Bit density can also be increased by employing multi-level voltage in a cell, which can then represent more than one bit (Micheloni and Crippa, 2010). In a single-level cell (SLC) (Kouchi et al., 2020, 2021), there are only two levels of voltage, which represent bit 0 and bit 1. In a multi-level cell (MLC) (Lee et al., 2016b; Micheloni et al., 2006), there are four levels of voltage, which represent two bits, with values 00, 01, 10, and 11 (Crippa and Micheloni, 2010). Triple-level cell (TLC) (Higuchi et al., 2021; Siau et al., 2019), and quad-level cell (QLC) (Kalavade, 2020; Shibata et al., 2019) NAND flash, which can store three and four bits per cell, respectively, have found their way to the consumer market, where the cost per bit is important. Penta-level cell (PLC) (Ishimaru, 2019), and hexa-level cell (HLC) (Aiba et al., 2021) NAND flash are currently being developed to store five bits and six bits in a single cell, respectively, further increasing the bit density of flash memory.

Having more bits in a cell comes with two major challenges: (a) slower read and write performance, due to more complicated read and write mechanisms[79]; and (b) lower reliability and endurance, due to higher susceptibility to errors (Jaffer et al., 2022). The former can be improved by using DRAM or SLC flash as cache, in front of the main multi-level flash cells (Alsalibi et al., 2018; Matsui et al., 2017), whereas the latter issue can be improved by using more advanced error correction mechanisms (Nicolas Bailon et al., 2022). In addition, high-order multi-level cells, such as PLC and HLC, are especially useful[80] in

Write-Once-Read-Many (WORM) applications, such as archival storage.

3D structure. A 3D structure[81] (Parat and Goda, 2018) for flash memory also enables higher capacity. This was first commercialized by Samsung: 3D V-NAND (2013) had 24 layers of stacked charge trap cells (Elliott and Jung, 2013). Due to increased demand for high-capacity flash storage, the number of layers has increased significantly, to more than 200 layers at the end of 2022. Samsung predicted (2021) 3D NAND flash will reach 1000 layers by 2030, answering the call for higher capacity flash storage (Kim, 2021; Nitayama and Aochi, 2011).

Between DRAM and NAND. Leaving endurance issues aside, high density and low cost-per-bit of flash memory makes it attractive for storing hot data, and thus, complementing DRAM. We highlight some examples.

3D XPoint (Wu et al., 2016) is a non-volatile memory technology, developed by Intel and Micron, which aims to be a bridge between DRAM and NAND Flash (Bourzac, 2017; Yang et al., 2020). Its access latency and cost-per-bit are between those of DRAM and NAND flash, whereas its endurance is higher than that of NAND Flash. 3D XPoint is used in Intel Optane DC Persistent Memory (DCPMM), and Intel Optane DC Solid State Drives (DCSSD).

DCPMM is used to provide additional capacity to DRAM, by using NVDIMM[82]; it can co-exist on the same DIMM slot, as a standard DDR4 DIMM[83] (Chen et al., 2016; Lee et al., 2020a), providing up to 4.5 TB of memory per socket on Intel Xeon Cascade Lake and Intel Xeon Cooper Lake CPUs, and up to 6 TB of memory per socket on Intel Xeon Ice Lake CPUs[84]. These values are enormous, and this memory capacity could not have been achieved otherwise, even by using all DRAM DIMMs supported by a CPU[85].

DCSSD is used to improve the performance of NAND flash, especially those types that use multi-level cells (Zhang et al., 2018).

Emerging memory technologies. We highlight several emerging memory technologies next.

Phase-change memory (PCM) uses materials that have two states with distinguishable electrical and optical properties. PCM promises to bridge the gap between fast, volatile (short-term), on-chip memory, and slow, non-volatile (long-term), off-chip memory[86] (Pernice and Bhaskaran, 2012). An alloy, known as GST[87], is one of the near-perfect materials for PCM, with sub-nanosecond switching time and stable operations, even after 10^{12} switching cycles (i.e., endurance) (Lencer et al., 2008; Raoux et al., 2008). Therefore, some researchers are investigating the integration of GST into microprocessors[88] (Ríos et al., 2015). GST enables simultaneous reading and writing through using multiple wavelengths, when optimized algorithms are used (Rios et al., 2014; Stegmaier et al., 2017). This makes them good candidates for alleviating memory bottlenecks of modern microprocessors, while maintaining competitive power consumption and bit density (Zhai et al., 2018).

Similar to PCM, resistive random access memory (RRAM) stores information by allowing for externally controlled modulation of electrical properties, in particular resistance, usually by changing the structure of the solid-state dielectric material between two electrodes. Compared to other technologies, RRAM has a simpler structure, faster storage speed, higher bit density, lower power consumption, and better compatibility with CMOS technology (Ji et al., 2016; Ligorio et al., 2017). While memristors are a form of RRAM, some argue conversely that all RRAM can be considered as having memristive properties[89] (Chua, 2011), although it is still an open question.

Opto-electronic materials are actively being investigated for developing light-assisted, field-effect transistor (FET) memory. Light-assisted FET memory is a promising candidate for replacing silicon-based FET memory. The latter has been widely used in flash-based NVM, and is reaching its scaling limits. Some of the materials that are currently explored (Zhai et al., 2018) include organic materials (polymers) (Narayan and Kumar, 2001; Noh et al., 2005; Tsuji and Nakamura, 2017), photochromic materials (Frolova et al., 2015b, 2015a; Jeong et al., 2016), photoluminescent materials (Li et al., 2015a; Mishra et al., 2016; Pinchetti et al., 2016), and two-dimensional metal dichalcogenide materials (Lee et al., 2016a, 2017). Lastly, using opto-electronic materials[90] for constructing RRAM is also receiving considerable attention.

Storage systems

Magnetic storage. Magnetic-based storage is used for permanent safeguarding of data. It has a significantly lower cost-per-bit, compared to flash-based storage, albeit, at lower performance. We highlight two key technologies that rely on magnetic-based storage.

Hard disk drive. Hard disk drive (HDD) is used to store less-frequently-accessed data. It represents a bit of data by using the direction of magnetic grains on top of magnetic platters. Heads, which are used to read and write data, are positioned on top of the magnetic platters, at a distance in the order of nanometers.

In recent years, the hard disk drive has not been growing in capacity. The largest 3.5-inch HDDs are 24 TB and 28 TB, for conventional magnetic recording (CMR) (Iwasaki, 1984), and shingled magnetic recording (SMR) (Amer et al., 2011), respectively.

Increased HDD capacity can be realized by adding more platters (Fontana et al., 2015). However, there is a limit to this, due to the form factor[91] of the HDD (Paulson, 2005). Moreover, a more powerful motor needs to be used to rotate the discs, which increases energy consumption (Hylick et al., 2008). Another strategy relies on increasing the platter bit density, by reducing the size of the magnetic

grains. This requires careful positioning of the head, very close to the platter. The actuator's arm must also become more precise, which cannot be achieved in atmospheric air conditions, due to aerodynamic drag.

In large-capacity drives, Helium in a sealed container is used instead, which allows for precise movement of the head, and reduced power consumption (Aoyagi et al., 2022). Smaller magnetic grains are less stable (Thompson and Best, 2000; Wood, 2000), and tend to change magnetic direction, leading to data corruption. To achieve stability on smaller grains at a certain temperature, a mix of materials to create a stable magnetic medium[92] has been proposed. However, writing on such material is challenging, since it is already stable. Energy-assisted magnetic recordings, such as heat-assisted magnetic recording (HAMR) (Kryder et al., 2008; Rottmayer et al., 2006), and microwave-assisted magnetic recording (MAMR) (Zhu et al., 2008), are used to write into such materials, by heating up the material, which reduces grain stability (Nordrum, 2019; Shiroishi et al., 2009). This approach is believed to improve HDD capacity beyond 50 TB by 2030.

Magnetic tape. Magnetic tape is primarily used for archival storage (Caddy, 2022; Dee, 2008). It is slow due to its nature of being a sequential-access-media. Nevertheless, it provides good data retention, and is expected to be around for decades to come.

Optical storage. Optical discs[93] store data by encoding them through a change in the way light is reflected. They are typically used for distributing read-only media[94]. Due to the popularity of cloud storage services for content delivery, as well as availability of cheaper flash-based NVM options, usage of optical discs is declining. Therefore, we do not review them.

What comes next?

Future general-purpose processors and accelerators (*Part I: What comes next?* (Hanindhito et al., 2026)) demand more memory bandwidth to feed their compute units. Accordingly, next-generation memory technologies aim to provide sufficient bandwidth in addition to higher capacity. While the advancement of SRAM will follow that of the process node technology[95], developments in DRAM face unique technological challenges.

The DRAM process node will stay at 1x nm for the foreseeable future. Nevertheless, manufacturers are actively researching technologies to improve the bit density and bandwidth of DRAM chips. While manufacturers struggle to bring the sub-1x nm process node to DRAM, a prominent technology currently being explored is 3D DRAM. Not to be confused with high-bandwidth memory (HBM), which stacks multiple DRAM dies, 3D DRAM consists of a single monolithic die with memory cells stacked on top of each other, in addition to the horizontal arrangement used in a planar memory die. Vertical stacking of memory cells[96] creates a broader gap between transistors. Therefore, it reduces interference and allows higher bit density per memory chip. Mainstream and graphics DRAM will see slight improvements in chip density and bandwidth, while HBM will continue to become the de facto memory for devices that require high bandwidth. Trends in pin data rate, chip density, and stack height (for HBM) will continue translating to higher bandwidth and greater device capacity.

Further developments in Near Memory Processing (NMP) and Processing-in-Memory (PIM), along with their integration into existing computing systems, may alleviate bandwidth bottlenecks in many applications. Software support[97] is crucial, and improves adoption of NMP and PIM, as it becomes easier for users to benefit from these technologies. This also holds true for custom accelerators that use compiler-managed memory.

Non-volatile Memory (NVM) will see exciting developments in terms of memory technologies, endurance, density, and cost-per-bit in the next decade. Finally, the quest for finding new technologies that combine the best of DRAM[98] and NVM[99] will remain strong.

Summary and remarks

Technological advances in processing units outpaced those in memory systems. Therefore, interactions between processing units and memory systems continue to be a major bottleneck for many applications, making algorithms that have reduced memory footprint attractive. Often times, several memory technologies need to work in tandem to alleviate these bottlenecks.

SRAM provides very fast, on-chip memory, and is often used in registers and caches. Compared to other memory technologies, SRAM is more expensive, uses more power, and consumes space on the same precious silicon die area that compute units reside. Some custom hardware use SRAM as their primary memory system to attain faster runtime for their targeted applications. SRAM is typically managed by hardware (i.e., cache), although some chips use a compiler or put the burden on the programmer to manage it. User-managed SRAM, typically supported by hardware-accelerators, is becoming more common, and may be exploited to improve performance. Successful management, however, will need intimate knowledge of the underlying algorithms of an application, and how the algorithms interact with the compute units and memory systems.

DRAM is typically implemented off-chip. Compared to SRAM, it provides larger capacity, but has a lower bandwidth. The most common DRAM implementation uses a capacitor and a transistor to represent a bit. Due to the leak of electrical current in the capacitor, DRAM needs to be refreshed periodically. While transistors are getting smaller, it is very difficult to make capacitors smaller. Smaller capacitors can hold a smaller charge, which makes sensing the charge more difficult. Moreover, the smaller capacitors need to be

refreshed more frequently, which reduces their performance. Not being able to make these transistors smaller is the biggest technological challenge that DRAM faces for the foreseeable future.

DRAM bandwidth is impacted by bus-width, which is the number of "lanes" or wires between the memory and the compute unit. Widening the bus is challenging, and therefore several technologies have been deployed to improve the memory bandwidth. These include: (a) using both the rising and falling edge of the clock to perform read and write operations; (b) increasing the memory bus clock frequency, and making the memory bus busy by using a wider prefetch; (c) interleaving read and write operations from several memory chips; and (d) splitting the bus channel into multiple independent sub-channels in order to increase concurrency.

DRAM memory that is used in GPUs (GDDR) needs to support high-bandwidth. Increased bandwidth is enabled through: (a) adding more periphery components to accelerate memory transactions (i.e., higher internal clock frequency) at the expense of reducing memory capacity; (b) using higher bus clock frequency; and (c) using denser signal modulation. The latter two are enabled by placing the memory modules very close to the compute units, which is done to limit the impacts of parasitic capacitance and electromagnetic interference on data integrity.

High-bandwidth-memory (HBM) has been enabled by advances in packaging technologies, which has allowed stacking of memory modules on top of the compute units. The stacked memory and compute units are then connected through a silicon interposer. This allows a significantly wider bus, and therefore, increased bandwidth. The short distance between the memory and compute units is necessary to ensure data-integrity. HBM has delivered significant bandwidth to GPUs. Recently, it has been used on CPUs and other hardware as well.

The traditional Von Neumann architecture creates a bandwidth bottleneck by separating memory and compute units. Near-memory processing alleviates this bottleneck, by splitting memory and compute units into smaller parts, and putting them closer to each other. Processing-in-memory is able to reduce the data movement by performing simple operations inside the memory cell, instead of sending the data into the processors. This results in significant energy savings. However, currently, it can only handle basic computations.

Flash memory can store data without continuous supply of power, making it energy-efficient. Compared to DRAM, it costs less, and has more density. Recently, it has been used to augment DRAM, resulting in significant increase in memory capacity on targeted CPUs. This trend is expected to continue, and will benefit data-intensive applications.

Inter-device communication

Communication remains the primary bottleneck for many workloads. When an application is small enough to fit within a single cluster node, within-the-node (intra-node) communication is often the main bottleneck. A notable example is communication between a microprocessor and its off-chip memory (Yazdanbakhsh et al., 2016), or between CPU, GPU and accelerator components in a heterogeneous system. For large-scale applications, where multiple cluster nodes have to work in tandem to provide scaling, between-the-node (inter-node) communication often constitutes the bottleneck, and could incur a significant cost in terms of performance degradation and energy consumption (Keckler et al., 2011; Kestor et al., 2013). Algorithm developers should be aware of these challenges and how future trends look like, as using a different algorithm can sometimes alleviate these bottlenecks. In what follows, we review on-chip, intra-node and inter-node communication technologies.

On-chip communication

SoCs comprise a diverse set of components, each implementing different functions[100], that need to communicate with each other. Below we delineate the two main types of on-chip communication architectures that have been used over the years, namely bus-based and Network-on-Chip (NoC). We also discuss the software support associated with orchestrating the on-chip communication between different components in an SoC.

Bus architectures. In the past, the most dominant architecture used for on-chip communication was bus based. A bus is a shared channel responsible for the communication between different components inside a chip. There are two types of such components in a system; masters and slaves. Master units (e.g., processor cores, DMA engines etc.) issue transactions, by sending out requests to the bus. Slave units, such as memory components, IO peripherals etc., receive requests issued by masters and respond with data when ready. Bus communication in an SoC is realized using an interface protocol. The interface is a collection of pins and signals that represent addresses, data and control information. Apart from the set of signals constituting the bus, the bus architecture is defined by several other defining characteristics, as elaborated below:

Physical structure: The physical structure refers to the hardware logic used to select which master/slave gets access to the bus. Three main implementations exist that use (i) tri-state buffers, (ii) AND/OR gates or (iii) multiplexers (MUX)[101].

Clocking. Defines whether or not a clock signal is part of the interface. If yes, the bus is called synchronous and every stage of a transaction occurs at a different clock cycle. If not, the bus is called asynchronous and it requires additional signals for synchronization.

Decoding and arbitration. Decoding logic is used to select the correct destination component in a transaction.

Arbitration logic is required when multiple masters try to access bus resources simultaneously. As the shared bus can handle a single transaction a time, the arbiter will grant bus access to only one master and the others will have to wait. Decoding and arbitration logic can be either centralized or distributed.

Data transfer modes. The simplest mode is the single non-pipelined transfer, where each transaction has to be fully processed before the next one is initiated. This requires the minimum amount of resources and wiring, but yields the lowest performance. Pipelined transfers increase performance by overlapping stages of subsequent transactions in time, thus allowing a transaction to be initiated while the previous is still processed. More sophisticated transfer modes that can further increase performance are the burst[102], split[103] and out-of-order (OOO)[104] transfers, among others.

Several standards for bus-based communication have been proposed over the years (Pasricha and Dutt, 2010). These standards are necessary, in order to seamlessly integrate the different components of an SoC with the bus. Some of the most popular standards are shown in Table 4, along with some of their main bus characteristics[105].

Bus-based architectures can be arranged using a variety of topology structures, which directly affect the cost, area, complexity, power and bandwidth, as shown in Figure 4. The simplest topology is that of the single shared bus, where all components in a system are connected to. The single shared bus used to be sufficient for simple SoCs with only a few components, but it fails to scale as the number of components increases. Examples of alternative topologies that aim to improve scalability and increase the system's bandwidth are the hierarchical, the ring and the split bus topologies. Another bus structure is that of a full crossbar (or full bus matrix), where every master component is connected to every slave component via dedicated buses. Due to the excessive area and power cost of this topology, partial crossbar (or partial bus matrix) topologies are often used instead.

Network-on-chip (NoC) architectures. Despite offering simplicity and a low area overhead, bus-based architectures do not scale well as the number of cores and processing units on the chip keeps increasing, due to their inherent shared nature. To address this issue, modern devices with heterogeneous cores employ a Network-on-chip (NoC) architecture to overcome the scalability issues of bus-based architectures (Alimi et al., 2021; Amin et al., 2020; Benini and De Micheli, 2002; Kundu, 2014). NoCs use packets to route data from source to destination components by using a network fabric that consists of switches, routers and interconnect links. In bus-based architectures, very long wires are often used to connect distant components, which can lead to long wire delays. Instead, NoCs address the wire delay problem, by cutting down the communication links and inserting routers in between, in a more structured manner. All links can transfer data simultaneously and independently, allowing for much higher on-chip bandwidth.

There are several factors that define an NoC architecture, mainly the network topology, the switching strategies and the routing algorithms (Pasricha and Dutt, 2010). The network topology specifies how the components, the switches and the links are laid out and connected to each other. There are three main categories: (a) direct network topologies[106], where each node in the network is directly connected to a subset of the other nodes; (b) indirect networks[107], where nodes are only connected to switches and switches are directly connected to subsequent switches; and (c) irregular networks that can use a mixture of NoC and bus-based topologies. Figure 5 shows two examples of direct NoC topologies, namely the mesh and the torus, and one example of an indirect NoC topology, the butterfly. The switching strategies determine how the data (packets) flow through the routers in the network. Two main strategies exist, circuit-switching and packet-switching[108]. Finally, the routing algorithms define what paths are chosen in the network for the communication between a source and a destination. The efficiency of the routing algorithms is a critical factor in the NoC's attainable performance as they directly

Table 4. On-chip synchronous bus communication standards.

		Modes[b]					Structure[d]		
Standard name[a]	Year	P	B	S	O	Data width[c]	M	A	T
ARM AMBA 2.0	1999	✔	✔	✔	—	8–1024	✔	—	—
ARM AMBA 3.0	2003	✔	✔	✔	✔	8–1024	✔	—	—
IBM coreconnect	2000	✔	✔	✔	—	32–256	✔	✔	—
STMicroelectronics STBus	2002	✔	✔	✔	✔	8–256	✔	—	—
Altera avalon	2006	✔	✔	✔	—	8–1024	✔	✔	✔
OpenCores wishbone	1999	✔	✔	—	—	8–64	✔	—	✔

[a]Each standard further defines several sub-standards, depending on performance requirements. Each sub-standard may support different transfer modes and data widths, among others.
[b]Supported modes include P: Pipelined, B: Burst, S: Split, and O: OOO. See the explanation of these modes in *Bus architectures*.
[c]Data width can be any power-of-two value within the indicated range and it refers to the data size of each transaction.
[d]Physical structure includes M: Multiplexer, A: AND/OR gates, and T: Tri-state buffer.

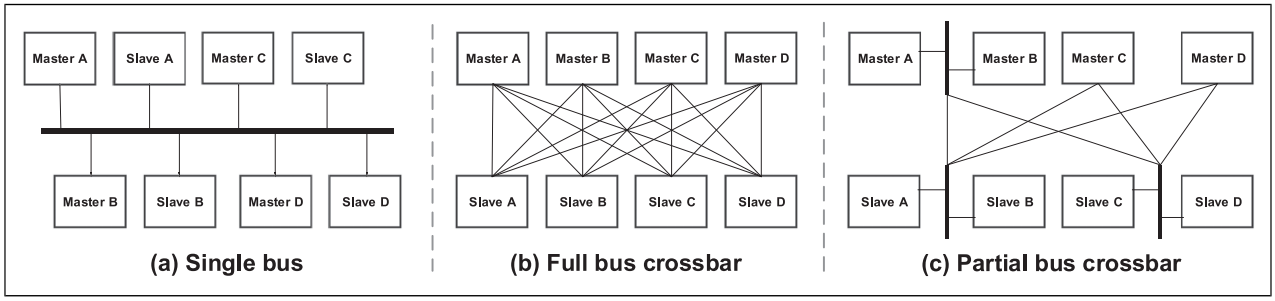

Figure 4. Examples of on-chip bus topologies (Pasricha and Dutt, 2010): (a) The single bus is the simplest and cheapest on-chip bus topology. All masters share the bus as the communication channel for all transactions with the system's slaves and only one master can have access at a time. The performance of the single bus does not scale with the number of components, due to increased traffic and congestion, where arbitration can lead to the starvation of some masters. (b) In the full bus crossbar, a dedicated bus is used for each possible master-slave connection. This corresponds to the highest possible system performance, as it maximizes the theoretical onchip communication bandwidth. Arbitration logic is now required for each slave, rather than for each bus. However, it becomes prohibitively expensive as the number of system components increases, adversely affecting area, cost, power consumption and routing complexity. (c) The partial bus crossbar is a mixture of the single bus and full bus crossbar. Essentially, it trades off the performance of the full bus crossbar for a reduction in area, cost and power consumption.

affect network contention and, thus, the system-wide communication bandwidth.[109]

Software for on-chip communication. As SoCs are typically built around a microprocessor (CPU), on-chip communication is usually orchestrated by the CPU using device drivers, when in the presence of an operating system. Device drivers are responsible for mapping the memory space of each of the SoC components to the CPU's memory space. Driver software is typically written in C/C++ or assembly code and is inherently complicated. Some works propose microarchitecture extensions that bypass driver complexity and overheads (Asri et al., 2020) or high-level software directives (Sommer et al., 2017; Tsog et al., 2021) that directly offload workloads to the accelerators by simply annotating a software code's candidate region with special instructions and intrinsics. In any case, high-level tools with easy-to-use APIs are often provided by vendors to simplify the communication for the end-user, such as CUDA (Buck, 2007), ROCm (Sun et al., 2018) and Vitis. Thus, to initiate a transaction, the user simply needs to define its size and data type, as well as the source and the destination.

Intra-node communication

A compute node typically comprises a general-purpose microprocessor, its off-chip memory, and its peripherals. The peripherals may include hardware accelerators, network communication interface[110], and non-volatile memory. The interface between the microprocessor and its off-chip memory has

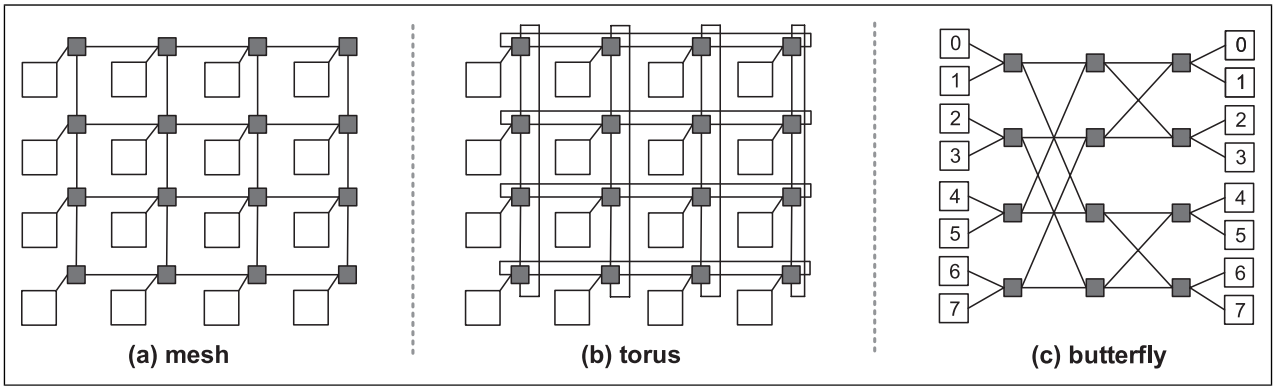

Figure 5. Examples of network-on-chip topologies (Pasricha and Dutt, 2010): (a) This is an example of a direct network topology, where the components are laid out in a 2D mesh. Each component (node) in the network has direct connections to its neighboring (left, right, top, bottom) nodes. The aggregate communication bandwidth of the system scales with the mesh size. (b) The torus is an extension of the mesh that includes additional connections between distant components (top-bottom, leftmost-rightmost). This increases bandwidth at the cost of area and wiring complexity. (c) The butterfly represents an indirect network topology where nodes exclusively connect to switches, and these switches establish point-to-point connections with other switches. Although depicted separately, the source and destination nodes are the same.

been reviewed in the DRAM section. In this part, we review the interface between the microprocessor and its peripherals.

Historical trends and key technologies. Table 5 shows historical trends of commonly-used interfaces since the 1980s, including modern interfaces. Key factors that impact the design of modern and next-generation, high-bandwidth interfaces include.

Data rate, and bus clock frequency. Copper-based wires are popular for very short-to medium-distance communication links due to their low manufacturing cost and high reliability. However, they already have met their technological limits (*Part I: Wiring, connectivity, and signal integrity* (Hanindhito et al., 2026)). Increasing the bus clock frequency, which is required to achieve higher bandwidth in synchronous serial interfaces[111] is challenging due to data integrity issues, especially over a long distance[112] (Kumar et al., 2023). Using other materials, such as platinum-group metals, or using optical signals (*Part I: Optical interconnects* (Hanindhito et al., 2026)), are among viable options for increasing the communication bandwidth.

Bus encoding and bus overhead. Modulation and encoding are needed before data is transmitted (*Part I: Signal coding and modulation* (Hanindhito et al., 2026)). Although denser modulation shapes promise higher bandwidth, they have higher bit error rates[113], and necessitate using a more advanced error correction mechanism[114]. Accordingly, a more advanced physical layer implementation (SerDes) to support a more advanced error correction mechanism (Aziz et al., 2011; De Paulis et al., 2022; Drenski and Rasmussen, 2018; Yue and Shekhar, 2022), and a more efficient (block) coding is needed to enable higher data rates. This adds more transistors to the microprocessor, and increases power consumption (*Part I: Architecture of communication interfaces* (Hanindhito et al., 2026)).

Bus width and routing complexity. Adding more serial lanes increases the communication bandwidth; however, this is expensive[115] (*Part I: Architecture of communication interfaces* (Hanindhito et al., 2026)). Many modern serial interfaces[116] utilize multiple serial lanes, where some of them permit a flexible configuration[117] for the number of lanes, based on bandwidth requirements.

Data transfer energy consumption. Longer-distance data transmission entails more energy consumption[118] (Keckler et al., 2011; Kestor et al., 2013). High-frequency signals consume more power, due to the effect of parasitic capacitance at higher frequencies (Tran, 2023). Lastly, a more advanced physical layer implementation (SerDes) consumes more power, which must be accounted for in the total power budget of the chip (Abdennadher et al., 2020; Rashdan et al., 2009, 2020).

Standardization and industry adoption. Communication interfaces are used to connect various devices. Their adoption by multiple hardware vendors enables compatibility and interoperability among different devices when each device is provided by a different vendor. The Common Electrical I/O (CEI) (*Part I: Architecture of communication interfaces* (Hanindhito et al., 2026)) is an effort to achieve standardization. Accordingly, multi-standard, multi-mode physical interfaces (SerDes) have been developed, which can then be used to implement various communication protocols (Chattopadhyay et al., 2018; Lin et al., 2021; Nishi et al., 2008; Roshan-Zamir et al., 2017; Vamvakos et al., 2012), reducing the development cost and time. An example of a widely adopted interface is PCI Express. On the other hand, proprietary interfaces, such as NVIDIA NVLink (Wei et al., 2023), and AMD InfinityFabric, are limited to devices that are provided by their respective vendors. The use of proprietary interfaces allows manufacturers to optimize their implementation to achieve more efficient interoperability across different devices from the same vendor. However, future computing systems will move toward more open and standardized interfaces, such as Open Compute Project (OCP) (Frachtenberg, 2012), Compute Express Link (CXL) (Sharma, 2022c; 2023a), and Universal Chiplet Interconnect Express (UCIe) (Das Sharma et al., 2022).

Connection topology. Due to the availability of various communication interfaces, a computer system often uses different types of interfaces, along with different connection topologies (Kong et al., 2023). For instance, a GPU-accelerated compute node may use AMD Infinity Fabric (interface) for inter-socket CPU communication, PCI Express (interface) for CPU-to-GPU, and CPU-to-NVMe[119] communication, and NVIDIA NVLink (interface) for GPU-to-GPU communication. Applications and communication libraries must be able to identify each communication interface, along with the connection topology between them (Amaral et al., 2017; Chu et al., 2020; Li et al., 2020; Muthukrishnan et al., 2021). This enables them to choose the most efficient interface for exchanging data between different devices.

Coherency. PCI Express has been widely used as a communication interface between microprocessors and various devices (Mayhew and Krishnan, 2003). However, PCI Express does not have built-in support for maintaining cache and memory coherency (Lutz et al., 2020; Stuecheli et al., 2015). Therefore, the coherency must be handled at the software level[120]. With the proliferation of hardware accelerators (*Part I: Hardware accelerators* (Hanindhito et al., 2026)) and heterogeneous computing, efficient communication interfaces and protocols become more important, especially for resource sharing and memory management.

Compute express link (CXL). The Compute Express Link (CXL) is an open standard protocol, which provides a cache-

Table 5. Evolution of intra-node device-to-device communication link.

Standard name	Year	Duplex	Type	Interface width	Bandwidth[a]	Remarks
Industry Standard Architecture						
ISA	1981[b]	Half	Parallel	8-bit	8 MB/s	8-bit wide with system clock 8 MHz
ISA	1984[b]	Half	Parallel	16-bit	16 MB/s	16-bit wide with system clock 8 MHz
Extended ISA	1988[b]	Half	Parallel	32-bit	33 MB/s	32-bit wide with system clock 8 MHz
Peripheral Component Interconnect						
PCI 1.0	1992[b]	Half	Parallel	32-bit	133 MB/s	32-bit wide with bus clock 33 MHz
PCI 2.0	1995[b]	Half	Parallel	32-bit	533 MB/s	64-bit wide with bus clock 66 MHz
PCI-X 1.0	1998[b]	Half	Parallel	64-bit	1066 MB/s	64-bit wide with bus clock 133 MHz
PCI-X 2.0	2003[b]	Half	Parallel	64-bit	4266 MB/s	64-bit wide with bus clock 533 MHz
Accelerated Graphics Port						
AGP 1.0	1996[b]	Half	Parallel	32-bit	533 MB/s	32-bit wide with 2X speed
AGP 2.0	1998[b]	Half	Parallel	32-bit	1066 MB/s	32-bit wide with 4X speed
AGP 3.0	2002[b]	Half	Parallel	32-bit	2133 MB/s	32-bit wide with 8X speed
PCI Express						
PCIe 1.0	2003[b]	Full	Serial	16 lanes/link	8 GB/s	Configurable 1/2/4/8/16 lanes per link
PCIe 2.0	2007[b]	Full	Serial	16 lanes/link	16 GB/s	Configurable 1/2/4/8/16 lanes per link
PCIe 3.0	2010[b]	Full	Serial	16 lanes/link	31.5 GB/s	Configurable 1/2/4/8/16 lanes per link
PCIe 4.0	2017[b]	Full	Serial	16 lanes/link	63.0 GB/s	Configurable 1/2/4/8/16 lanes per link
PCIe 5.0	2019[b]	Full	Serial	16 lanes/link	126 GB/s	Configurable 1/2/4/8/16/32 lanes per link
PCIe 6.0	2022[b]	Full	Serial	16 lanes/link	242 GB/s	Configurable 1/2/4/8/16/32 lanes per link
PCIe 7.0	2025[b]	Full	Serial	16 lanes/link	484 GB/s	Configurable 1/2/4/8/16/32 lanes per link
HyperTransport Technology[c]						
HTT 1.0	2003	Full	Serial	16 lanes/link	6.4 GB/s	AMD SledgeHammer CPU; 1.67 GT/s[a]
HTT 2.0	2004	Full	Serial	16 lanes/link	8 GB/s	AMD Athens CPU; 2 GT/s[a]
HTT 3.0	2008	Full	Serial	16 lanes/link	16 GB/s	AMD Budapest CPU; 4 GT/s[a]
HTT 3.1	2010	Full	Serial	16 lanes/link	25.6 GB/s	AMD Magny-Cours CPU; 6.4 GT/s[a]
AMD Infinity Fabric Inter-socket[d]						
IFIS 1.0 CPU-CPU	2017	Full	Serial	16 lanes/link	42.68 GB/s	AMD EPYC Naples CPU; 10.67 GT/s[a]
IFIS 2.0 CPU-CPU	2019	Full	Serial	16 lanes/link	72 GB/s	AMD EPYC Rome CPU; 18 GT/s[a]
IFIS 2.0 GPU-GPU	2020	Full	Serial	16 lanes/link	92 GB/s	AMD Instinct CDNA GPU; 23 GT/s[a]
IFIS 3.0 GPU-GPU	2021	Full	Serial	16 lanes/link	100 GB/s	AMD Instinct CDNA2 GPU; 25 GT/s[a]
IFIS 3.0 CPU-CPU	2022	Full	Serial	16 lanes/link	128 GB/s	AMD EPYC Genoa CPU; 32 GT/s[a]
IFIS 3.0 GPU-GPU	2024	Full	Serial	16 lanes/link	128 GB/s	AMD Instinct CDNA3 GPU; 32 GT/s[a]
Intel Quick Path Interconnect						
QPI	2009	Full	Serial	20 lanes/link	25.6 GB/s	Intel Xeon Nehalem EP CPU; 6.4 GT/s[a]
QPI	2012	Full	Serial	20 lanes/link	32 GB/s	Intel Xeon SandyBridge EP CPU; 8 GT/s[a]
QPI	2014	Full	Serial	20 lanes/link	38.4 GB/s	Intel Xeon Haswell EP CPU; 9.6 GT/s[a]
Intel Ultra Path Interconnect						
UPI	2017	Full	Serial	20 lanes/link	41.6 GB/s	Intel Xeon Skylake SP CPU; 10.4 GT/s[a]
UPI	2021	Full	Serial	20 lanes/link	44.8 GB/s	Intel Xeon Icelake SP CPU; 11.2 GT/s[a]
UPI 2.0	2023	Full	Serial	24 lanes/link	>64 GB/s	Intel Xeon Sapphire Rapids SP CPU; 16 GT/s[a]
UPI 2.0	2024	Full	Serial	24 lanes/link	>80 GB/s	Intel Xeon Emerald Rapids SP CPU; 20 GT/s[a]
NVIDIA NVLink[e]						
NVLink 1.0	2016	Full	Serial	8 lanes/link	40 GB/s	NVIDIA Pascal GPU; 20 GT/s[a]
NVLink 2.0	2017	Full	Serial	8 lanes/link	50 GB/s	NVIDIA Volta GPU; 25 GT/s[a]
NVLink 3.0	2020	Full	Serial	4 lanes/link	50 GB/s	NVIDIA Ampere GPU; 50 GT/s[a]
NVLink 4.0	2022	Full	Serial	2 lanes/link	50 GB/s	NVIDIA Hopper GPU; 100 GT/s[a]
NVLink 5.0	2025	Full	Serial	2 lanes/link	100 GB/s	NVIDIA Blackwell GPU; 200 GT/s[a]

[a]The GT/s (Giga transfer per second) measures the performance of each sublink of the protocol, including protocol overheads. The bandwidth shown is bidirectional raw, derived from GT/s. Effective bandwidth will be lower.
[b]Year shown is when the standard was released, not when the first product adopting that standard hit the market.
[c]Although HTT can have up to 32 lanes (32-bit) per HTT link, in their implementation, AMD microprocessors use multiple HTT links with only 16 lanes (16-bit) per HTT link in each direction.

[d]AMD Infinity Fabric is the successor of HTT and is used widely in AMD products, including CPUs and GPUs. Infinity Fabric on Package (IFOP) is used for inter-chiplet communication on the same package while Infinity Fabric Inter-socket (IFIS) is used for inter-socket communication (CPU-CPU, GPU-GPU, CPU-GPU) (Munger et al., 2023; Suggs et al., 2020).
[e]NVIDIA Tesla P100, Tesla V100, A100, and H100 have 4, 6, 12, and 18 NVLink links per GPU, respectively, with aggregate bidirectional bandwidth of 160, 300, 600, and 900 GB/s, respectively.

and memory-coherent communication interface between microprocessors, memory, and hardware accelerators (Sharma, 2022c; 2023a). It uses PCI Express as its physical layer, with added coherency protocol between microprocessors and the attached devices, which allows simplified resource sharing (Cabrera et al., 2022; Jung, 2022). Therefore, it can limit the involvement of users in managing the memory of hardware accelerators. CXL aims to improve communication performance across multiple devices, simplify the software stack, and lower the overall system cost.

The first generation of CXL, CXL 1.0/1.1 (2019), introduced three types of protocols: (a) CXL.io, which has the same functionality as the PCIe protocol; (b) CXL.cache, which allows hardware accelerators to efficiently access host microprocessor's memory; and (c) CLX.mem, which permits host microprocessors to access device-attached memory. These protocols enable the coherent sharing of memory resources between the host microprocessor and different devices[121]. The second and third generation of CXL (2020, 2022, respectively) primarily deal with inter-node communication.

Software for intra-node communication. Communication between different devices on a node, e.g. CPU-CPU, CPU-GPU, GPU-GPU etc. can be realized using interfaces, such as PCIe, Ultra Path, NVLink. In software, intra-node communication between CPUs was historically first managed by using pthread, a low-level, shared memory execution model compliant with most operating systems. To improve productivity, libraries like OpenMP were developed that offer easy-to-use, high-level APIs. By contrast, inter-node communication was handled using Message Passing Interface (MPI) libraries, such as OpenMPI (Gabriel et al., 2004) and MVAPICH (Panda et al., 2021). As modern nodes comprising both CPUs and GPUs grew in popularity, MPI libraries began to be utilized for intra-node communication as well, due to extensions that allowed CPU-GPU communication. Some of these libraries are also GPU-aware[122], allowing efficient CPU-GPU and GPU-GPU communication. These libraries enable direct GPU-GPU communication, without the requirement of staging data through the host CPU memory.

Off-chip accelerator devices, such as FPGAs and ASICs are typically connected to a host CPU device via a PCIe IO interface (Firoozshahian et al., 2023; Jouppi et al., 2017, 2023; Prabhakar et al., 2022). Similarly to on-chip communication, the CPU can orchestrate transactions with the FPGA or the ASIC by using device drivers. Again, the use of high-level APIs can greatly boost productivity for the end user. Direct off-chip communication between FPGAs or ASICs requires implementation of low-level hardware mechanisms, which are more complicated in nature. A sophisticated software stack support is required to abstract this complexity from the end user.

Inter-node communication

Only a limited amount of processing power, memory, and storage, can be placed on a single compute node[123]. Power delivery and thermal dissipation are among the main limiting factors[124]. Moreover, the pin scarcity of CPUs limits the number of memory channels and off-chip interconnection links, which, in turn, limits how much memory, storage, and accelerator cards a single compute node can have. To be able to handle increasingly larger problem sizes, multiple compute nodes are typically utilized. The interconnection between compute nodes is the weakest link in the inter-device communication. Inter-node communication often limits the overall performance (Rumley et al., 2020), and consumes considerable amounts of energy (Georgakoudis et al., 2019), especially for workloads that have excessive inter-node data movement.

Technologies and standards. To provide high-bandwidth, low-latency connection between compute nodes, different technologies (Table 6) are used by large-scale clusters around the world (Li et al., 2022; Lu et al., 2022). The most commonly-used technologies are Ethernet and InfiniBand.

Ethernet interconnects have been popular in large-scale clusters due to their lower cost, e.g. multi-gigabit Ethernet interconnects, with 100 Gb/s bandwidth and beyond. In June 2023, 45.4% of clusters in the Top500 list used Ethernet as their inter-node communication, with 37.8% of them supporting 100 Gb/s. None of the clusters in the Top50 list use multi-gigabit Ethernet, as most[125] of them use InfiniBand.

InfiniBand has been adopted by 40% of the clusters in the Top500 list, with 32.5% of them supporting 200 Gb/s High Data Rate (HDR), and 30% of them supporting 100 Gb/s Extended Data Rate (EDR) and HDR combined. Other clusters in the Top500 list use 100 Gb/s Slingshot-10 (3.4%), 200 Gb/s Slingshot-11 (3%), and 100 Gb/s Omnipath (7%).

Herein, we focus on InfiniBand, due to its wide adoption, and availability of public documentation. Other interconnect technologies will most likely follow the roadmap of InfiniBand due to competition. Advances in physical layer implementation, modulation, and coding schemes will play a key role in realizing the bandwidth of different interconnect technologies[126]. Table 7 summarizes historical trend and future roadmap of InfiniBand, as detailed below.

Table 6. HPC inter-node interconnection technology landscape.

Technology	Ethernet	Infiniband	Omni-path	Slingshot	Aries	TofuD	BXI
Manufacturer	Many	NVIDIA/Mellanox	Intel/Cornelis	Cray/HPE	Cray/HPE	Fujitsu	Atos
Bandwidth (Gb/s)[a]	400	400	100	200	40	56	100
Latency (μS)	1–30[c]	<1	<1	<2	1	0.5–1	<1
Topology	Various	Fat-tree	Fat-tree	Dragonfly	Dragonfly	Torus	Various
Congestion control	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RDMA[b] support	Yes[d]	Yes	Yes	Yes	Yes	Yes	Yes

[a]The highest unidirectional bandwidth achieved by each technology as of 2023.
[b]Remote Direct Memory Access (RDMA) is a networking technology that enables data exchange between the memory of two compute nodes without involving CPUs of the host or receiver, or the operating system. The Network Interface Card (NIC) performs the exchange, resulting in reduced latency and communication overheads. For instance, RDMA reduces communication overheads significantly when GPUs on different nodes exchange data.
[c]RDMA over Converged Ethernet (RoCE) provides significantly low latency for Ethernet (around 1μS) while standard Ethernet protocol has higher latency.
[d]RDMA is only supported with RDMA over Converged Ethernet (RoCE) technology. Standard Ethernet protocol does not have RDMA support.

SDR. InfiniBand emerged in 1999 as InfiniBand Trade Association (IBTA), from the merger of two competing technologies: Future I/O, and Next Generation I/O (NGIO). Mellanox, which was founded in 1999 to develop NGIO, shipped its first InfiniBand product, InfiniBridge, in 2002. InfiniBridge is an integrated host channel adapter[127] (HCA) and switch[128], capable of providing eight 1X InfiniBand lanes at 2.5 Gb/s; or two ports, each having 4X InfiniBand lanes, enabling a total bandwidth of 10 Gb/s, known as Single Data Rate (SDR) (Deirling, 2001; Eddington, 2002).

Either copper or optical cables[129] can be used to connect two InfiniBand points (e.g., HCA to a switch). The PCI interface (Table 5) turned out to be the limiting factor on how much bandwidth the HCA can achieve. Therefore, in 2003, Mellanox introduced InfiniHost with PCI-X interface for the HCA chip, along with a dedicated switching chip, called InfiniScale. InfiniScale had eight 4X InfiniBand ports, with improved scalability and switching capacity for larger clusters.

DDR. As serial clock rates advanced to 5 GHz, the Double Data Rate (DDR) version of InfiniBand was introduced in 2005, with InfiniHost III HCA chip, and InfiniScale III switching chip.

The InfiniHost III HCA was the first InfiniBand HCA to use PCI Express interface to provide more bandwidth between host CPU and HCA. The InfiniScale III was the first switching chip to reach almost 1 Tb/s of switching capacity. DDR had twenty-four 4X InfiniBand ports, and enabled a total bandwidth of 20 Gb/s.

QDR. In 2008, Mellanox released its fourth-generation HCA, named ConnectX (Sur et al., 2007), which provided significantly lower latency, higher packet processing[130] performance, and higher bandwidth; it was first used in Quad Data Rate (QDR) InfiniBand. It features the Virtual Protocol Interconnect (VPI), which allows the HCA to be configured as either an Ethernet, or an InfiniBand card (Grant et al., 2009). This flexibility makes it easier for clusters to progressively adopt InfiniBand technologies, while keeping some of their less-demanding infrastructure[131] on Ethernet.

FDR. The VPI feature was also integrated into the switching chip, and was called SwitchX (2011). SwitchX uses a 14 GHz serial clock rate, and the new and more efficient 64b/66b block coding technique[132] (*Part I: Signal coding and modulation* (Hanindhito et al., 2026)), resulting in 4 Tb/s of switching capacity; along with ConnectX-3 HCA chip, it constitutes the Fourteen Data Rate (FDR) InfiniBand.

EDR. In 2015, Mellanox separated the switching chips: Switch-IB for 100 Gb/s Extended Data Rate (EDR) InfiniBand; and Spectrum for 100 Gb/s Ethernet[133]. Nevertheless, the ConnectX HCAs was still used to support both InfiniBand and Ethernet, using VPI.

HDR. The High Data Rate (HDR) InfiniBand switched from NRZ modulation to PAM-4 modulation, in order to push more data into the cables. HDR achieved 200 Gb/s bandwidth on 4X InfiniBand links, with a 51.6 GHz serial clock rate. The Quantum switching (2018) chip provides 16 Tb/s of switching capacity, with forty 200 Gb/s InfiniBand ports.

NDR. The Next Data Rate (NDR) was released in 2022, with a 112 GHz serial clock utilizing PAM-4 modulation and 256b/257b block coding.

XDR. The Extreme Data Rate (XDR) was released in 2024 with 224 GHz SerDes (Chan et al., 2022).

Going forward, adoption of inter-node communication technologies will depend on how a cluster is intended to be utilized. Multi-gigabit (e.g., 50 Gb/s to 200 Gb/s) Ethernet will likely see rapid deployment on mainstream enterprise clusters[134], as they are cost-effective, and easy to implement. HPC clusters will likely use 100 Gb/s to 200 Gb/s Ethernet, InfiniBand, or their equivalent technologies for CPU-based

Table 7. InfiniBand history and future roadmap.

Year	2001	2002	2005	2008	2011	2015	2018	2021	2024	2027
Link specification										
Standard	SDR	SDR	DDR	QDR	FDR	EDR	HDR	NDR	XDR	GDR
Block coding	8b/10b	8b/10b	8b/10b	8b/10b	64b/66b	64/66b	64b/66b	256b/257b	TBD	TBD
Modulation	NRZ	NRZ	NRZ	NRZ	NRZ	NRZ	PAM-4	PAM-4	TBD	TBD
SerDes rate (GHz)	2.5	2.5	5	10	14	25.8	51.6	112	224	448
Link speed (Gb/s)	2	2	4	8	13.6	25	50	100	200	400
Bandwidth (GB/s)	1	1	2	4	6.82	12.5	25	50	100	200
Host channel adapter										
Architecture[a]	IB	IH	IH III	CX CX-2	CX-3 C-IB	CX-4 CX-5	CX-6	CX-7	CX-8	TBD
Host interface	PCI 2.0	PCI-X 1.0	PCIe 1.0 x8	PCIe 2.0 x8	PCIe 3.0 x8 PCIe 3.0 x16	PCIe 3.0 x16 PCIe 4.0 x16	PCIe 4.0 x16	PCIe 5.0 x16 PCIe 5.0 x32	PCIe 6.0 x16 PCIe 6.0 x32	TBD
# Ports (Gbps)	2 (10)	2 (10)	2 (20)	2 (40)	2 (56)	2 (100)	2 (200)	2 (400)	2 (800)	2 (1600)
Switch infrastructure										
Architecture[b]	IB	IS	IS III	IS IV	SX SX-2	S-IB S-IB2	Q	Q-2	Q-X800	TBD
# SerDes (Gb/s)	8 (2.5)	32 (2.5)	96 (5)	144 (10)	144 (14)	144 (25)	160 (51.6)	256 (112)	576 (224)	TBD (448)
# Ports (Gb/s)	2 (10)	8 (10)	24 (20)	36 (40)	36 (56)	36 (100)	40 (200)	64 (400)	144 (800)	TBD (1600)
Switching capacity	45 Gb/s	160 Gb/s	960 Gb/s	2.88 Tb/s	4 Tb/s	7.2 Tb/s	16 Tb/s	51.2 Tb/s	225 Tb/s	TBD

[a]Abbreviation for Host Channel Adapter architectures: IB: InfiniBridge, IH: InfiniHost, CX: ConnectX, and C-IB: Connect-IB.
[b]Abbreviation for Switch architectures: IB: InfiniBridge, IS: InfiniScale, SX: SwitchX, S-IB: Switch-IB, and Q: Quantum.

clusters, and 400 Gb/s InfiniBand or their equivalent technologies for GPU-based clusters. Cloud clusters will likely aim for 100 Gb/s to 400 Gb/s interconnection technologies, based on their workloads[135]. Bandwidth-hungry AI- or ML-clusters will likely need an even higher bandwidth, possibly in the range of 800 Gb/s or beyond; this will make them early adopters of advanced interconnection technologies.

Network interface card (NIC) and data processing unit (DPU). The basic functionality of a network interface card (NIC) is to allow each node to communicate with other nodes through a particular networking standard[136]. Basic NICs can only perform lower-level network protocol functions, while the CPU handles the higher-level protocols. Offload NIC is a hardware accelerator that performs higher-level network protocol functions[137] (Maccabe et al., 2002; Shivam and Chase, 2003), and thus, removes the burden from the host CPU. The offloading strategy becomes more important for higher bandwidth networks (e.g., 1 Gb/s and 10 Gb/s Ethernet), since the network protocol overheads become larger, and can consume significant cycles from the host CPU. However, this offloading mechanism is often limited to the TCP/IP layer or below[138] (Pismenny et al., 2021a). Therefore, the host CPU must still handle the application layer and above[139], such as data encryption and decryption (Pismenny et al., 2021b; Sabin and Rashti, 2015), data compression and decompression (Li et al., 2023), storage protocols (Kim et al., 2021a; Zhang et al., 2023), remote procedure call protocols, and key-value store protocols.

With network bandwidth at or exceeding hundreds of Gb/s, the overhead of network applications becomes more significant. SmartNICs extend the capabilities of Offload NICs, by integrating a general-purpose CPU into the NIC (Figure 6). This offers programmable offloading capabilities for network applications[140] (Qiu et al., 2020). Data processing units (DPUs) extend SmartNICs, by providing even more compute capabilities and memory capacity. Accordingly, DPUs enable running more complex data-intensive applications in NIC, such as using neural networks to detect security threats on traffic flows throughout the computing infrastructure (Tasdemir et al., 2023), data encryption, and traffic management and scheduling[141]. To enable the development of custom computing pipelines that are tailored for specific needs, a reconfigurable DPU could be used, which is a DPU with an integrated FPGA (Caulfield et al., 2018; Dastidar, 2023; Trivedi and Brunella, 2023).

We remark that a robust and easy-to-use software stack is also important for the end users to optimally harness the performance and efficiency that is offered by DPUs. We highlight a few DPU products next:

NVIDIA BlueField-2 DPU. It combines ConnectX-6 with 8-core ARM A72 CPUs, to offload critical networking, storage, and security tasks from the host CPUs. BlueField-2X DPU adds a GPU[142] into the same PCB that hosts the ConnectX-6 NIC and the ARM CPU, in order to run AI or ML analytics tasks on the data while the data is being transmitted. This capability, for instance, could benefit the telecommunication industry.

BlueField-3 DPU. It improved its predecessor by combining ConnectX-7 with 16-core ARM A78 CPUs, in order to improve its throughput and processing power.

Other products. Other vendors followed the same approach by introducing their own DPUs, such as AMD, with their Pensando DPU, and Intel, with their Infrastructure Processing Unit (IPU).

Network topologies. While the bandwidth of inter-node communication is increasing, connecting all nodes within a large cluster is challenging. Even with the most advanced switches, with Terabytes per second of switching capacity, there will not be enough ports to directly connect all nodes (Maniotis et al., 2020, 2022; Teh et al., 2022; Villar et al., 2013; Yan et al., 2021). Instead, multiple switches are used to provide connection between different parts of a cluster (Alizadeh and Edsall, 2013; Krause et al., 2018; Li et al., 2015b): leaf switches, to provide connection between nodes within the same rack; spine switches, to provide connection between racks; and sometimes super-spine switches, to provide connection between spine switches in extremely large clusters.

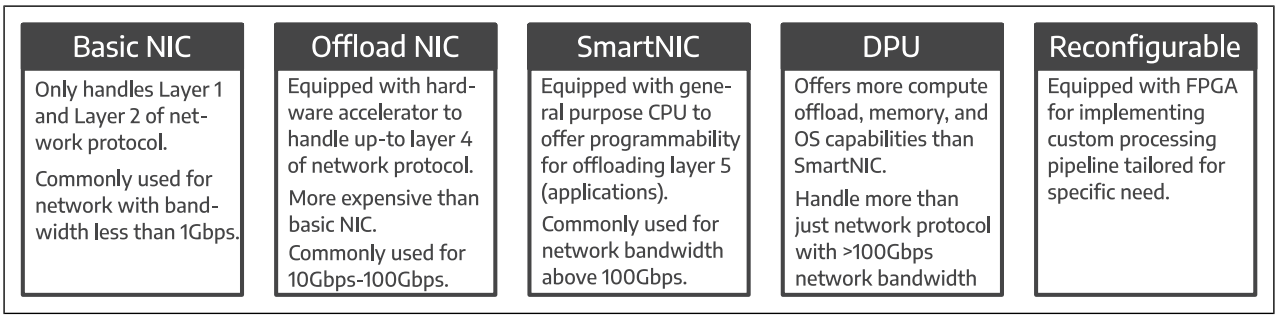

Figure 6. As the network bandwidth reaches hundreds of Gb/s, more functionalities are being added to the network interface card, mainly to reduce the network and application overhead from host CPU.

Each cluster may have a different inter-node interconnection topology, with different characteristics in terms of bandwidth and latency between nodes.

For instance, the Frontera cluster (2019) at the Texas Advanced Computing Center (TACC) uses two-level switches in a fat-tree topology. AI or ML clusters may use a more complex topology, with multi-rail InfiniBand per node[143]. Optimizing software and workloads when running on a specific cluster (e.g., by partitioning a workload appropriately across different nodes) may significantly impact the overall performance.

Disaggregated infrastructure. Data centers or HPC clusters have three main components in their compute racks (infrastructure): compute, storage, and network. The compute part is provided by a large number of servers interconnected to each other. The network part is responsible for providing high-bandwidth and low-latency connections to the servers, allowing them to work together. Each server may have a different configuration, essentially resembling a stand-alone computer, where they have their own processors, memory (e.g., DRAM), peripherals (e.g., GPUs), and storage. However, using local storage creates challenges for managing data across different servers. Accordingly, centralized storage is typically provisioned to provide parallel, scalable data access across all servers. This choice also eases replicating the storage for backup. Figure 7 illustrates different types of infrastructure for data centers and HPC clusters (Park et al., 2022).

Traditional infrastructure implements the three components as separate entities, often from different manufacturers. In this manner, servers that implement the compute are connected through a networking fabric[144]. On the other hand, storage is usually implemented by using proprietary hardware, which often comes with its own proprietary networking fabric[145]. These four components must be architected to work together as a cluster system. Deploying and maintaining these components, possibly from different manufacturers, is costly and inefficient (Haag, 2018).

Converged infrastructure is a pre-packaged and pre-validated compute, storage, and network, which is provided by a single manufacturer. While individual components can be manufactured by different vendors, they have been validated to work together (Garber, 2012). This guarantees their interoperability and reliability, enabling seamless integration and scalability. In this design approach, the storage system is commonly implemented with commodity hardware and typically shares the same network fabric for communication that the rest of the components are using. This eliminates the need for using a specialized storage fabric, and greatly simplifies deployment and maintenance of the infrastructure. However, it adds pressure to the network as it needs to handle both the compute and storage traffic.

The composable/disaggregated infrastructure (CDI) provides pooled resources[146] in physically separated units or racks that can be allocated dynamically based on specific needs of an application. Consequently, each server is no longer a complete compute system. In the extreme case, there may be entire racks with pooled CPUs versus pooled memory versus pooled discs. Accordingly, CDI allows resource allocation at finer granularity, which improves efficiency. However, interconnection between devices can be challenging, and requires further advancements in communication technologies[147], including silicon photonics (Michelogiannakis et al., 2023).

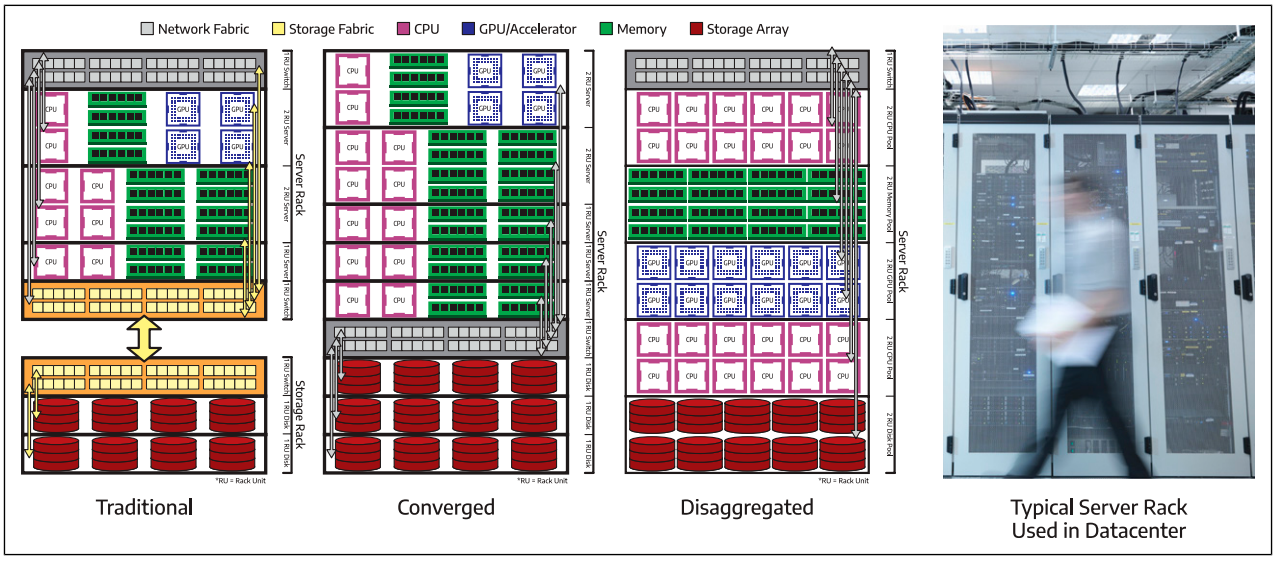

Figure 7. Examples of infrastructure (compute rack) in data centers and HPC clusters (Haag, 2018): (a) traditional, with separate compute, network, and storage entities (left); (b) converged, with pre-validated, packaged components (middle); and (c) disaggregated, with pooled resources (right). For comparison, an image of a representative cluster showing three compute racks is also included.

While CXL 1.0/1.1 focuses on interconnects on a single node (Sharma, 2019), subsequent versions of CXL, i.e. CXL 2.0 (Sharma, 2020a) and CXL 3.0 (Sharma, 2022b), focus on cluster-wide resource sharing[148]. These technologies make the implementation of future CDIs easier. With copper wires approaching their end of life when it comes to support low latency, high bandwidth, and long-distance communication, optical and silicon photonics will become key drivers in the future of cluster computing with CDI.

Shared infrastructure. Computing infrastructure is built differentially for HPC, AI/ML, and cloud computing, due to their unique requirements. For instance, advanced AI/ML applications typically run communication-intensive workloads. They exploit the latest hardware accelerators (e.g., GPUs), the most advanced network technologies, and interconnect topologies that are available today. HPC applications demand significant compute power. They use high-bandwidth, low-latency interconnects, although not always as demanding as AI/ML applications. On the other hand, cloud computing has low or moderate requirements, depending on (micro)-services they serve. Accordingly, AI/ML, HPC, and cloud infrastructure have been evolving independently for decades. Nevertheless, since the demand for computing is rapidly increasing, and the cost of building computing infrastructure is surging, designing infrastructure that supports a more diverse workload is becoming more desirable. For instance, by adopting the Everything-as-a-Service[149] model of cloud computing, Acceleration-as-a-Service (XaaS) (Hoefler et al., 2024) enables running HPC applications in a computing infrastructure that adopts cloud-computing principles (i.e., containerization). Accordingly, resources are allocated based on container types (e.g., HPC container, or cloud service container), ensuring better utilization and lowering the total cost of ownership for such computing infrastructure.

What comes next?

At the on-chip and in-package level, with trends of breaking large monolithic chips into smaller chiplets (*Part I: Advanced packaging technologies* (Hanindhito et al., 2026)) that are integrated in one package, communication between chiplets may limit the overall performance of chips for some applications (Chirkov and Wentzlaff, 2023). The Universal Chiplet Interconnect Express (UCIe) provides a milestone for open industry standards for future chips that use the chiplet packaging technology (Das Sharma et al., 2022; Sharma, 2023b).

Development of intra-node interconnects will continue according to the trend shown in Table 4, and will depend heavily on advances in SerDes (*Part I: Architecture of communication interfaces* (Hanindhito et al., 2026)) and silicon photonics technologies (Michelogiannakis et al., 2023), given that electrical signals on metal wires have hit their bandwidth limits. A key desire is development of intra-node interconnects that are as fast as the interconnect between chips and off-chip memory. This can especially be beneficial to workloads that deal with vast amounts of data, which require aggregate memory capacity across multiple devices.

Finally, inter-node interconnects will rely heavily on optical communication, which is a key enabler of disaggregated infrastructure. Similarly to intra-node interconnects, providing inter-node bandwidth that is on par with off-chip memory bandwidth is a key objective. Processing-in-network[150] will become more popular in communication-intensive workloads since it offloads data aggregation computations[151] to modern network switches and SmartNICs, which minimizes the amount of data that is needed for communication and thus lowers overhead for CPUs and GPUs.

System integration and heterogeneous computing

In order to keep up with modern computational and power requirements, systems comprising CPUs, GPUs and/or other accelerators are typically employed. In this section, we discuss challenges associated with deciding what type of system a given application should be run on and how its different components should be integrated.

While GPUs and other hardware accelerators offer much higher performance and energy efficiency for a certain class of applications, they are not meant to replace general-purpose microprocessors (CPUs). The CPUs are still needed for running operating systems and drivers (Chishiro et al., 2019; Stuecheli et al., 2015), orchestrating resources (e.g., memory, data) between devices (Bolchini et al., 2015; Hamano et al., 2009; Panneerselvam and Swift, 2016; Usui et al., 2016; Yogatama et al., 2022), and executing non-accelerated workloads[152] (Agosta et al., 2018; Baskaran et al., 2022; Marongiu et al., 2015; Tsog et al., 2021). The use of multiple such types of processing units in a well-orchestrated and coordinated way to provide improvements in performance and energy efficiency is called heterogeneous computing (Khokhar et al., 1993; Zahran, 2017), which is an ongoing trend that is expected to continue (Shafique and Garg, 2017).

To this end, two types of heterogeneous systems exist. First, systems that comprise multiple different devices on the same node, such as a CPU, a GPU and/or an FPGA or ASIC. Each of these chips can be a completely different hardware platform and have its own dedicated memory (e.g. DRAM or HBM). Intra-node communication between these chips is realized via dedicated links, such as PCIe, NVLink, Ultra Path Interconnect, or newer CXL-based technologies. Second, the individual chips themselves may be heterogeneous (SoCs), in that they can encapsulate multiple types of

processing units on the same package, where each processing unit may be part of a different in-package chiplet. One simpler example of such systems is heterogeneous CPU devices, such as the ARM big.LITTLE and Intel Alder Lake (Rotem et al., 2022) architectures, which contain high-power, high-performance with low-power, high-efficiency cores, a method which reduces power during execution of relatively lighter workloads. These cores may exhibit heterogeneity in their microarchitecture, but all use the same instruction set architecture. On the other hand, popular examples of more complex SoCs comprising entirely different processing units include: (a) Apple M1–M4 (Ali et al., 2022; Kasperek et al., 2023; Kenyon and Capano, 2022; Zhang, 2021), used in Mac laptops and iPad tablets, which integrate CPU, GPU, and Neural Processing Unit (NPU) cores on a single chip, with the NPU specifically dedicated to accelerating AI workloads. Similarly, Apple's A16–A18 chips also comprise CPU, GPU and NPU cores, but are instead targeted for smartphones (iPhone); (b) the NVIDIA Grace-Hopper (Evans, 2022; Wei et al., 2023) and Grace-Blackwell (Tirumala and Wong, 2024) super chips and AMD Mi300a (Patel et al., 2023), that integrates CPU and GPU chiplets; and (c) AMD/Xilinx's Versal Adaptive Compute Acceleration Platform (ACAP) (Ahmad et al., 2019) that comprises a CPU, an FPGA, and an AI Engine (AIE) array consisting of up to 400 programmable VLIW/SIMD processors. SoCs are becoming increasingly popular, as integrating accelerators on the same chip can greatly reduce communication costs that relate to performance and power. However, naively moving the accelerator on-chip may not always be advantageous compared to using off-chip accelerators. Additional optimizations are often required to properly manage all the resources and extract maximum performance (Asri et al., 2021). In the following, we discuss on-chip, intra- and inter-node integration aspects of heterogeneous systems. We end by describing Anton, which is a heterogeneous computing system customized for molecular dynamics simulations.

On-chip integration

Heterogeneous SoCs with a diversity of compute platforms and connectivity mechanisms can be complicated and usually require systematic methods for efficient utilization. Key issues related to these devices are the data-sharing between the processing units and how tasks should be partitioned and scheduled across them.

In terms of data-sharing, there are three approaches (Cota et al., 2015; Giri et al., 2018; Peccerillo et al., 2022): (a) non-coherent accelerators, in which each accelerator and the CPU have separate memory spaces, where the data-sharing can be achieved using direct memory access (DMA) (Kim et al., 2021d; Ma et al., 2019; Pham-Quoc et al., 2013; Su et al., 2011; Wang et al., 2021a). DMA is managed by users or the software libraries (Awamoto et al., 2020; Gong et al., 2014); (b) accelerators with coherency at the last-level cache, in which the accelerators share the data with the last level cache of the CPU (Asri and Gerstlauer, 2022; Garcıa et al., 2016; Mekkat et al., 2013); and (c) fully-coherent accelerators, in which a cache coherency protocol handles the data coherency between the CPU and the accelerators, (Kumar et al., 2015; Olson et al., 2017), thus simplifying programming (Lim and Kim, 2012). Some recently released heterogeneous architectures use full-coherence unified memory to simplify programming. Examples include Apple M1 and M2 SoCs (Ali et al., 2022; Kasperek et al., 2023; Kenyon and Capano, 2022; Zhang, 2021), NVIDIA Grace-Hopper super chip (Evans, 2022; Wei et al., 2023), and AMD Mi300a (Patel et al., 2023).

The software stack[153] can boost productivity by detecting available accelerators and offloading tasks to them, hiding most of the inherent complexity from the programmers (Andrade and Crnkovic, 2018; Bragança et al., 2018; Castrillon et al., 2018; Gummaraju et al., 2010; Wang et al., 2018; Zhong et al., 2019; Zuckerman et al., 2021). While the robustness of the software stack[154] can influence the adoption rate of the accelerators in a heterogeneous device, it is not enough for efficiently utilizing all available resources of these devices. One major question that first needs to be answered is what runs where and when. Each computing platform is suitable for different types of workloads. For example, tasks with canonical, mature and widely-used kernels with a large degree of parallelism and a high arithmetic intensity[155] may best be suited for ASICs. Highly memory-bound[156] tasks with inherent parallelism may best be suited for GPUs, due to their large memory bandwidth. Tasks with complicated control flows and limited parallelism are best executed on CPUs. Additionally, the communication cost between different resources in a system is a crucial factor in attainable performance that can affect where each task should be mapped to. Ultimately, maximizing an application's performance on a heterogeneous system is a hardware/software partition and scheduling problem.

The partition problem refers to the assignment of an application's tasks to processing units that optimizes metrics, such as execution time and energy consumption. The scheduling problem refers to mapping each processing unit's tasks to a start time, such that performance is maximized. These problems become increasingly difficult with the complexity of modern heterogeneous platforms. Different strategies that use exact[157], constructive[158], or iterative[159] methods can be used in practise to find optimal or sub-optimal solutions. As each vendor's SoCs use widely different architectures and means of communication, no universal tools that automatically solve the hardware/software partition problem on all SoC platforms are available. However, SoCs typically come with a software suite that provides tools to help with the

exploration and the integration of the system, such as CUDA (Buck, 2007), ROCm (Sun et al., 2018), and Vitis V++ compiler.

Intra-node integration

Nowadays, most devices in a node have their own on-chip memory and are separated from the host CPU. The host CPU is responsible for offloading a task to the accelerator device and reading the results back upon completion. Systems with newer CXL-based memory hierarchy will add more diversity. For example, multiple devices (e.g., CPUs, GPUs, ASICs) may use a unified address space, by sharing a single, large memory device on the compute node in a fully coherent manner, enabling high-bandwidth communication.

In all these cases, the same partitioning and scheduling problem mentioned earlier is applied for such heterogeneous systems. The main difference is that communication, which now accounts for off-chip data transfers, can typically be much slower compared to on-chip. Several tools and libraries are available that can efficiently solve the partition and scheduling problem of intra-node systems with minimal intervention from the programmer (Augonnet et al., 2009; Trott et al., 2022). Each tool supports different sets of architectures and application domains.

Inter-node integration

Inter-node communication can be very slow. However, it is necessary for workloads with very large datasets that require more memory and compute resources. Achieving high performance on these workloads requires identifying suitable types of nodes in a cluster, while also utilizing algorithmic optimizations that minimize inter-node data transfers.

With the advent of disaggregated infrastructure technologies, such as CXL 2.0 and 3.0, more flexibility is added for inter-node communication, which can enable implementations tailored to a workload's needs. Resources, such as accelerators, memory, and storage, can be provisioned dynamically to each compute node[160]. For instance, instead of a rack full of identical compute nodes (which is common today), the disaggregated architectures will have a rack with some nodes containing only memory, some nodes only containing accelerators, and some nodes only containing CPUs. Then, available resources can be dynamically assigned per use case basis. For example, a workload may require two memory nodes and one accelerator node, whereas another workload may need a single memory node and four accelerator nodes. This way, memory, accelerators, and CPUs will no longer reside on the same PCB, as they will be located on different nodes, or even on different racks. This makes interconnection between nodes and racks crucial, in order to achieve the same or even better latency and bandwidth, compared to today's architectures.

An example: Anton specialized chip and system for molecular dynamics

In this part, we consider an example of a heterogeneous and specialized computing system. Anton is an accelerator ASIC and system, developed by D. E. Shaw Research, to accelerate all-atoms molecular dynamics (MD) simulations (Deneroff et al., 2008; Dror et al., 2011). These simulations can predict three-dimensional movement of atoms based on their inter-atomic force interactions. The primary motivation for designing Anton seems to be the enablement of fast MD simulations for drug discovery. To the best of our knowledge, Anton is the only ASIC and ASIC-based system that has been developed primarily for advanced scientific computing. Anton (as a system) is not just a specialized chip: the specialized chip is complemented with the rest of the supercomputing ecosystem[161], which has been specialized to run certain MD simulations as fast as possible. In this sense, the Anton supercomputer is truly unique, and therefore, we review it in more detail.

The first generation of the ASIC (Anton 1), launched in 2008, was manufactured using 90 nm process node technology, and had 200 million transistors that ran at 400 MHz of clock frequency (Shaw et al., 2021). The ASIC had four subsystems: a high-throughput interaction subsystem (HTIS), a flexible subsystem, a communication subsystem, and a memory subsystem.

HTIS handles the most compute-intensive parts of the targeted MD simulations, which are evaluation of the electrostatic and van der Waals interactions between a pair of atoms. Theory that is used for these calculations is well-established, and is unlikely to change during the lifetime of the ASIC. HTIS runs at twice the rate of the system clock, and features 32 pairwise point interaction modules[162] (PPIM). Since they handle the most compute-intensive components of the targeted MD simulations, and the algorithm to calculate them is not expected to change, PPIMs are the most hardwired components of the ASIC. The streaming architecture of each PPIM consists of a 26-stage pipeline, 18 fixed-point adders, 27 fixed-point multipliers, and three look-up tables. Despite being hardwired, multiplexers are included to provide some degree of configurability. HTIS is controlled by a programmable general-purpose processor, referred to as the interaction control block (ICB) core.

The flexible subsystem controls the operation of the ASIC, computes the bond forces, performs FFT and inverse FFT, carries out time integration, and controls the simulation time-step. The flexible subsystem has four processing slices. Each slice comprises a single-core general-purpose (GP) CPU (Tensilica), a remote access unit (RAU), and two geometry cores (GCs) (Kuskin et al., 2008). The GP core is

responsible for managing the MD simulation, monitoring computation progress, and coordinating data transfers. The RAU is a programmable data transfer unit for offloading data transfer tasks from and to the flexible subsystem for every time step. Finally, the GCs are programmable, dual-issue, statically-scheduled SIMD processors that are responsible for calculating bond forces, FFT (Young et al., 2009), inverse FFT, and integration.

The communication subsystem consists of intra-chip and inter-chip (inter-node) interconnects. The intra-chip interconnect is implemented by using two 256-bit communication rings that run at 400 MHz to connect HTIS, the flexible subsystem, the memory subsystem, and six external communication ports. These six ports are used for inter-chip interconnect[163], each providing 50.6 Gb/s bandwidth in each direction (Dror et al., 2010). At the system level, which consists of 512 nodes, arranged in an 8 × 8 × 8 3D torus configuration, each node is directly connected to its six neighbor nodes. This configuration is consistent with the way atoms in MD simulations interact with each other, and, therefore, greatly contributes to the efficiency of the communication.

The memory subsystem provides an interface to the external DRAM. While the aggregate, on-chip SRAM memory can only fit 200 k particles in the 512-node system, using DRAM allows Anton to simulate billions of atoms. This can be achieved by splitting each simulation box into virtual sub-boxes, and then intelligently swapping each sub-box in and out of the chip[164] (Shaw et al., 2007a).

Anton is a great example of a heterogeneous computing system: it comprises three types of embedded cores (i.e., ICB in HTIS, and four GPs and eight GCs in the flexible subsystem), which are tightly coupled to specialized hardware units (Grossman et al., 2008). Programming Anton is not an easy task. The four GP cores and the ICB core are specialized versions of the Tensilica LX processor, and can be programmed in C language. While the GCs are specifically designed for Anton, and are programmed via assembly language.

Preliminary estimated assessments showed Anton 1 is about 500× and 100× faster than two general-purpose clusters at the time, namely, the Sun Fire V20z cluster[165] and IBM Blue Gene/L cluster, respectively (Shaw et al., 2007b; 2009a). Performance results based on the actual hardware were published subsequently, and exceeded the estimated performance results due to higher achieved clock frequency[166] and more optimized algorithms and software (Shaw et al., 2009b).

The second generation of Anton (Anton 2) was introduced 5 years later (2013), relied on 40 nm process node technology, and contained two billion transistors. It runs at a significantly higher clock frequency of 1.65 GHz, has more processor cores[167], more PPIMs[168], and more on-chip SRAM memory[169]. While the terminology remains the same, the chip organization has changed with redesigned subsystems tailored for fine-grained event-driven computation. Instead of having one HTIS and one flexible subsystem, Anton 2 features 16 flexible subsystems (flex tiles) and two HTIS. Each flexible subsystem consists of four GCs with no GPs. Each HTIS consists of a small version of the flexible subsystem featuring one GC and 38 PPIMs. The ICB is no longer used to control HTIS. Instead, the GC inside the small version of the flexible subsystem controls HTIS operations. This simplifies Anton 2 programming since the same tools can be used to program both the flexible subsystems and HTIS. Anton 2 can be programmed using software written in C++. At the system level, Anton 2 is scalable up to 4096 nodes, organized in a 3D torus. The communication bandwidth and efficiency between neighboring nodes has also significantly increased[170] (Grossman et al., 2015; Towles et al., 2014). With these modification, Anton 2 with 512 nodes was about 4.4× to 10.7× faster than Anton 1, depending on the problem size. On average, Anton two was more than two orders of magnitude faster than available general-purpose systems at that time while being more power efficient.

The third generation of the chip (Anton 3), launched in 2020 and manufactured through 7 nm process node, has 32 billion transistors, and runs at 2.8 GHz (Shaw et al., 2021). It is redesigned from the ground up to maximize parallelism. Each chip features 288 core tiles and 24 edge tiles. The core tile comprises two redesigned PPIMs, two GCs, and the new bond calculator, along with 128 kB of flex SRAM. The edge tiles are responsible for establishing inter-chip communications with six neighbor nodes arranged in 3D torus, and provide 464 Gb/s communication bandwidth to each neighbor node per direction. Anton 3 with 512 nodes is about 460× faster than the 512-node HYDRA heterogeneous cluster consisting of two 10-core Intel Xeon E5-2680v2 CPUs and two NVIDIA K20X GPUs per node for a total of 10,240 CPU cores and 1024 GPUs (Páll et al., 2015), when simulating 2.2 million atom ribosomes. It is also 19× faster than 512-node Anton 2. Compared to a single cutting-edge GPU at the time (NVIDIA A100), Anton 3 was 3,000× faster, while being an order of magnitude more energy-efficient (Shaw et al., 2021).

We summarize this part with key observations. The nature of the targeted MD simulations play a fundamental role in Anton's impressive performance: they involve significant local computations, with relatively smaller inter-node communication, when compared to most industry-relevant PDE solvers. Moreover, the communication topology is aligned with the problem geometry, which positively impacts performance. Furthermore, even though the theoretical framework of the simulations are well-established, each generation of Anton was re-designed significantly to adapt to changes in hardware technology. Lastly, this effort highlights the cost of designing ASIC (and computing system) for scientific computing: designing each

generation of Anton took 5–7 years, and involved a multi-disciplinary team of tens of scientists.

Energy consumption of large computing centers and its implications

Energy consumption of large computing centers has significant implications on cost and the power delivery infrastructure, as we discuss next.

Energy consumption of large clusters

The large and ever-increasing energy demands of datacenters comprise a significant portion of their TCO[171]. With the end of Dennard scaling (*Part I: Dennard scaling* (Hanindhito et al., 2026)), datacenter energy consumption will increase from 1.15% of global energy consumption in 2016 to 2.13% in 2030 (Koot and Wijnhoven, 2021), corresponding to 85% increase. With an average electricity cost of five cents per kWh in the United States (Dabbagh et al., 2020), a datacenter with a peak server power consumption of 13 MW (e.g., Summit at ORNL; see Table 8) would require over 5.5 million dollars per year for energy costs alone, excluding overheads due to cooling and power conversion.

Energy efficiency due to improvements in hardware. Energy efficiency of hardware resources depends on many factors, such as architecture, utilized technology, and software optimizations. Newer chips and servers have an ever-increasing power density. At the same time, due to the increased number of transistors per die, as well as ongoing architectural optimizations, computational throughput has greatly increased over time. Accordingly, while the latest devices have higher power demands, they exhibit improved energy efficiency.

Table 8 shows peak performance, power consumption, and energy efficiency for some of the world's most powerful supercomputers[172] over the past two decades. It shows energy efficiency has improved over the years, whereas the total power consumption has increased. For instance, TACC's Frontera (Stanzione et al., 2020) is 80% more energy efficient compared to TACC's Stampede2 (Stanzione et al., 2017), but uses about 40% more power.

Clusters that extract more floating-point performance from GPUs (rather than CPUs) are more energy-efficient, as can be seen by comparing ORNL's Frontier (2021) to Fugaku (2021), where Frontier is about 4 times more efficient. Some clusters, however, continue to rely heavily on CPUs or multi-core processors due to the flexibility they provide. This trend is particularly favored by some academic institutions and industries that primarily rely on scientific computing, have diverse workloads, and deal with complex legacy software that may not be easily ported to GPUs. Representative clusters of this category are Roadrunner (2009), Stampede2 (2017), Frontera (2019),

Table 8. High-performance computing power trends (excluding cooling): While computing efficiency has improved, the need for solving larger and more complex problems has resulted in using more overall power, challenging existing power systems; furthermore, HPC that supports general workloads (e.g., TACC) has lower power efficiency due to its reliance on general-purpose microprocessors.

Launch date	System name	Peak perfor-mance	Flops CPU:GPU	DRAM memory (PB)	Power (MW)	Efficiency GFLOP/s per watt
2023	Aurora[a]	>2 EF	TBD	10.9	60	TBD
2021	Frontier[b]	1.68 EF	10%:90%	9.2	29	69
2021	Fugaku[c]	537 PF	100%:0%	4.85	30	17.9
2021	Lonestar6[d]	5.89 PF	55%:45%	0.17	0.46	12.8
2019	Frontera[d]	44.5 PF	87%:13%	1.5	5.5	8.1
2018	Summit[b]	200 PF	4%:96%	2.67	13	15.4
2018	Sierra[e]	126 PF	7%:93%	1.38	11	11.5
2017	Stampede2[d]	18 PF	100%:0%	0.76	4	4.5
2016	Sunway[f]	125 PF	100%:0%	1.31	15	8.3
2016	Piz Daint[g]	25 PF	10%:90%	0.34	1.6	15.6
2012	Titan[b]	27 PF	10%:90%	0.76	8.2	3.3
2009	Roadrunner[h]	1.7 PF	100%:0%	0.11	2.35	0.72
1997	ASCI Red[i]	1.3 PF	100%:0%	0.001	0.85	0.0015

[a]Argonne National Laboratory (ANL), USA.
[b]Oak Ridge National Laboratory (ORNL), USA.
[c]Riken Center for Computational Science (R-CCS), Japan.
[d]Texas Advanced Computing Center (TACC), USA.
[e]Lawrence Livermore National Laboratory, USA.
[f]National Supercomputer Center, China; Many-core RISC CPUs.
[g]Swiss National Supercomputing Centre, Switzerland.
[h]Los Alamos National Laboratory (LANL), USA.
[i]Sandia National Laboratories (SNL), U.S.A.

and Fugaku (2021) (Table 8); energy efficiency of these systems has improved by about 20 times over the past 14 years.

Energy efficiency due to improved cooling systems and reduced overheads. Several metrics have been proposed for measuring the overall energy efficiency of a datacenter (Shao et al., 2022). One of the most widely-used metrics is Power Usage Effectiveness (PUE):

$$\text{PUE} = \text{Total Facility Energy} \;/\; \text{IT Equipment Energy},$$

where IT Equipment Energy is the energy consumed by servers, networking and other IT equipment. A metric similar to PUE is the server PUE (SPUE), which measures the overheads inside the servers themselves. Accordingly, SPUE is the ratio of total server power to the power consumed by components that are directly responsible for computing[173] (Barroso et al., 2019). Therefore, a datacenter's energy efficiency is measured through:

$$\begin{aligned}\text{Efficiency} &= \frac{\text{Compute Throughput}}{\text{Total Facility Energy}} \\ &= \frac{1}{\text{PUE} \times \text{SPUE}} \frac{\text{Compute Throughput}}{\text{Compute Energy}},\end{aligned}$$

where Compute Energy is the energy that was used by hardware that is directly responsible for computing. Accordingly, power used for cooling impacts PUE and reduces Efficiency.

A large variety of cooling systems have been proposed and deployed, such as air cooled, air chilled, liquid cooling, and free cooling, leading to a large variation of attainable datacenter PUE. While the average datacenter PUE in 2022 was 1.55, according to the Uptime institute's global datacenter survey, large-scale, state-of-the-art data centers achieve PUEs below 1.2 nowadays (Barroso et al., 2019). For instance, Google has been achieving a fleet-wide PUE average of less than 1.1 since 2020.

Interplay between future computing centers and the power infrastructure

Considerable energy demands of modern clusters and datacenters have significant implications, highlighted below.

Upgraded power infrastructure. The large power required by modern computing centers typically imposes a heavy load on the power grid. Sometimes, this requires upgrading and expansion of the power infrastructure (e.g., power grid and transformers), which could be complicated and costly. For instance, 2.5 mile-long power lines connected to the nearest power plant had to be installed in order to power Frontier.

Transition to lower-carbon and renewable energy. Some companies attempt to reduce their carbon footprint by powering their computing centers with lower-carbon and renewable energy sources. For instance, Google's datacenters operate at over 67% on renewable energy (Harkness, 2023), with setting a goal to run on 100% lower-carbon energy by 2030.

Data-centers near nuclear power plants. To use more reliable, efficient, and low-carbon energy sources, data-center companies are examining nuclear energy as a power source for the near future. Current projects consider proximity to an existing nuclear power plant. In January 2023, Cumulus Data completed a 48 MW data-center shell, which is directly powered by the 2.5 GW Susquehanna nuclear power plant station in northeast Pennsylvania (Chernicoff, 2023).

Small modular reactors (SMRs) for data-centers. Compared to conventional nuclear power plants, SMRs are much smaller in size, allowing them to be portable. An SMR can typically provide between 50 MW and 300 MW of power. Recent advances in SMRs motivate placing them near large datacenters. There is currently about 80 commercial SMR designs under development worldwide. However, data-center companies are still unsure as to how they will perform in practice, and how the operating costs will be affected.

Some companies (e.g., Microsoft) are planning to install and deploy SMRs to power their data-centers in the future (Bradstock, 2023). NuScale Power is the first company that designs SMRs to receive approval by the U.S. Nuclear Regulatory Commission (NRC), and is planning to power two new data-centers in Ohio and Pennsylvania (Larson, 2023).

Impacts on high-performance scientific computing

For a long time, continuous improvements in computing hardware resulted in less pressure on many scientific computing applications to fundamentally reconsider their underlying algorithms. During this era, performance was significantly improved through increasingly more powerful processors as a result of technology scaling. Many applications also realized higher performance through parallelization, either via multi-core processors or multi-node computing. Concurrently, new algorithms, such as fast Fourier transform (1960s), multigrid (1970s), the fast multipole method (1980s), sparse grids (1990s), H matrices (2000s), multi-level Monte Carlo (2000s), and randomized matrix algorithms (2010s), considerably reduced the computational complexity of their predecessors (Dongarra and Keyes, 2024).

Even compared to 10 or 15 years ago, the hardware technology has changed strikingly: (a) number of cores within a chip has increased significantly, which makes finding opportunities for (within-the-chip) parallelization even more important; (b) the capacity of processors to

crunch numbers have outpaced their ability to fetch data from off-chip memory, which puts greater emphasis on communication-reducing algorithms; (c) existence of different types of off-chip memory technologies invites novel algorithms for efficient data orchestration between the different memory components and the processing units; (d) energy efficiency considerations are driving heterogeneous computing, making accelerators, general-purpose processors, and other devices that exist on a computing system increasingly more diverse, which then makes efficient data orchestration between these components more challenging; and (e) AI's large market size has popularized GPUs, and currently, is the primary driver of custom hardware development. While these aspects aimed to boost performance through various means, they have introduced new challenges that will not go away with time; they will likely get worse.

Furthermore, as technology scaling[174] is tapering, it will no longer deliver considerable improvements in performance. Accordingly, Shalf (2020) suggests meaningful performance gains may be realized through: (a) *designing hardware-friendly algorithms* in order to harness an existing hardware more effectively; (b) *designing specialized hardware for certain algorithms or applications* (*Part I: Examples of custom and specialized hardware* (Hanindhito et al., 2026)); however, this approach typically involves making changes to the underlying algorithms as well and is therefore closely related to (c) *hardware-algorithm co-design,* where hardware and the underlying algorithms or applications are designed together to maximize performance, including energy efficiency (Krueger et al., 2011). Hardware-algorithm co-design could be challenging, as it demands significant research and development efforts. Consequently, it may be adopted by limited groups within the scientific computing community that may be able to provide sustained funding to such undertakings, such as those who design and construct leading-edge high-performance computing systems (Reed et al., 2023).

Through a techno-economic analysis, Thompson and Spanuth (2021) argue that only applications with a high-value market can afford the R&D investments needed for specialization and thus be in the 'fast lane'; applications with a small market size will be in the 'slow lane' and have to contend with slowly-improving CPUs. This is ominous news for those scientific computing applications that may not have a high-value market.

Lastly, we refer to a thought-provoking article by Matsuoka et al. (2023) for a different presentation style and perspective on where high-performance computing is heading before sharing our own. The authors discuss diverse topics such as hardware accelerators, extreme specialization, cloud, zettascale, and quantum computing, among others.

In the remainder of this section, we highlight the diversity that exists in scientific computing software, along with associated challenges. We then feature several options with varying levels of R&D effort, which promise to effectively harness modern hardware to deliver improved performance.

Software diversity in scientific computing

Numerical linear algebra, numerical solution of ordinary and partial differential equations (PDEs), and numerical optimization, frequently occur in many applications. Each of these branches have many variations, making scientific computing very diverse.

For instance, industry-relevant, large linear systems of equations are often sparse. Each particular application lends a specific mathematical structure to the linear system. Exploiting this structure in the utilized algorithm often improves performance. For instance, when a steady-state stress analysis in the automotive industry involves many different components[175], the resulting linear system of equations offers some properties and structure to exploit. These include sparsity, positive-definiteness, and likely a large condition number due to the considerable differences in the stiffness matrices of the various components. In these situations, sparse direct solvers are often used due to their robustness. Often times, this step takes a significant portion of the total simulation time. Unfortunately, sparse direct solvers do not scale well on parallel computing systems. On the other hand, fluid dynamics simulations in the automotive industry, aimed at minimizing drag, lend a different structure to the linear system of equations. This allows using sparse iterative solvers, which scale better on parallel computers. However, effective utilization of iterative solvers often involves exploiting problem-specific mathematical structures in the adopted algorithm, which may include designing a customized preconditioner.

Similarly, industry-relevant partial differential equations are also very diverse. For instance, numerical solution of the acoustic and elastic wave equation dominates the computing resources of the oil and gas industry, and often relies on explicit, high-order finite differences, which scale well on parallel computers. Porting algorithms of this type to GPUs results in performance gains, and is becoming more common. On the other hand, unstructured-mesh-based PDE solvers that are used for hurricane storm surge predictions typically rely on low-order finite elements for robustness. They also have complex workflows[176]. Moreover, they involve solving a large, sparse linear system of equations, often through an iterative scheme. The utilized algorithms are mostly suited for multi-core CPUs. Porting these algorithms to GPUs, in a way that leads to meaningful performance gains, is extremely challenging.

Numerical optimization is commonly used in scientific computing. Some mathematical optimization schemes, such as the branch-and-bound method for mixed-integer nonlinear optimization, do not exhibit a faster runtime when parallelized (Maher et al., 2021). Therefore, there is little to be gained by porting them to GPUs, or multi-core

processors. Performance gains in this area has often been enabled by improved algorithms. Gradient-based optimization schemes[177] often inherit a good portion of the structure of their associated underlying problem. For instance, industry-relevant gradient-based optimization schemes that are applied to the wave equation for oil and gas discovery parallelize well on modern architectures, such as multi-core processors and GPUs. They typically involve a significant memory footprint[178], which requires careful data orchestration as one migrates from multi-core CPUs to GPUs[179].

These examples, while not comprehensive, illustrate the diversity of algorithms that are encountered in scientific computing. Accordingly, a successful strategy that leads to meaningful performance gains necessitates careful analysis of the underlying algorithms of a specific application, their potential for parallelization, and their computational bottlenecks. Often, a commonly-used algorithm needs to be markedly revised to enable meaningful performance gains when executed on modern hardware. The diversity of scientific computing applications can hinder the securing of resources for such endeavors.

Challenges

We highlight some of the key challenges that hinder the adoption of modern hardware in scientific computing.

Old software. Many scientific computing applications rely on software that have been under continuous development for a long time. Some widely-used examples include: Computer-Aided Engineering (CAE) software packages, such as ANSYS, Abaqus, and Nastran; Earth system modeling packages, such as ADCIRC (Westerink and Luettich, 2024), MITgcm, and WRF (Skamarock et al., 2019); and software for molecular dynamics simulations, such as LAMMPS-MD (Thompson et al., 2022), GROMACS (Abraham et al., 2015), and NAMD (Phillips et al., 2005). These developments have been verified, extensively tested, and sometimes certified by regulatory agencies.

Changing the underlying algorithms of these software systems to make them attractive to modern hardware presents significant challenges. These include: (a) it is very difficult to extract performance from modern hardware (e.g., GPUs) if the underlying algorithms have low potential for parallelization, or if they have low arithmetic intensity[180]. For instance, many of the industry-favored algorithms use low-order discretization methods. Developing efficient, industry-attractive, high-order[181] discretization schemes require significant R&D efforts, and is often a high-risk, multi-year, multi-disciplinary endeavor; (b) migration of the workflow to modern hardware may not be a top priority due to resource limitations. Sometimes, a small part of the workflow gets ported to modern hardware (e.g., GPUs) as an exploratory step. However, migration of the entire workflow may be hindered; and (c) the re-certification process by regulatory agencies, if needed, could potentially be long and costly, and sometimes, it may discourage changing the underlying algorithms.

Flexibility vs. performance. Satisfying both flexibility and high-performance in scientific software is often challenging. By flexibility, we refer to the ability of a software package to take contributions from a large group of users, often scientists, who may not be well-versed in advanced coding. The flexibility allows users to make changes to the source code in order to implement functionalities based on their specific needs. Examples include LAMMPS-MD, which allows users to code their own potential energy function, and ADCIRC, where its algorithms have been under continuous improvement by computational scientists over the past decades. In addition, a top priority for a large-enough group of such users is productivity, which the tools support by allowing users to write code in C++ or Fortran[182]. This allows users to spend their time on testing ideas and algorithm development, rather than thinking about writing performant code on a non-general hardware platform, e.g., GPUs. Once the ideas and algorithms have been sufficiently tested on CPUs, they may find their way to a GPU code. Accordingly, some applications maintain a CPU code for productivity and development purposes, and a corresponding GPU version for performance. For instance, while LAMMPS-MD can run on GPUs for routine calculations, it still needs to run on CPUs for non-routine cases.

Poorly-parallelizable algorithms. Many industry-relevant applications rely on algorithms that do not scale well when more cores are available. A remarkable example is sparse direct solvers, which, often times, consume the bulk of the simulation time in CAE software. Developing algorithms that offer parallelizability, without compromising robustness, could be very challenging. Mixed-integer nonlinear optimization is another example where parallelization does not yield meaningful performance gains.

Scientific computing's small market. Unlike machine learning and AI, many scientific computing applications have a small market size. This can sometimes make justifying investment decisions in the high-risk areas difficult. Some examples include migration of a complex workflow from CPUs to GPUs; and designing specialized hardware for specific applications (e.g., molecular dynamics for drug discovery, or wave simulation for oil and gas exploration). Making the situation even more difficult, it is often easier for ML and AI to attract top talent, thanks to their large and rapidly growing market, compared to scientific computing.

The large market size of ML and AI has a considerable impact on the scientific computing community, and is inspiring it to explore ways to ride the wave of ML and AI to

manage investment costs. Examples include: solving PDEs on Cerebras's large chip, which was primarily developed for ML applications (*Part I: Examples of custom and specialized hardware* (Hanindhito et al., 2026)); computational fluid dynamics (CFD) simulations on Tensor Processing Units (TPU) (Wang et al., 2022); and physics-informed ML, which often exploits GPUs. Nevertheless, these approaches have yet to become mature enough to be adopted for industry-relevant applications at scale.

Opportunities

Based on the level of effort one is willing to invest, there is some flexibility in how modern hardware can be used to improve performance. We highlight some of the most promising options next.

Leveraging modern hardware with minimal code change. This option could potentially be attractive to many academic researchers, as well as industry practitioners, since improving performance through making extensive algorithmic modifications may not be their highest priority. It may also appeal to old software that has been under development for a long time, and sometimes, may face difficulties in securing resources for code modernization. For instance, the Texas Advanced Computing Center (TACC) provides support for this approach through its CPU-heavy clusters (Table 8) in order to accommodate a diverse group of users. According to Stanzione (2022), Executive Director of TACC, less than 10% of TACC's workload would work on Department of Energy (DOE) Oak Ridge National Laboratory's Frontier, which is a GPU-heavy cluster (Table 8). The DOE has invested over a billion dollars on exascale codes, which primarily benefit extreme-scale applications that are intended to run on clusters similar to Frontier; however, the type of codes that run on TACC are largely different (Stanzione, 2022).

A successful strategy to improve performance under this approach requires a thorough understanding of the code, its execution flow, potential for parallelization, and memory footprint. Identification of hot-spots can provide insights on whether limited and targeted efforts directed at alleviating major bottlenecks can deliver meaningful performance gains. These low-effort options include targeting a small part of the code through: (a) limited algorithmic modifications; (b) using a modern external library[183] to perform the task of a hot-spot; and (c) offloading a compute-intensive hot-spot to a hardware accelerator.

Little can be done when a software has a complex execution flow, does not have a few distinct hot-spots, and offers limited opportunities to benefit from parallelization. Examples include sparse direct methods for solving a linear system of equations, and branch-and-bound method for mixed-integer programming. In these situations, a modern, low-core-count CPU[184], with a high clock frequency, would likely result in a faster run-time. The low core-count allows for a higher power budget, which then permits higher clock frequencies. This enables higher single-thread performance. Furthermore, more cache per core is often available to modern, low-core-count CPUs. This typically helps applications that have complex execution flows. Lastly, a multi-core CPU with HBM can enable significant performance gains if the application is small enough to fit in the high-bandwidth memory.

For applications that already have been parallelized, but have a complex execution flow, using multi-core CPUs with a large amount of last-level cache could be most impactful in terms of faster run-time. If these CPUs are further equipped with HBM, they may enjoy faster run-times, as HBM may be used as a large last-level cache. Depending on the application, its arithmetic intensity, and the amount of intermediate computations that need to be kept near the registers, the desired amount of cache per core could vary. For applications with very high arithmetic intensity, a high-core-count CPU may deliver the highest run-time performance. Applications that have low arithmetic intensity with large amounts of intermediate results may benefit mostly from a low-core-count CPU that often provides more cache per core. Alternatively, to provide more cache per core, a high-core-count CPU may be used where not all cores are utilized. Larger applications may be impacted by inter-node communication. Oftentimes, a satisfactory configuration[185] that meets runtime, budgetary, and other desired metrics could be found through experimentation.

Leveraging modern hardware via considerable algorithmic changes. Some applications offer considerable opportunities for parallelization. These applications are at the forefront of algorithmic modification to harness modern hardware, if they are used by a large-enough market. Most likely, these applications have already benefited from parallel processing, multi-core, and many-core processors. Some examples include full-waveform inversion for oil and gas exploration, electromagnetic wave simulations for communication and defense applications, and molecular dynamics for drug discovery and material design. If these applications happen to have few canonical kernels, simple execution flow, and high computation to communication ratio, they become prime candidates for GPUs.

Next, we highlight several approaches that hinge on algorithmic modifications, resulting in a more effective way of harnessing modern hardware: (a) possible reformulation to increase arithmetic intensity, i.e., reducing data movement (Abduljabbar et al., 2017), perhaps at the expense of increasing local computation, e.g., through using different discretization schemes (Modave et al., 2015); (b) increasing locality to improve cache efficiency (Malas et al., 2015); (c) relaxing communication and synchronization constraints (Kumari and Donzis, 2020); (d) exploiting the growth in on-chip cores (Ang et al., 2014; Kogge and Shalf, 2013); (e)

judiciously choosing the needed level of arithmetic precision at different parts of the code to alleviate communication bottlenecks and improve resource utilization (Abdulah et al., 2022; Croci and Giles, 2022; Croci and Rosilho De Souza, 2022; Goddeke et al., 2007; Haidar et al., 2017; Higham and Mary, 2022; Komatitsch et al., 2010); (f) selectively recomputing certain variables instead of storing them in memory, as the memory per core decreases; (g) simplification of the execution flow (Bielak et al., 2005); and (h) performing computations more efficiently, e.g., through using octree-based meshes (Burstedde et al., 2011; Rudi et al., 2015; Sundar et al., 2012), hierarchical algorithms (Abduljabbar et al., 2019; Keyes et al., 2020), and hierarchical matrices (Boukaram et al., 2018; Litvinenko et al., 2019).

Such algorithmic modifications are sometimes easier said than done for real-world applications. Generally, specific characteristics and needs of an application impede a straightforward implementation of a new algorithm into an existing framework. Typically, such endeavors involve multi-year, multi-disciplinary efforts for complex engineering applications.

Lastly, we point to the *algorithmic Moore's law* suggested by Keyes (2023), where he observed an exponential rate of improvement for several algorithms that are fundamental in computational fluid dynamics and complex kinetics. Sustainment of such improvements has been viewed with caution by others; for instance, once an algorithm achieves linear complexity, there will be limited opportunities for further improvement (Matsuoka et al., 2023).

Leveraging developments in machine learning. Some algorithms, frameworks, or hardware that were originally developed for machine learning may also benefit scientific computing. A notable example is scientific machine learning (SciML), which refers to the solution of problems in scientific computing through leveraging data-driven approaches and/or techniques (e.g., automatic differentiation) that are typically used in machine learning. SciML methods are becoming increasingly popular in the scientific computing community. This popularity is primarily attributed to: (a) availability of many open-source ML-libraries and software, which significantly impact productivity; (b) ability of these methods to seamlessly incorporate data into first-principle-based models; (c) large market size of ML and AI, which impacts scientific computing[186]; and (d) prevalence of GPUs in academia and industry compared to a few years ago.

Current SciML methods have mostly been applied to academic examples: they often consider one- or two-dimensional geometries and focus on proof-of-concept. Recently, there have been numerous attempts to incorporate complexities that exist in real-world problems, such as material heterogeneity, complex geometry, and richer physics. Some industries are also exploring the potential of SciML, including its ability to handle complexities that exist in real-world problems.

Oftentimes, SciML methods are not able to compete with traditional numerical methods in terms of accuracy and robustness in a classical sense. Nevertheless, this property is typically not needed for them to be impactful. Therefore, they are not expected to replace traditional techniques, such as the finite element method to solve PDEs. However, the ability of some SciML techniques to provide quick solutions to parameterized problems (e.g., PDEs that rely on a few input parameters) and to seamlessly incorporate observations, or other forms of data, make them attractive. SciML could possibly be leveraged in frameworks that rely on ensembles, such as uncertainty quantification, or inner loops of a multi-fidelity optimization scheme that can exploit fast but less accurate function evaluations (Ghattas et al., 2021). In other words, SciML will likely be most impactful in situations where accuracy can be compromised for speed, as well as in circumstances where data should be incorporated or its utilization lends robustness to the solution. Next, we highlight ML-based methods that have sparked considerable interest in the scientific computing community.

Physics-informed neural networks (PINNs). In an influential work, Raissi et al. (2019) demonstrated how modern technologies in ML can be used to solve PDEs. They used neural networks to represent the solution space and automatic differentiation to readily compute partial derivatives in a PDE. A loss function, comprised of the PDE terms, along with initial and boundary conditions, is then minimized to compute the weights and biases of the associated neural network. The ability of the technique to enforce PDE terms, incorporate partial data (Karniadakis et al., 2021) and observations (e.g., in inverse problems), its simplicity due to exploiting automatic differentiation, as well as existence of software systems that allow others to test ideas quickly, continues to generate a lot of interest in the computational science community, resulting in many related works (Cuomo et al., 2022; Haghighat et al., 2021; Jagtap et al., 2020; Kharazmi et al., 2021; Krishnapriyan et al., 2021; Lu et al., 2021b; Penwarden et al., 2023; Yang et al., 2021; Yu et al., 2022).

Fourier neural operator (FNO). The inspiring work of Kovachki et al. (2023) presents a technique for solving parametric PDEs, where a special neural network architecture, motivated by using Green's functions to solve PDEs, is used to represent the numerical solution. The original technique uses sequential layers, where each layer performs discrete Fourier transform on the incoming input, discards higher modes, and then performs inverse Fourier transform, effectively filtering out higher frequencies. The Fourier transform is applied across all spatial dimensions, as well as in time, when applicable.

FNO and its variants have been applied to a wide range of problems in scientific computing (Grady et al., 2023; Kovachki et al., 2023; Lehmann et al., 2024; Leinonen et al., 2024; Li et al., 2023b, 2024).

Deep operator network (DeepONet). Lu et al. (2021a) proposed a special neural network architecture for solving parametric ODEs and PDEs, referred to as DeepONet. The approach is inspired by a theorem (Chen and Chen, 1995) that states a single hidden layer in a neural network is able to accurately approximate any continuous operator[187]. The architecture of DeepONet is directly motivated by the above-mentioned theorem, but with added layers for better expressibility. We highlight key characteristics of DeepONet by considering $\frac{d^2u(x)}{dx^2} = f(x)$, where we wish to find $u(x)$ for any given $f(x)$ within a family of functions. DeepONet uses two separate networks to learn a relationship between input functions $f(x)$, and output functions $u(x)$. A *branch* network represents input functions sampled at predefined points: $\{f(x_1), \ldots, f(x_m)\}$. A *trunk* network represents at which arbitrary points $\{x_i, x_{ii}, \cdots\}$ the output is evaluated. The above branch and trunk networks are combined to compute output at the evaluation points: $\{u(x_i), u(x_{ii}), \cdots\}$. DeepONet has inspired many variants that enhance its capabilities (Ahmed and Stinis, 2023; Gao et al., 2024; Garg et al., 2022; Lee et al., 2023a; Mandl et al., 2025).

Autoencoders (AEs). Autoencoders are used for dimensionality reduction. They have similarities to the singular value decomposition (SVD) in the sense that both can be used to compress data by storing its most salient features. These salient features are referred to as singular vectors in SVD and latent-space vectors in AE. Nevertheless, SVD is a linear transformation, whereas AE can capture nonlinear relationships in data due to its reliance on neural networks. Specifically, an autoencoder takes input x, and maps it to the latent space z, through an encoder: $z = E_\phi(x)$, where ϕ represents the parameters of the associated neural network. A decoder maps the latent-space vector back to the original space: $x' = D_\theta(z)$, where θ denotes the parameterization of the decoder. Parameters ϕ and θ are learned such that x and x' are close to each other in some sense, and the dimension of z is much smaller than that of x. Several works (Kadeethum et al., 2022; Kim et al., 2022c; Wang et al., 2021b) have used autoencoders for reduced order modeling (ROM).

Variational autoencoders (VAEs) extend autoencoders by encoding the latent space as a probability distribution[188] rather than as fixed variables. The probabilistic latent space allows it to be sampled, which can then be used to generate new outputs through the decoder. Therefore, VAEs are known as *generative* models. This property may be used for image generation and data augmentation. Several works (Abubakar et al., 2022; Chen et al., 2023a; Lopez-Alvis et al., 2021), for instance, have used VAEs to generate new realizations of the subsurface geology that are consistent with a training dataset. The new realizations may then be used in parameter estimation and uncertainty quantification (UQ), among other applications.

Generative adversarial networks (GANs). GANs (Goodfellow et al., 2014) can generate new data (e.g., images), or augment missing data, such that they share similarities with a training dataset. For instance, GANs have been used to generate new images of the subsurface geology (Feng et al., 2022; Miele and Azevedo, 2024; Puzyrev et al., 2022; Zhang et al., 2021) that have similarities with an available training dataset. GANs are comprised of two main components: a generator (G) and a discriminator (D), each typically being constructed with a neural network. G takes input from a latent space and transforms it to data that resembles the training dataset. D determines whether the generated data is statistically similar to the training dataset or not. The training continues until G can generate high-quality data in the sense that it becomes difficult for D to determine if the newly-generated data is produced by G or is from the original training dataset.

The latent spaces in GANs and VAEs are slightly different: a VAE enforces an explicit probability distribution (e.g., multivariate Gaussian) for its latent space, whereas the probability distribution of a GAN's latent space is general and learned from data. Lastly, GANs are able to generate sharper images than VAEs due to using a discriminator in training.

Normalizing flows. Normalizing flows can improve the expressibility of the latent space in VAEs and GANs, among other things (Kobyzev et al., 2020). They are invertible functions that map a simple probability distribution (e.g., a multivariate Gaussian) to a more complex (e.g., multimodal) distribution, where the latter is better suited to model complex relationships (Papamakarios et al., 2021). The invertibility property enables a two-way mapping between the latent space and the data space. Consequently, a straightforward statistical sampling in the latent space from a simple distribution can be mapped to the data space. On the other hand, mapping transformed data back to the latent space enables exact density estimation and computing the likelihood of observed data under the model. Some problems in scientific computing, such as probability density estimation (Maeda and Ukita, 2023), and variational inference (Rezende and Mohamed, 2015), can benefit from ML-based techniques that use normalizing flows, among other applications (Albergo et al., 2019; Köhler et al., 2020; Noe et al., 2019).

Transformers. Transformers (Vaswani et al., 2017) are predominantly used in natural language processing (NLP) for language translation, text summarization, and sentiment

analysis. They can capture long-term dependencies within a sequence (e.g., text) through a mechanism called attention, which measures similarities between each two entries in the sequence. At a high level, a transformer consists of an encoder and a decoder[189]. The encoder captures dependencies and relationships within the input sequence. The decoder takes the encoded representations from the encoder and generates contextually relevant output, step by step, using the previously-generated parts of the sequence as context. Detailed description of various mechanisms of attention is beyond the scope of this paper. A mathematically precise and clear description of transformers is given by Turner (2023). Due to their ability to capture long-term dependencies[190], transformers have been used for data-driven operator learning of PDEs (Geneva and Zabaras, 2022; Li et al., 2023c; Zhou et al., 2024) and ODEs (Shih et al., 2025).

Diffusion probabilistic models (DPMs). Diffusion probabilistic models (Ho et al., 2020), also known as diffusion models (Sohl-Dickstein et al., 2015), belong to the broader class of generative models, which can generate new data that are statistically similar to an associated training dataset. DPMs are widely used for image generation, image denoising, image inpainting, and improving image resolution (super resolution). Through a *forward diffusion* process, Gaussian noise is incrementally added to an input image, via a Markov process, effectively transforming the input image into pure noise. Subsequently, a *reverse diffusion* process learns how to remove the aforementioned noise and recover the original input image. Consequently, DPMs can take pure noise as input to generate data that is statistically similar to an associated training dataset. Diffusion models have been used as conditional probability density samplers for subsurface inference in geophysics (Ravasi, 2025; Stojkovic et al., 2024; Zhang et al., 2024), subsurface multi-physics monitoring (Huang et al., 2024), solving PDEs (Yang and Sommer, 2023), and improving the resolution of discretized solutions (Lu and Xu, 2024), among other applications.

Specialized hardware for ODEs and PDEs. Developing specialized hardware for solving ODEs and PDEs is becoming more attractive. Anton, a specialized chip and computing system which was designed to target certain large-scale ODEs in molecular dynamics, is a successful example.

Building specialized chips for solving PDEs is more challenging, as it entails more kernels, which depend on the specific PDE, the associated discretization scheme, and sometimes the targeted problem. Moreover, most industry-relevant PDE simulators are limited by memory bandwidth. This implies that, while building a specialized chip for a particular PDE may lead to substantial performance gains on a single chip, multi-chip systems that are needed to solve larger problem sizes may not carry on the same level of performance. Therefore, algorithmic modifications to alleviate the memory bottlenecks become even more important for these multi-chip systems. Oftentimes, a hardware-algorithm co-design approach is pursued in such circumstances where both the hardware and algorithm are specialized to deliver maximum performance.

FPGAs are typically used for prototyping the chip design, prior to implementing it on an ASIC. There are several research works that use FPGAs for solving PDEs with different numerical methods. The work presented in (Lindtjorn et al., 2011) accelerated finite element sparse-matrix solvers and 3D convolutions for acoustic waves. Some works use the continuous Galerkin finite element (He, 2010) and spectral element (Karp et al., 2021) methods, while others employ the discontinuous Galerkin (dG) finite element method on meshes with unstructured tetrahedral elements (Kenter et al., 2018, 2021), or meshes with structured hexahedral elements (Gourounas et al., 2023). These works showed promising results against reference implementations on CPUs and GPUs. The FPGA design in (Lindtjorn et al., 2011) outperformed CPUs and GPUs by up to 70× and 14×, respectively, while the one in (Kenter et al., 2021) outperformed the reference CPU implementation by 43× to 144×. Moreover, the accelerator in (Gourounas et al., 2023) achieves 4.27× higher throughput than 24 Xeon CPU cores, with 31.33× higher energy efficiency. However, it does not utilize HBM, showcasing suboptimal scalability with memory bandwidth. In a later work (Gourounas et al., 2025), the authors extended their design to HBM-enabled FPGAs, outperforming GPUs by up to 48%, with up to 2.84× higher energy efficiency. Additionally, in (Gourounas et al., 2023, 2025), the proposed dataflow and architecture can support a plethora of hyperbolic PDE solvers under the dG scheme, by simply making minor changes to the HDL code (*Part I: Implementation choices* (Hanindhito et al., 2026)) and reconfiguring the FPGA. Such approaches can be reinforced with the use of custom HDL generation tools, thus minimizing programmability overheads of the FPGA accelerators. Therefore, FPGAs can achieve much higher flexibility than ASICs, while showing higher energy efficiency compared to CPUs and GPUs (Boutros et al., 2020; Gourounas et al., 2023, 2025; Zhuang et al., 2023) As a result, FPGAs are a challenging hardware platform candidate for PDE solvers, where the large diversity of compute kernels and their possibly continuous evolution imposes severe challenges to the design of ASICs that can support a large number of applications.

Overall, the main disadvantages of FPGAs compared to GPUs include longer and more complex development cycles, as well as much lower available memory bandwidth[191]. The latter can become a major performance bottleneck for many memory-bound HPC workloads (Gourounas et al., 2023; Hanindhito et al., 2022). Nevertheless, as FPGA software support matures and approaches like programmable overlays (Fowers et al., 2018) become more popular, the productivity gap between the two is expected to become

smaller. Moreover, as FPGAs evolve to more advanced technology nodes, and faster HBMs are integrated on the same package, higher clock frequency and memory bandwidth will become available at even better power envelopes[192].

We are not aware of a publicly-known ASIC that was specifically built for solving PDEs. Nevertheless, there are a few examples of using machine learning chips that were also used to solve PDEs (*Part I: Examples of custom and specialized hardware* (Hanindhito et al., 2026)). These examples required significant effort to map the PDE to the specialized hardware, and sometimes needed algorithmic modifications to address bottlenecks. These examples demonstrate significant performance gains as along as the problem size fits into a single chip. Off-package communication severely degrades the run-time performance when several packages have to communicate in order to target larger problem sizes.

While an ASIC designed specifically for solving PDEs could yield higher performance and energy efficiency compared to other hardware platforms, it may not always be the most viable approach. PDE solvers could include a wide variety of equations, discretization schemes, computational details, and dataflows. To support a sufficiently large group of applications, an ASIC must be designed to be highly programmable, which brings additional overhead. Even then, different applications will exhibit different problem sizes or arithmetic intensities. Designing the ASIC to account for the worst case scenario can lead to the severe under-utilization of the hardware in some applications. For instance, imagine an ASIC that can handle both the acoustic and elastic wave equation. The elastic wave equation has more variables, and entails more intermediate computations. Therefore, the specialized memory and compute units of this ASIC need to be carefully balanced (designed) to maximize desired performance metrics. The acoustic wave equation would need a different balance to be optimal. Therefore, an ASIC that is aimed to handle both would be suboptimal.

Emerging computing technologies. The slowdown in Moore's law and the growing computational demands of today have spurred the development of alternative[193] approaches to computing, whose potential impact cannot be understated. Leveraging specific physical characteristics of a system, such as high-speed transmission in photonic computing or the storage capacity of DNA molecules in DNA computing, to perform computational tasks may unlock innovative solutions to complex problems. More established emerging paradigms such as quantum computing and physical annealing hold promise for revolutionizing computational endeavors across many application domains. Quantum computing's innate ability to efficiently model complex quantum systems, that are difficult or impossible to simulate with classical approaches, make it particularly well-suited for addressing challenging problems in physics and chemistry. Physical annealing techniques offer a unique approach to optimization tasks by mimicking physical processes to find globally optimal solutions, with applications ranging from chip design to protein folding. When seamlessly integrated with traditional high-performance computing architectures, these novel technologies can serve as potent accelerators, propelling computational science into uncharted realms of discovery and innovation.

It is imperative to acknowledge that while promising, these novel technologies are still in nascent stages of development and not yet fully matured. Despite this, their impending emergence underscores the significance of proactive recognition within the computational science community. By embracing and adapting to the evolving landscape of computational technology, practitioners can position themselves at the forefront of innovation, poised to harness the transformative potential these advancements offer. This forward-thinking approach not only fosters a culture of adaptability but also ensures that practitioners are primed to capitalize on the unparalleled opportunities that lie ahead in the realm of computational science.

Practitioners within the field of computational science are urged to discern and embrace the fluid trajectory of computational technology, strategically harnessing external advancements for maximal efficacy.

Outlook

If the future of computing could be summarized in two words, they would be specialization and diversity. As technology scaling reaches its limits, improvements in performance will increasingly rely on exploiting the specifics of an application. This entails building hardware that is specialized for a specific application, or a class of similar applications, as well as designing algorithms that exploit properties and structures of the specified application and also adapt to modern hardware. This approach is known as hardware-algorithm co-design.

This paradigm has generally been embraced in ML/AI: there are several examples of specialized hardware for AI, not to mention GPUs, which have emerged primarily due to the large market size of AI and its rapid growth[194]. Adapting to modern hardware, a large number of ML algorithms enjoy high arithmetic intensity, use multiple levels of precision to use resources more efficiently, and strive to reduce their communication bottlenecks. Sometimes, these algorithmic adaptations result in loss of accuracy in a classical sense. However, many ML/AI algorithms may be able to tolerate this loss of accuracy, as statistical accuracy may be sufficient for many applications. One may observe that on the one hand, ML/AI are adapting to technology trends in modern hardware; on the other hand, hardware manufacturers are influenced by

the large market size of ML/AL. This has led to a virtuous cycle, resulting in growing number of new hardware products that are more suited for ML/AI than other applications. We refer to the insightful works by Reed et al. (2023) and Thompson and Spanuth (2021) for detailed perspectives on how big tech companies are influencing the computing industry.

Embracing this paradigm in scientific computing is more challenging. Scientific computing has a smaller market size with a modest projected growth, which impacts R&D investments. Moreover, traditionally, scientific computing has dealt with calculations that were expected to be accurate in a classical sense, where consequences of inaccuracies could be significant. This makes algorithmic relaxations more challenging for many scientific computing applications. Nevertheless, technology trends in hardware cannot be ignored, and scientific computing is responding in several ways. Some examples include: (a) growing number of scientific computing applications that run on GPUs; (b) active research on algorithms that better adapt to modern hardware, such as high-order discretizations, mixed-precision calculations, and communication-reducing algorithms; (c) exploring ways to incorporate ML-based techniques in traditional scientific computing frameworks; (d) designing specialized hardware for certain applications; and (e) running scientific computing applications on specialized hardware that was primarily designed for ML. These trends are expected to continue. Some options, such as hardware specialization, may become less costly due to the emergence of open-source, automated design tools.

Not all applications can follow the main technology trends, either due to inherent complexities within their algorithms, or lack of needed resources for software modernization. These applications will continue to run on CPUs, and will likely benefit from higher clock frequencies, and larger on-chip (and on-package) memory capacity of certain multi-core CPUs.

Energy efficiency will be the central theme in hardware design for the years to come. While computer designers strive to improve energy efficiency, the quest to run more, larger, and more detailed models is resulting in larger computing centers with a growing energy demand. In some cases, this level of growth in energy demand was not anticipated when large computing centers were built. Therefore, their expansion may necessitate upgrades in the power grid infrastructure. Large computing centers in the future may even rely on small modular nuclear reactors as a reliable source of energy.

Access, or lack of access to cutting-edge technology will impact computing for the years to come, and will likely lead to more diversity. For instance, when access to advanced process node technology is regulated, performance may rely more on hardware specialization. Sunway Taihulight (Table 8) is a notable example. The future of scientific computing is inevitably more diverse and more challenging than its past. This provides new opportunities to computational scientists.

Frequently asked questions

We end the paper by answering several questions that are often asked by managers, decision makers, and practitioners:

> What are some quick options to improve the performance of a large and old software that took decades to develop?

The answer requires investigating whether the old software is highly parallelizable, how much it may benefit from parallelization, and how complex are its control flows.

If the software does not gain much from parallelization and has complex control flows, using modern low-core-count CPUs might lead to the fastest runtime, as it allows higher clock frequency, and has higher on-chip memory per core.

If the application scales well on parallel systems but has complex control flows, a modern multi-core CPU may enable the fastest runtime. Specifically, if the application involves many intermediate computations and data, a low-core-count CPU may work best, as it provides more on-chip memory per core. Alternatively, if the application has few intermediate calculations and needs little data to operate on, a high-core-count CPU could be more appropriate. Using modern high-core-count CPUs and turning off some cores to provide more on-chip memory per active core is also an option if needed.

If the application parallelizes well, and has simple control flows, migration to GPUs may provide more speedup compared to using multi-core CPUs. It could be costly and time-consuming, and thus performance gains need to be large enough to justify the effort.

In all these cases, using high-bandwidth memory (HBM) will likely make the application run faster, as it improves on-package bandwidth, and increases memory capacity near the compute units. Lastly, if inter-node communication is the main bottleneck, partitioning the problem according to the network topology, or using a network interconnect that has a higher bandwidth would likely help.

> Inter-node communication is limiting the performance of an important application. Can the bandwidth be increased by adding more (InfiniBand) “cables” between the nodes?

Inter-node bandwidth can be increased by adding more NICs to each node and connecting them to the network. This forms a multi-rail network (Coll et al., 2001), and has been used in high-end clusters that run multi-node machine-learning applications.

This approach faces three main challenges: (a) it is very expensive; (b) since processors are pin-limited (*Part I: Off-chip interfaces and pin limitations* (Hanindhito et al.,

2026)), there is a limit on how many NICs can be placed on each node; and (c) there is a limit on how much bandwidth a network switch can handle, which translates to how many network endpoints the switch can serve. With each node needing multiple connections, the number of switches needed will skyrocket. Not only this is expensive, it also necessitates using higher-order level switching that imposes certain topologies (*Network topologies*). This also makes it more difficult for communication libraries to optimize communication patterns.

It is possible for the network topology and configuration to be reasonably specialized in order to provide higher bandwidth to a certain group of applications. We consider two cases to further clarify this point. In the first case, most of the jobs on a cluster may not be extremely large, and only need a few nodes to run. This is a common situation for many applications in industry and academia. For instance, full-waveform inversion for oil and gas exploration falls into this group. To benefit these applications, the network can be designed to provide higher bandwidth and low latency communication for the nodes in the same rack through using top-of-the-rack switches. However, the job scheduling algorithm must carefully allocate the nodes on the same rack. The small-multi-node simulations then may be restricted to run on compute nodes within the same rack. Spine and super-spine switches are then used to connect all top-of-the-rack switches. However, they are often over-subscribed, and thus, bandwidth between racks may not be as high as bandwidth within a rack. The second case involves specializing the network topology to the communication pattern of a specific problem in order to limit inter-node communication bottlenecks. This option may be considered when a cluster is primarily used to run a single application: an unlikely situation in academia, while possible in some industrial applications. Molecular dynamics (MD) simulations for drug discovery by the pharmaceutical industry, and solving PDEs through domain decomposition fall into this group. In this case, each node may be connected to its six neighbors to form a 3D torus network topology. This topology follows the pattern that atoms interact with each other in the physical space, and therefore, minimizes communication bottlenecks. The Anton specialized computing system for MD simulations uses this network topology. The downside of this design is that the cluster could become much slower for applications that do not follow this communication pattern.

How to choose the most suitable computing platform (CPU, GPU, ASIC) for a specific application?

The selection of the hardware platform depends on a variety of factors, such as productivity or development rate, cost, flexibility, performance constraints and/or power budgets. CPUs are great for software development, and algorithmic debugging and exploration, since they are easy to program and fully flexible. However, they may not be an ideal choice for a competitive end-product, due to their high inherent programmability overheads, which greatly limit their attainable throughput and energy efficiency compared to other alternatives.

GPUs are a better alternative for some applications, capable of outperforming CPUs by an order of magnitude. CPUs remain suitable for tasks within the workload that lack parallelization and entail complex control flows. Conversely, GPUs excel in handling the most computationally intensive and parallelizable kernels. However, despite their programmability, developing a GPU code is more challenging than CPUs, typically demanding a profound comprehension of the underlying architecture. Furthermore, GPUs come at a higher cost than CPUs, hence amplifying the budgetary needs for constructing a computing cluster.

The effectiveness of specialized hardware (ASICs) is contingent upon the specific deployment circumstances. Established computing kernels intended for deployment with minimal modifications over a long time-frame and adaptable across diverse applications are ideal candidates for ASIC development. Nevertheless, it is an expensive, protracted, and complicated endeavor. It necessitates the recruitment of highly skilled teams and individuals, along with substantial investments of time and capital, consequently elevating the total cost of ownership significantly. This aspect often dissuades non-tech companies from pursuing ASIC development. A comprehensive analysis, weighing all available alternatives, is imperative before committing to the design of specialized hardware.

What are some key issues that should be investigated prior to developing specialized hardware for a specific application (e.g., solving PDEs)? What kind of performance improvements are expected?

First and foremost, an extensive performance analysis of the algorithm is required. What are the parallelizable portions of the code and what percentage of the total runtime do they constitute for? Do data dependencies limit parallelism and what is the expected runtime on a proposed hardware architecture (estimated via analytical modeling)? What are the communication bandwidth requirements to allow efficient utilization of the compute resources?

Next, the range of applications that the hardware should support must be considered, along with the evolution rate of the algorithms. These aspects will dictate the selection of architecture alternatives and how programmable the computation engines need to be. It would be hard to justify investing in a multi-year project that would likely cost tens of millions of dollars to develop a chip for an application considered outdated soon after it is manufactured.

Putting together a system that includes a hardware accelerator, the necessary peripherals and an extensive software stack necessitates the recruitment of expert teams comprising computer architects, hardware

designers, silicon engineers, software engineers etc. Different teams will work on different aspects of chip development, such as RTL design, verification, physical layout and the software stack, among others. Finally, the selection of the manufacturing technology (e.g. TSMC, Samsung, Intel) and process will play a crucial factor in the device's area footprint, performance and power envelope. Thorough verification and debugging through simulation are essential before tape-out. Any bugs or errors discovered post-tape-out lead to a complete repetition of the manufacturing process, substantially increasing manufacturing costs.

Regarding performance benefits, these depend on many factors, such as the size of the chip, the technology process, application characteristics and the architecture. In general, an ASIC can achieve one to two orders of magnitude higher performance compared to CPUs, and significantly outperform the GPU as well, unless the application is highly memory-bound, in which case the GPU may have a similar performance. This holds true in cases where the problem size can fit in one node. Large-scale problems that require multi-node deployment may be greatly limited by inter-node communication, thus negating the performance benefits of the specialized hardware.

An application-specific chip can also be orders of magnitude better than CPUs and GPUs in terms of energy efficiency, depending on the application characteristics mentioned above. However, some of its performance and efficiency must usually be sacrificed to enable some degree of programmability in order to support an adequate range of applications.

> Does the end of Moore's law mean we will no longer be getting faster chips?

In the broader sense, the end of Moore's law does not necessarily mean chips will not get faster anymore. Instead, the traditional methods of doubling transistor density every 2 years, at the same cost, as predicted by Gordon Moore in 1975, are no longer sustainable due to the physical and technological limitations (*Part I: Transistor size and Moore's law* (Hanindhito et al., 2026)). Therefore, chip manufacturers are exploring alternative techniques in materials, lithography, advanced packaging, and architecture, to continue improving chip performance. This has increased the cost of designing and manufacturing the chips (*Part I: Cost of designing and manufacturing semiconductors* (Hanindhito et al., 2026)).

While specialized chips and hardware accelerators can enable significant performance improvements, general-purpose microprocessors will no longer see a significant jump in individual core's performance. Improvements in CPU performance will primarily be due to more parallelism, enabled by using more cores. However, this will result in higher power consumption as well as higher manufacturing costs.

> Why should one be concerned about the increasing power consumption of modern chips if electricity bills are a small fraction of their expenses?

Aside from the increase in electricity bills, which may not be too significant in itself, the increasing power consumption of modern chips creates two major challenges: (a) supplying power to them becomes more difficult; and (b) cooling them becomes increasingly more challenging and expensive. Moreover, the growing energy demand of modern chips sometimes requires upgrading the power infrastructure for cluster-sized systems. Lastly, computing platforms are increasingly becoming a major portion of the global energy consumption. Therefore, taking merit in optimizing for computing energy efficiency becomes even more important.

Acknowledgements

Arash Fathi and Dimitar Trenev are grateful to ExxonMobil for supporting this work, and permitting its publication. We would like to thank Laurent White for stimulating conversations and commenting on an earlier draft of this paper, which improved its quality. We are also grateful to Amir Gholami, Arben Jusufi, Ardavan Pedram, Brent Wheelock, Chirath Neranjena, Dakshina Valiveti, Dimitri Papageorgiou, Rahul Sampath, and Wenting Xiao, for insightful conversations and feedback.

ORCID iDs

Bagus Hanindhito https://orcid.org/0000-0002-8485-581X
Arash Fathi https://orcid.org/0000-0002-6809-9815
Dimitrios Gourounas https://orcid.org/0000-0002-5011-9646

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by ExxonMobil Research and Engineering Company, agreement number EM10480.36, Division of Computing and Communication Foundations (1763848) and National Science Foundation (NSF) grant numbers 2326894 and 2425655.

Declaration of conflicting interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Disclaimer

Any opinions, findings, conclusions, or recommendations are those of the authors and not of the sponsors.

Notes

1. e.g. a node with many CPUs, or a node with many GPUs, or a node with a lot of memory.
2. Not to be confused with a flat memory model, or a linear memory model, in which applications see a single, continuous

address space of system memory. The system memory may still consist of multiple types of memory technologies (Crandall and Chong, 2004; McAuley and Francis, 1993; Ortiz, 2001; Villela, 2020). Examples include the "flat-mode" on Intel Xeon Phi Knights Landing and the "HBM flat-mode" on Intel Xeon Sapphire Rapids with HBM.

3. Data that is being processed.
4. Data that is not being processed.
5. Other variations of SRAM include 4T, 5T, 8T, and 10T.
6. e.g. adder and multiplier.
7. For instance, each Golden Cove core inside the Intel Xeon Sapphire Rapids (2023) has 508 × 512-bit, and 320 × 256-bit registers, for a total of 42,752 bytes. This is already a 32% increase from the Sunny Cove core of Intel Xeon Ice Lake (2021), with 32,256 bytes registers, and 91% increase from the Skylake core inside Intel Xeon Cascade Lake (2020), with 22,272 bytes registers.
8. Intel Sapphire Rapids has 42,752 bytes of register in each Golden Cove core for a total of 2505 kB of registers in 60-core variants. On the other hand, the NVIDIA H100 GPU has 256 kB of register per SM, for a total of 33 MB of register files across 132 SMs.
9. e.g. the Streaming Multiprocessor (SM) inside the GPU can handle up to 1024 concurrent threads, running in parallel per thread block, compared to 1 or two threads running in parallel in a CPU core.
10. e.g. cache-friendly (Inoue and Taura, 2015) or cache-oblivious (Bader and Zenger, 2006) algorithms.
11. This is called compulsory cache miss, which happens when the memory address for the data is accessed for the first time (Yu et al., 2001).
12. Eviction happens when the data is not used for a while, and the cache needs to make room for new, recently-used data. Each cache implementation has its own replacement policy (Olanrewaju et al., 2016), such as Random Replacement (RR) (Panda et al., 2016), First-in First-out (FIFO) (Panda et al., 2016), Least Recently Used (LRU) (Wang et al., 2019), and Least Frequently Used (LFU) (Matani et al., 2021). The data that is evicted from the first-level cache can still be accessed from higher-level caches, which is still better than accessing from off-chip memory, in terms of energy consumption and performance. If the data is not used for a longer period of time, it is evicted from the last-level cache, and should be accessed from off-chip memory (Mazumdar et al., 2021), if needed.
13. This can be a conflict miss or capacity miss (Tao and Karl, 2006). The conflict miss occurs when the recently-used data is mapped and stored in occupied cache space (Pratoomtong and Hu, 2005). This type of miss happens for direct-map cache, and set-associative cache. If the cache is already fully associative, but cache miss still occurs, then it is called capacity miss, which indicates the cache is too small for the running application.
14. Some strategies rely on working on a subset of a large data at a time, known as tiling. The small data-subset then fits in the cache hierarchically. GotoBLAS for GEMM is a notable example (Goto and Van de Geijn, 2008).
15. Picture cache as a bookshelf: the larger the bookshelf, the longer it takes to find a book.
16. e.g. DRAM.
17. For instance, AMD EPYC Milan-X with 3D-stacked SRAM chiplet has 768 MB of last-level cache, which benefits applications that require larger working memory (e.g., due to large number of intermediate results and operands).
18. Except during the branches, instructions are accessed sequentially. This provides great temporal locality. With a separate structure, the instruction cache, can better serve the spatial locality, which is inherent in the instruction sequence. Data access patterns, on the other hand, vary between applications, and do not necessarily have good spatial locality.
19. Generally, this has not caused issues for two reasons: (a) the SIMT model executes the threads in a lock-step manner with little to no data dependency between the SM/CU/SLC (Ren and Lis, 2017); and (b) users must synchronize the threads to avoid race conditions and maintain the correctness of the operations, either through explicit synchronization (e.g., barriers or fences), or implicit synchronization (e.g., at the kernel boundaries) (Alsop et al., 2016).
20. e.g. NVIDIA Fermi, NVIDIA Volta, NVIDIA Ampere, and NVIDIA Hopper.
21. WSE does not have off-chip memory.
22. DRAM can be implemented on-chip, as embedded DRAM (eDRAM) (Diodato, 2000; Keitel-Schulz and Wehn, 2001), which then enables higher bandwidth and lower latency compared to off-chip DRAM, at a significantly higher cost-per-bit. This is messy because logic circuits and DRAM have different manufacturing trade-offs and methods to achieve their optimal structure (Iyer and Kalter, 1999; Wehn and Hein, 1998), such as: (a) to accommodate increased voltage during read and write, DRAM gate dielectric is thicker; however, the logic circuit gate dielectric is thinner to ensure maximum switching performance; and (b) DRAM technologies favor a single gate material (usually n-type) to reduce cost, at the expense of sub-optimal switching performance; however, logic circuit uses both p-type and n-type (CMOS) to maximize switching performance.

Manufacturing embedded DRAM by using logic-process will need modifications to the process flow, which then increases the manufacturing complexity by 25% (Iyer and Kalter, 1999). eDRAM can benefit applications that require high-capacity, on-chip storage, which becomes too expensive to be implemented via SRAM. Examples include an integrated graphics processing unit (iGPU) that shares the same die with a general-purpose microprocessor (e.g., Intel Iris Pro Graphics, on select Intel Haswell, Broadwell, Skylake, and Coffee Lake microprocessors (Doweck et al., 2017; Hammarlund et al., 2014; Hamzaoglu et al., 2014; Hamzaoglu et al., 2015)), or very large, last-level DRAM cache in some microprocessors (e.g., IBM POWER9 (Sadasivam et al., 2017)).

23. In DRAM terminology, density is the number of bits that can be stored in a DRAM chip. Multiple DRAM chips can be combined to produce larger capacity DRAM modules.
24. For instance, in a DRAM chip with a bit capacity of 32 Gb (1x nm process node technology), the periodic refresh accounts for 30% loss in system performance, and over 20% in higher energy consumption (Bhati et al., 2016).
25. There exist other DRAM implementations, such as 2T1C and 4T2C (Rodríguez-Ibarra et al., 2022).
26. While the industry uses *density* to refer to the number of bits on a memory chip, we prefer to use *capacity* since it is more descriptive, and provides more clarity to our targeted audience.
27. This may contain standards that have not been approved by the Joint Electron Device Engineering Council (JEDEC).
28. The memory row is also called a page, and the process of selecting memory row is called "opening a page", just like opening a page in a book.
29. FPM DRAM (Saito et al., 1985) allows the row address to be held constant, making the sense amplifiers hold the selected row over multiple data accesses on the same row, greatly reducing the overhead of opening a row of memory and improving the bandwidth of the memory for consecutive data access on the same row (Cuppu et al., 1999a). When the next requested data is not on the same row, the lengthy process of opening the row happens with additional overhead called precharge (Alakarhu and Niittylahti, 2002; Rohbani et al., 2021), which sets the voltage level of the bit lines to the required level for the sense amplifier. Precharge is basically closing the page, which will invalidate output data, and thus it must be done after the requested data is read.
30. EDO DRAM (Mormann, 1996) minimizes the precharge latency by adding buffers (latches) to the output of the DRAM to hold the requested data, allowing the DRAM to precharge sooner without affecting the requested data (Cuppu et al., 1999a). The subsequent version, called Burstable EDO (BEDO), added a counter to drive the column selector.
31. Memory controller is a logic circuit that handles transactions (e.g., read, write, refresh) between the microprocessor and DRAM (Lee et al., 2008; Mirosanlou et al., 2020). Before the 2010s, the memory controller was implemented as part of the North Bridge chip, which managed communications between a microprocessor and high-speed peripherals, such as memory (DRAM) and graphics accelerator (GPU) (Radhakrishnan et al., 2007; Solihin et al., 2003). Improved transistor density allowed integration of the North Bridge functionality (including the memory controller) into modern microprocessors (Conway and Hughes, 2007; Owen and Steinman, 2008), eliminating the need of using a separate chip.
32. e.g. due to DRAM design and manufacturing process variations.
33. i.e. clock cycle.
34. i.e. pipelining (Yoo, 1997; Yoon et al., 1999a).
35. i.e. interleaving (Bondurant, 1994; Hur et al., 2019).
36. 64 (data lanes) × 66.67 Mb/s (pin transfer rate) = 533 MB/s (module bandwidth).
37. 2 (channels) × 1066 MB/s (channel bandwidth) = 2132 MB/s (aggregate bandwidth).
38. With reduced voltage, the amount of power needed to read, write, and refresh each bit cell is also reduced. Using the same power budget per DRAM chip, more bit cells can be packed to deliver higher bit capacity per chip (Bhati et al., 2016; Itoh, 2008).
39. i.e. ability to read and write two words.
40. 64 (data lanes) × 400 Mb/s (pin transfer rate) = 3200 MB/s (module bandwidth). Module with this theoretical bandwidth is identified as DDR-400 PC-3200.
41. At the same bus clock.
42. To enable higher bandwidth for memory, the bus clock frequency increased significantly from SDR to DDR5. On the other hand, the internal clock, which drives the memory arrays inside DRAM chips, remained between 100 MHz and 500 MHz to limit power consumption, especially as the bit capacity per chip was increased. The wider gap between the internal clock and the bus clock increases memory access latency, which is defined as the time required after receiving the read command to get the first requested data (Fujisawa et al., 2007).
    Column Access Strobe (CAS) Latency, or simply CL, is the metric that indicates the latency (in bus clock cycle) of a given DRAM module. SDR and DDR modules can have a CL of three clock cycles, while DDR2, DDR3, DDR4, and DDR5 can have a CL as high as 6, 11, 18, and 40 clock cycles, respectively. To find the true latency, measured in nanoseconds, the CL is multiplied by the bus clock period (i.e., 1/bus clock frequency) of a given memory module. For instance, a DDR3 module that has a bus clock of 667 MHz, and a CL of 9, will have the same true latency as a DDR4 module, which has a bus clock of 1333 MHz and a CL of 18, corresponding to 13.50 ns (i.e., 1/667 × 9 = 1/1333 × 18 = 13.50). While both have the same true latency, the DDR4 has double the bandwidth of DDR3.
43. Module with this theoretical bandwidth is identified as DDR2-1066 PC2-8500.
44. 1.35 V for DDR3L (Ji et al., 2010).
45. Typically 32-bit (Gomony et al., 2012).
46. Multi-ported DRAM allows multiple reads or writes at nearly the same time. In VRAM, there are two sets of ports: the first ports are used by the host CPU (or graphics accelerator) to access VRAM, in the same way DRAM does. The second ports are used by the video engine (e.g., digital-to-analog converter) to read images and then output them to the screen, through a high-throughput synchronous serialized data.
47. i.e. smaller memory arrays due to reduced available space to accommodate larger periphery components (e.g., sense amplifiers, decoders) (Dunning et al., 2009).
48. GDDR2 was a mid-point between DDR and DDR2. It was announced in 2002 before DDR2 specification was finalized in September 2003. The first GPU that used GDDR2 was NVIDIA GeForce FX 5800, which was launched in March 2003.

49. In order to realize the double data rate (DDR), GDDR, GDDR2, GDDR3, and GDDR4, use both edges of the clock (i.e., rising and falling edges) to read or write the data. GDDR5 uses the clock (CK) to generate two forwarded clocks (WCK) that have twice the frequency of the CK: the first WCK and CK share the same phase, whereas the second WCK is delayed by half of the period of CK. Consequently, read and write operations can be performed during the rising edge of the first WCK, or the rising edge of the second WCK, realizing the double data rate (DDR).
    On the other hand, GDDR5X and GDDR6 use the clock (CK) to generate four forwarded clocks (WCK) that have twice the frequency of the CK. The first WCK and CK share the same phase; the second WCK is delayed by one-quarter of the period of CK; the third WCK is delayed by half of the period of CK, and the fourth WCK is delayed by three quarters of the period of CK. Consequently, read and write operations can be performed during the rising edge of all WCKs, realizing the quad data rate (QDR).
50. This has not been standardized by JEDEC.
51. i.e. 1 TB/s versus 336 GB/s.
52. i.e. PAM-4 modulation instead of NRZ modulation.
53. Junction temperature is the temperature of the semiconductor material that forms the transistors. It is the sum of the ambient temperature and the temperature increase due to power dissipation (Weste and Harris, 2010).
54. The maximum safe junction temperature is 110°C before the chips automatically throttle back to protect themselves. The absolute junction temperature that must not be reached is 120°C, which immediately degrades the chips. This rating is somewhat higher than previous GDDR5, GDDR5X, and GDDR6 chips, whose junction temperature should not reach 100°C.
55. The printed circuit board (PCB) is usually manufactured by using epoxy (Ehrler, 2002), which is a dielectric material. Parasitic capacitance is formed between two metal wires (either located on the same layer, or different layers of PCB) and the dielectric material, whose effect becomes more severe at higher frequency (Genc et al., 2020; Swirbel et al., 1999).
56. That connect GDDR to GPU.
57. It was first used in AMD Fury X GPUs (Hu et al., 2018; Macri, 2015).
58. The MCDRAM on Knights Landing has three modes of operation that can be chosen: (a) cache-mode, where MCDRAM acts as hardware-managed cache for the off-chip Dynamic Random Access Memory (DRAM); (b) flat-mode, where MCDRAM acts as regular memory, by occupying the same memory space as DRAM; it is then up to the programmer to determine which data goes into MCDRAM to take advantage of its high bandwidth (i.e., software-managed); and (c) Hybrid-mode, which is a combination of cache-mode and flat-mode, where a portion of MCDRAM is used as cache while the rest is used as regular memory (Sodani, 2015).
59. Intel Xeon Max (Sapphire Rapids with HBM) features 64 GB of on-package HBM2E. This HBM can be configured in three ways: (a) HBM-only mode, which is intended to be used in workloads that fit in the HBM; this provides 1 to 2 GB of memory per core, and allows the system to operate without any DDR DIMMs installed; (b) HBM flat mode, which provides access to a larger DDR memory, and requires the programmer to explicitly control the use of HBM and DDR to manage bandwidth and capacity, respectively; and (c) HBM cache mode, which makes the HBM to act like hardware-managed cache for DDR (Biswas, 2021).
60. i.e. between 19 nm and 10 nm.
61. Samsung, Micron, and SK-Hynix.
62. For instance, moving 256 bits of data by 10 mm, consumes more energy than performing a double precision floating point operation (Keckler et al., 2011; Kestor et al., 2013).
63. e.g., compiler, library, framework.
64. Not to be confused with Data Processing Units for inter-node communication.
65. Each memory chip contains 8 DPUs. A memory module (DIMM) may contain 16 memory chips for a total of 128 DPUs. A CPU can have 8 memory channels, each with 2 DIMMs, for a total of 2048 DPUs.
66. i.e. DIMMs.
67. e.g. phase-change memory, resistive RAM, and spintronic RAM.
68. e.g. modulate analog input signals into weighted analog output signals.
69. i.e. boolean algebra, such as NOR.
70. NAND flash arranges the memory cells in series, by sharing the bit line and source line (i.e., they are wired just like the transistors inside the NAND CMOS gate) through two select transistors (i.e., ground select transistor and bit line select transistor), as shown in the bottom-right side of Figure 1 (Weste and Harris, 2010). This arrangement enables a smaller cell size, and lower die cost, at the expense of slower read performance due to having a smaller read current (i.e., due to the series connection) (Crippa et al., 2008). Fortunately, the series arrangement makes writing faster, since many bits can be programmed in parallel due to Fowler-Nordheim tunneling (Compagnoni et al., 2007).
71. NOR flash arranges the memory cells in parallel, by connecting them into a common drain as the bit line (i.e., they are wired just like the transistors inside the NOR CMOS gate), allowing fast individual reading on each cell (Kim et al., 2022b). Although the read performance of NOR flash outperforms that of the NAND flash (Weste and Harris, 2010), this structure provides lower density compared to NAND flash.
72. In memory, a cell is the smallest storage unit, where it stores the bit value as an electrical charge. In 1T1C DRAM (Yin et al., 2019), a cell consists of one transistor and one capacitor whereas in flash memory, a cell consists of one floating-gate transistor (Pavan et al., 1997).
73. Examples include USB flash drives, SD cards, universal flash storage (UFS), and solid-state drives (SSD). SSD has become popular in consumer and enterprise markets, thanks to its significantly-fast read and write performance, while consuming less power, compared to a magnetic drive.

74. e.g. polycrystalline silicon (Bi, 2019).
75. and NOR Flash.
76. e.g. thinner oxide layer, which creates more leakage, and is more likely to breakdown when subjected to high voltage during reading; smaller stored electrical charge, which makes sensing more difficult; and interference between neighboring cells (Lu, 2012).
77. i.e. floating gate transistors, insulated by oxide layers.
78. e.g. silicon nitride (Bi, 2019; King et al., 2001).
79. e.g. more sensitive voltage sensing, due to more levels of voltage that need to be detected.
80. Both PLC and HLC have a considerably low cost-per-bit, sometimes comparable to the magnetic-based storage. Accordingly, they become suitable for implementing large capacity storage systems that have fewer write operations. The latter makes endurance less concerning.
81. There are several types of 3D memory architectures. Examples include Bit Cost Scalable (BiCS) (Tanaka et al., 2007), Pipe-shaped Bit Cost Scalable (P-BiCS) (Katsumata et al., 2009), Terabit Cell Array Transistor (TCAT) (Jang et al., 2009), Vertical Stacked Array Transistor (VSAT) (Kim et al., 2009a), and Vertical Gate (VG) (Kim et al., 2009b; Silvagni, 2016).
82. There are three types of NVDIMM, standardized by JEDEC: NVDIMM-F, NVDIMM-N, and NVDIMM-P NVDIMM-F is a fully flash-based, memory DIMM, without any DRAM chips, resulting in higher density, but lower performance than DRAM. NVDIMM-N uses both DRAM chips and flash chips on the same DIMM, where only DRAM chips are accessible (i.e., mapped to system memory). It is used to allow for data back-up from DRAM to flash. In the event of power failure, it preserves the data, and cuts the time that is needed to restart the system. NVDIMM-P uses both DRAM and flash as memory (e.g., in the Intel Optane DCPMM), where both DRAM and flash are accessible (i.e., mapped to system memory).
83. The mix between DRAM DIMMs and NVDIMMs can be configured in three modes: Memory mode, App Direct mode, and Dual mode (Hildebrand et al., 2021; Waddington et al., 2019). The Memory mode uses NVDIMM to extend memory capacity, and utilizes DRAM as cache. The App Direct mode provides direct access from applications to NVDIMMs, allowing applications to manage the use of NVDIMMs, just like regular SSDs (Weiland et al., 2019). The Dual mode allows for mixed usage of NVDIMMs, so they can be used for both Memory mode and App Direct mode.
84. The Intel Optane DCPMM 100 series (codenamed Apache Pass) was launched in 2019, with 128 GB, 256 GB, and 512 GB capacity per DIMM, at up to 2666 MT/s transfer rate for the Intel Xeon Cascade-Lake CPUs with six memory channels. Its successor, Intel Optane DCPMM 200 series (codenamed Barlow Pass), was launched in 2020 with 128 GB, 256 GB, and 512 GB capacity per DIMM, at up to 3200 MT/s transfer rate for Intel Xeon Cooper Lake CPUs with six memory channels, and Intel Xeon Ice Lake CPUs with 8 memory channels. Its subsequent version, Intel Optane DCPMM 300 series (codenamed Crow Pass), was announced in January 2023, with 128 GB, 256 GB, and 512 GB capacity per DIMM, at up to 4400 MT/s transfer rate for Intel Xeon Sapphire Rapids CPUs with 8 memory channels. In the same month, Intel decided to cancel the Intel Optane DCPMM 300 series product lineup, marking the end of NVDIMM support in Intel CPUs. There are three reasons related to the decision: (a) Micron (co-developer of 3D XPoint) sold its 3D XPoint manufacturing facility in 2021, effectively ending the production of 3D XPoint technology; (b) DDR5-SDRAM chip has increased density, compared to the previous DDR4-SDRAM, allowing the implementation of 512 GB and 1 TB capacity per DIMM, which exceeds the single DIMM capacity offered by NVDIMM; and (c) the Compute Express Link (CXL) protocol allows attachment of memory expansion devices over PCI Express 5.0 interface, eliminating the need of using specialized memory architectures, such as 3D XPoint.
85. The highest capacity for DDR4 DIMM is a 256 GB Load-Reduced Registered DIMM (LRDIMM). With six memory channels and two DIMMs per channel, the maximum available memory can be 3 TB when all DRAM DIMMs are used. With a mix of six 256 GB LRDIMMs, and six 512 GB NVDIMM-Ps of Intel DCPMM, the maximum available memory becomes 4.5 TB. This can especially be useful for big data applications (Hayot-Sasson et al., 2020; Shanbhag et al., 2020).
86. e.g. NAND flash and hard drive.
87. $Ge_2Sb_2Te_5$.
88. e.g. optical readout in electrically-driven PCM.
89. A memristor in its original form is a two-terminal, non-linear component that relates electric charge and magnetic flux.
90. Either organic (Lee et al., 2010b; Simão et al., 2011; Ye et al., 2012), or inorganic (Bera et al., 2013; Park et al., 2012).
91. i.e. increased thickness.
92. i.e. material with higher magnetic coercivity.
93. e.g. CD, DVD (Kobota, 2004) and Blu-ray (Miyagawa, 2014).
94. e.g. film, video games.
95. Also used to implement logic.
96. By using the Vertical Channel Transistor (VCT) technology.
97. e.g. through compilers.
98. e.g. endurance, high bandwidth, low access latency.
99. e.g. density, power, cost-per-bit.
100. e.g. CPUs, local memories, and accelerators, among others.
101. Tri-state buffers have a small area footprint, but are hard to debug and lead to increased power consumption and wire delay. Therefore, most architectures use AND/OR gate-based or MUX-based implementations.
102. A bursting master requests access to the bus only once for multiple consecutive transactions, thus minimizing arbitration and addressing overheads. Burst transfers can also be pipelined, leading to further performance improvements.
103. In cases where slave components require multiple clock cycles to respond, the bus waits idle and does not accept new requests. The split mode overcomes this bottleneck by allowing to "split" or interrupt the pending transaction and allow other masters to access the bus while waiting.

104. OOO is an advanced extension of the split mode, that allows multiple transactions from different masters to be split by slaves and be in progress simultaneously.
105. Each of these standards defines several sub-standards depending on the system requirements. For example, a popular protocol, Advanced Microcontroller Bus Architecture (AMBA) 2.0, defines three distinct sub-standards: (a) Advanced high performance bus (AHB), a high-performance bus meant to connect high-bandwidth components operating at a high clock frequency, such as microprocessors, on-chip memories, external memory interfaces and DMA engines; (b) Advanced system bus (ASB), a light-weight alternative to AHB that offers high performance without supporting advanced features; and (c) Advanced peripheral bus (APB), which is a simple, low-power standard meant for peripheral components that operate at small frequencies and do not require high bandwidth, such as timers, keyboards and UARTs.
106. e.g. point-to-point, mesh, torus, hypercubes etc.
107. e.g. butterflies, crossbars etc.
108. In circuit-switching, a set of links that make up the transmission path is determined and reserved prior to the transmission itself. Once all links in the path are free, the transaction can be initiated and all packets in the message will take the same pre-determined path. In packet-switching, different packets in a message can be transferred along different routes with varying delays. Packets are forwarded to routers on-the-fly based on router availability.
109. A wide variety of routing algorithms exist, which can be classified into three broad categories: (i) Dynamic versus static routing. In static algorithms, paths between each source and destination are predetermined and fixed during system operation, while in dynamic algorithms the paths are determined at runtime by making decisions based on information, such as network congestion. (ii) Source versus distributed routing, where routing decisions are either made once at the source or in each router along the transmission path. (iii) Minimal versus non-minimal routing, based on whether a constraint of minimizing each transaction's routing length is imposed or not.
110. e.g. Ethernet, InfiniBand.
111. Some asynchronous serial interfaces do not need a clock signal. An example includes the universal asynchronous receiver-transmitter (UART) (Dawoud and Dawoud, 2020). They are usually used with lower data rates, and thus, are not discussed in this paper.
112. For example, PCI Express 4.0 requires more retimers (i.e., signal-conditioning chips) compared to PCI Express 3.0, in order to maintain signal integrity over a long distance. This raises manufacturing costs, and makes the design more complex (EL Sabbagh, 2018; Shah et al., 2022).
113. e.g. 10^{-6} in PCI Express 6.0 with PAM-4, compared to 10^{-12} in PCI Express 5.0 with NRZ (Sharma, 2020b).
114. PCI Express 6.0 uses two primary mechanisms to detect and correct errors: forward error correction (FEC) and cyclic redundancy check (CRC). For every 256 bytes of transmitted data (called flow control unit), 242 bytes are the useful payload, 8 bytes are the CRC protection, and 6 bytes are the FEC protection, and thus the effective bandwidth will be 242/256 of raw bandwidth (Sharma, 2022a).
115. Some interfaces try to minimize the number of lanes that are needed to deliver the same aggregate bandwidth by increasing the bandwidth of each lane (e.g., by increasing the bus clock frequency, or by using a more advanced bus encoding). For instance, PCI Express 4.0 x8 (8 lanes) and PCI Express 5.0 x4 (4 lanes) can deliver the same aggregate bandwidth, where the latter interface uses half of the pins and wires.
116. e.g. PCI Express, AMD InfinityFabric, and NVIDIA NVLink.
117. For instance, the PCI Express (PCIe) interface can be configured with 1 (x1), 4 (x4), 8 (x8), 16 (x16), and 32 (x32 on PCIe 5.0 devices using OCP form factor) lanes depending on bandwidth requirements. Through a feature called bifurcation (link-subdivision) (Yoon and Malone, 2022), a PCIe interface with a higher number of lanes can be split into multiple PCIe interfaces with a lower number of lanes. For example, a 16-lane PCIe interface can be split into two 8-lane PCIe interfaces or four 4-lane PCIe interfaces. This flexibility allows the manufacturer to allocate the available PCIe lanes provided by the microprocessor based on the bandwidth requirements of each device. For example, a microprocessor could provide 64 lanes of PCIe, consisting of four 16-lane interfaces. These 64 lanes can be configured as three 16-lane interfaces to connect three accelerator cards (e.g., GPUs) and four 4-lane interfaces to connect four NVME storages.
118. To reach a longer distance, signals may need to be amplified by using repeater (re-timer and re-driver) chips (Dorren et al., 2015; Tang et al., 2020; Tran, 2023), which add cost, design complexity, and energy consumption.
119. Non-volatile memory express.
120. e.g. driver, library, or application.
121. These devices are classified into three types (Al Maruf and Chowdhury, 2023; Boles et al., 2023; Sharma, 2023a): (a) Type 1 device (e.g., smart NICs) leverages cxl.io and cxl.cache to maintain cache coherency, which enables seamless access of the device to the host microprocessor's memory; (b) Type 2 device (e.g., GPUs, and AI accelerators) has its own dedicated memory (i.e., device memory), and leverages cxl.io, cxl.cache, and cxl.mem to maintain cache and memory coherency between the host and device; and (c) Type 3 device (e.g., add-on memory module, and persistent memory) leverages cxl.io and cxl.mem to allow the host microprocessor to access memory expansion, resulting in increased memory capacity and bandwidth.
122. e.g. OpenMPI (Gabriel et al., 2004), MVAPICH (Panda et al., 2021), MPICH (Gropp et al., 1996), Spectrum MPI.
123. While high-density, CPU-only nodes may have up to eight CPU sockets, most HPC compute nodes have two sockets of CPUs, with additional two, four, or eight accelerator cards (e.g., GPUs, FPGAs, or ASICs).
124. The thermal design power of future CPUs and GPUs is expected to increase as shown by the trend in Part I Tables 6

and 7 (Hanindhito et al., 2026). To efficiently cool the resulting heat, future compute nodes will likely be liquid-cooled, as opposed to air-cooled systems that are common today (Koronen et al., 2020; Pambudi et al., 2022; Zhang et al., 2022). This will increase the overall cost of HPC infrastructure.

125. Some of them use Slingshot and Omnipath, while a small percentage of them use proprietary interconnects.
126. Other competing technologies (e.g., Slingshot and Omnipath) will likely follow InfiniBand, by using the most advanced SerDes, to realize the highest achievable bandwidth at that time.
127. Host channel adapter (HCA) or host bus adapter (HBA) is a specialized chip, usually in the form of an expansion board, which allows the host microprocessor of a compute node to communicate with another compute node through a communication channel (e.g., optical cable, copper cable) using a particular communication protocol (e.g., Ethernet, InfiniBand) (Eddington, 2002).
128. Switch provides a way for a compute node to communicate with many compute nodes by managing the communication channels and the communication traffic between them (Eddington, 2002). Switch does not generate traffic; it only receives and forwards the traffic to the correct destination (Eddington, 2002).
129. InfiniBand cables may be categorized into three groups: passive copper cable (PCC) or direct attach cable (DAC); active copper cable (ACC); and active optical cable (AOC). The PCC or DAC is just a simple copper wire, without any electronics. It provides the lowest cost, lowest latency, and lowest insertion loss. With minimal power consumption for shorter distance, In order to maintain signal integrity, it is typically used for distances less than 10 m, depending on the InfiniBand rate (e.g., 2 m for HDR 200 Gb/s, 5 m for EDR 100 Gb/s, and 7 m for FDR 56 Gb/s).
    As the rate of InfiniBand increases, ACC with its included signal-boosting chip is preferable to achieve adequate distance, while maintaining signal integrity (e.g., up-to 4 m for HDR 200 Gb/s).
    In large and dense clusters, copper cables may not be preferred since they are bulky and heavy, making it difficult to route the cables inside a rack and across racks. In this case, AOC are preferred, as they have significantly smaller form factor, and are able to reach 100 m distance with EDR 100 Gb/s and HDR 200 Gb/s.
130. The data is sent over the network in the form of packets, which have a specific format. The HCA needs to perform packet processing for sending and receiving the data, which needs considerable computational power as the bandwidth increases.
131. Less-demanding infrastructure is defined as the one that does not need the most advanced technology available to provide high bandwidth and low latency. This includes the remote management tools (e.g., IPMI or BMC) (Frazelle, 2020; Kavanagh et al., 2016)) to manage clusters, as well as remote access for users (e.g., SSH, SCP (Amor-Martin et al., 2015; Nakamura and Kuga, 2023)) to access clusters.
132. Instead of the 8b/10b block coding.
133. Spectrum is the product line-up for Ethernet switch from Mellanox, with Spectrum (2015) providing 3.2 Tb/s of switching capacity, Spectrum-2 (2018) providing 6.4 Tb/s of switching capacity, Spectrum-3 (2020) providing 12.8 Tb/s of switching capacity, and Spectrum-4 (2022) providing 51.2 Tb/s of switching capacity.
134. Mainstream clusters are used to handle enterprise workloads that are not compute and communication intensive. Some examples include server virtualization, virtual desktop infrastructure, databases, business applications, web applications, business intelligence, and collaboration applications.
135. Cloud providers typically provide multiple types of instances that can be chosen by customers, based on their workload. Bandwidth-hungry workloads should be provisioned in clusters that provide the highest inter-node bandwidth.
136. e.g. Ethernet, InfiniBand.
137. e.g. TCP/IP offload engine (Ding et al., 2016), and Remote DMA (RDMA) (Guo et al., 2016; Yi et al., 2017).
138. i.e. layer 4 or below.
139. i.e. layer five and above.
140. i.e. layer five and above.
141. Routing, tunneling, telemetry.
142. Either NVIDIA A30 GPU, or NVIDIA A100.
143. e.g. NVIDIA Selene AI cluster.
144. e.g. InfiniBand, Ethernet.
145. e.g. Fibre Channel.
146. e.g. accelerators, memory, storage.
147. e.g. optical and silicon photonics, as well as SerDes.
148. CXL 2.0 standard deals with switching, memory pooling, and fabric management, to support disaggregation of system resources (Sharma, 2020a). CXL 3.0 standard improves the previous switching capability by adding multi-level switching support. This allows devices to perform point-to-point communication without the need for host support, adds memory coherency sharing across devices, and improves bandwidth and latency by using PCI Express 6.0 for the physical layer (Sharma, 2022b).
149. e.g. Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Container-as-a-Service (CaaS), Function-as-a-Service (FaaS), Software-as-a-Service (SaaS), and Data-as-a-Service (DaaS).
150. e.g. NVIDIA's Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) (Graham et al., 2016).
151. e.g. reduce, all-reduce, gather, and all-gather operations.
152. i.e. workloads that are not suitable for the accelerators.
153. e.g. compilers, libraries, frameworks.
154. e.g. NVIDIA CUDA (Buck, 2007), Intel OneAPI (Aktemur et al., 2020), and AMD ROCm (Sun et al., 2018).
155. Arithmetic intensity refers to the number of operations that need to be performed per byte fetched from main memory.
156. Memory-bound workloads are workloads with a relatively low arithmetic intensity and their performance is bounded by the bandwidth to main memory.
157. e.g. exhaustive search and Integer-Linear-Programming (ILP).

158. e.g. hierarchical clustering or list scheduling.
159. e.g. simulated annealing, Kernighan-Lin etc.
160. Conventional clusters consist of compute nodes, each with their own CPUs, memory, storage, and accelerators (e.g., GPUs). However, not all workloads are able to utilize all of the resources provided in each compute node. A class of workloads may only run on CPUs, and need huge amounts of memory, leaving the GPUs idle. Another class of workloads may better run on GPUs. Therefore, designing the right compute node for diverse workloads becomes difficult. Disaggregated architectures provide flexibility and performance, as resources can be assigned dynamically, based on the needs of each workload.
161. i.e. general-purpose processors, memory, inter-node interconnects, power supply, etc.
162. This provides a theoretical peak rate of 25,600 interactions per microsecond (Larson et al., 2008).
163. These interconnects are full-duplex and packet-based.
164. i.e. sub-box paging.
165. With AMD Opteron 250 CPUs.
166. 400 MHz planned versus 485 MHz actual.
167. 1 ICB, 4 GPs, and 8 GCs in Anton 1, for a total of 13 cores versus 66 GCs for a total of 66 cores in Anton 2.
168. 32 in Anton 1 versus 76 in Anton 2.
169. 384 kB in Anton 1 versus 5280 kB in Anton 2.
170. The bandwidth is 224 Gb/s per neighbor in each direction versus 42 Gb/s per neighbor in each direction in Anton 1. The communication system also supports packet filtering, reduction operation, and synchronization capability over the network.
171. The price of owning and maintaining a large computing center, i.e., Total Cost of Ownership (TCO), comprises capital (CAPX) and operating (OPEX) expenditures. CAPX includes purchasing of the servers and the cooling system, whereas OPEX includes maintenance and energy costs. TCO greatly varies based on the location of the datacenter and its type.
172. These machines were used primarily for scientific computing.
173. e.g. CPUs, IO, DRAM, etc.
174. See Moore's law (Part I Section 2.3 (Hanindhito et al., 2026)) and Dennard scaling.
175. e.g. doors, roof, and chassis, each having different materials and stiffness.
176. e.g. lots of branches to manage wetting and drying due to flooding of land.
177. e.g. adjoint-based methods.
178. Due to storing the solution of the PDE at several time instances, referred to as checkpointing.
179. This is due to GPU's limited on-package memory (e.g., HBM), and slow transactions between HBM and off-package memory.
180. Arithmetic intensity of an algorithm is the ratio of floating point operations to the size of data that needs to be moved from a memory unit to the compute unit. For instance, multiplying two matrices has a higher arithmetic intensity than adding them, since the former involves more floating point operations than the latter for similar data movement.
181. Designing efficient and stable high-order discretization methods is very challenging.
182. As opposed to CUDA.
183. e.g. using Intel Math Kernel Library for solving a sparse direct linear system of equations on multi-core processors, even if the application itself is serial.
184. For instance, EPYC 9174F is a 16-core processor (low-core-count) with 4.1 GHz base clock frequency and 256 MB of total L3 cache, which translates to 16 MB of L3 cache per core. On the other hand, the EPYC 9654 is a 96-core processor (high-core-count) with 2.4 GHz base clock frequency and 384 MB of total L3 cache, which translates to 4 MB L3 cache per core. Both processors use the same 5 nm process node technology for their core-complex-die, packaging technology, and power envelopes.
185. e.g. number of nodes and active cores per CPU.
186. e.g. through more funding and employment opportunities.
187. This result is rather theoretical, in the sense that such a network may require a large number of neurons, or may make the optimization process difficult. Accordingly, several layers are often used in practice.
188. Typically, a multivariate Gaussian, characterized by means and variances: $z \sim \mathcal{N}(\mu, \Sigma)$, s.t. $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$, and $\Sigma = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2)$, where n is the dimension of the latent space.
189. An encoder in an autoencoder is primarily tasked with reducing the dimensionality of the input data. In transformers, however, the encoder's primary responsibility is to capture contextual information within the input sequence.
190. e.g. within the spatial domain of a PDE.
191. The FPGA with the highest memory bandwidth (Intel Agilex M-series) has only 1.1 TB/s, while the AMD Mi300a GPU has 5.7 TB/s.
192. Intel and AMD already have HBM-equipped FPGAs that exceed 1 TB/s peak bandwidth, namely Intel Agilex M-series and AMD Versal HBM series.
193. i.e. beyond the traditional CMOS-based technologies.
194. In its recent research (December 2023), the Business Research Company (TBRC) estimated the growth in the hardware market due to AI (The Business Research Company, 2023): it reached $16.8 billion in 2022 with a compound annual growth rate (CAGR) of 16.0% since 2017. It is expected to grow to $48.2 billion in 2027 (23.4% CAGR), and the trend is expected to reach $131 billion (22.3% CAGR) by 2032. Another study by Precedence Research estimated the AI hardware market size to exceed $473.5 billion by 2033 with a CAGR of 24.3% from 2024 to 2033 (Precedence Research, 2024). In mid-2023, Bloomberg Intelligence estimated that the market size of generative AI products, not just hardware, will reach $1.3 trillion in 2032, a 42% CAGR since 2022 with just $40 billion (Catsaros, 2023). All three estimates indicate significant growth in the coming years.

References

Abdennadher S, Altmann M and Xue B (2018) Challenges and emerging solutions in testing HBM IO & systems. In:

2018 IEEE 19th Latin-American Test Symposium (LATS). IEEE, pp. 1–4.

Abdennadher S, Tripician K and Singaravelu S (2020) At speed testing challenges and solutions for 56Gbps and 112Gbps PAM4 SerDes. In: *2020 IEEE Latin-American Test Symposium (LATS)*. IEEE, pp. 1–5.

Abdulah S, Cao Q, Pei Y, et al. (2022) Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with PaRSEC. *IEEE Transactions on Parallel and Distributed Systems* 33(4): 964–976.

Abduljabbar M, Markomanolis GS, Ibeid H, et al. (2017) Communication reducing algorithms for distributed hierarchical N-Body problems with boundary distributions. *High Performance Computing, Lecture Notes in Computer Science* 10266: 79–96.

Abduljabbar M, Farhan MA, Al-Harthi N, et al. (2019) Extreme scale fmm-accelerated boundary integral equation solver for wave scattering. *SIAM Journal on Scientific Computing* 41(3): C245–C268.

Abraham M, Murtola T, Schulz R, et al. (2015) GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2: 19–25.

Abubakar A, Di H, Kaul A, et al. (2022) Deep learning for end-to-end subsurface modeling and interpretation: An example from the Groningen gas field. *The Leading Edge* 41(4): 259–267.

Advani RN (2016) *The Business of NAND, Chapter 1*. Springer Netherlands, pp. 1–28.

Agarwala S, Fuoco C, Anderson T, et al. (2000) A multi-level memory system architecture for high performance DSP applications. In: *Proceedings 2000 International Conference on Computer Design*. IEEE, pp. 408–413.

Agosta G, Fornaciari W, Massari G, et al. (2018) Managing heterogeneous resources in HPC systems. In: *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM'18*. Association for Computing Machinery, pp. 7–12.

Ahmad S, Subramanian S, Boppana V, et al. (2019) Xilinx first 7nm device: versal AI core (VC1902). In: *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE, pp. 1–28.

Ahmed SE and Stinis P (2023) A multifidelity deep operator network approach to closure for multiscale systems. *Computer Methods in Applied Mechanics and Engineering* 414: 116161.

Aiba Y, Tanaka H, Maeda T, et al. (2021) Cryogenic operation of 3D flash memory for new applications and bit cost scaling with 6-Bit per cell (HLC) and beyond. In: *2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE, pp. 1–3.

Aktemur B, Metzger M, Saiapova N, et al. (2020) Debugging SYCL programs on Heterogeneous Intel® architectures. In: *Proceedings of the International Workshop on OpenCL, IWOCL '20*. Association for Computing Machinery, pp. 1–10.

Al Maruf H and Chowdhury M (2023) Memory disaggregation: advances and open challenges. *ACM SIGOPS - Operating Systems Review* 57(1): 29–37.

Alakarhu J and Niittylahti J (2002) A comparison of precharge policies with modern DRAM architectures. *9th International Conference on Electronics, Circuits and Systems* 2: 823–826.

Albergo MS, Kanwar G and Shanahan PE (2019) Flow-based generative models for Markov Chain Monte Carlo in lattice field theory. *Physical Review D* 100: 034515.

Albericio J, Ibáñez P, Viñals V, et al. (2013) The reuse cache: downsizing the shared last-level cache. In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46*. Association for Computing Machinery, pp. 310–321.

Ali Z, Tanveer T, Aziz S, et al. (2022) Reassessing the performance of ARM vs x86 with recent technological shift of apple. In: *2022 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, pp. 01–06.

Alimi IA, Patel RK, Aboderin O, et al. (2021) Network-on-chip topologies: potentials, technical challenges, recent advances and research direction. *Network-on-Chip-Architecture, Optimization, and Design Explorations IntechOpen* 1: 1–23.

Alizadeh M and Edsall T (2013) On the data path performance of leaf-spine datacenter fabrics. In: *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, pp. 71–74.

Alsalibi AI, Mittal S, Al-Betar MA, et al. (2018) A survey of techniques for architecting SLC/MLC/TLC hybrid flash memory–based SSDs. *Concurrency and Computation: Practice and Experience* 30(13): e4420.

Alsop J, Orr MS, Beckmann BM, et al. (2016) Lazy release consistency for GPUs. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO, pp. 1–14.

Amaral M, Polo J, Carrera D, et al. (2017) Topology-aware GPU scheduling for learning workloads in cloud environments. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17*. Association for Computing Machinery, pp. 1–12.

Amer A, Holliday J, Long DDE, et al. (2011) Data management and layout for shingled magnetic recording. *IEEE Transactions on Magnetics* 47(10): 3691–3697.

Amin W, Hussain F, Anjum S, et al. (2020) Performance evaluation of application mapping approaches for Network-on-Chip designs. *IEEE Access* 8: 63607–63631.

Amor-Martin A, Martinez-Fernandez I and Garcia-Castillo L (2015) Posidonia: a tool for HPC and remote scientific simulations [EM programmer's notebook]. *IEEE Antennas and Propagation Magazine* 57(6): 166–177.

Andrade H and Crnkovic I (2018) A review on software architectures for heterogeneous platforms. In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, pp. 209–218.

Ang J, Barrett R, Benner R, et al. (2014) Abstract machine models and proxy architectures for exascale computing. In:

2014 Hardware-Software Co-Design for High Performance Computing. IEEE, pp. 25–32.

Aoyagi A, Stipe B, Wood R, et al. (2022) Helium sealed hard disk drive. *Journal of Magnetism and Magnetic Materials* 564: 170146.

Asri M and Gerstlauer A (2022) CASPHAr: cache-managed accelerator staging and pipelining in heterogeneous system architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41(11): 4325–4336.

Asri M, Dunham C, Rusitoru R, et al. (2020) The Non-Uniform Compute Device (NUCD) architecture for lightweight accelerator offload. In: *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, pp. 38–45.

Asri M, Malhotra D, Wang J, et al. (2021) Hardware accelerator integration tradeoffs for high-performance computing: A case study of GEMM acceleration in N-Body methods. *IEEE Transactions on Parallel and Distributed Systems* 32(8): 2035–2048.

Augonnet C, Thibault S, Namyst R, et al. (2009) StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. In: *Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference*. Springer, pp. 863–874.

Awamoto S, Focht E and Honda M (2020) Designing a storage software stack for accelerators. In: *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. USENIX Association, pp. 1–7.

Aziz PM, Healey A, Liu C, et al. (2011) SerDes design and modeling over 25+ Gb/s serial link. *International Symposium on Microelectronics* 2011(1): 000044–000060.

Bader M and Zenger C (2006) Cache oblivious matrix multiplication using an element ordering based on a Peano curve. *Linear Algebra and Its Applications* 417(2): 301–313.

Bae SJ, Park KI, Ihm JD, et al. (2008) An 80 nm 4 Gb/s/pin 32 bit 512 Mb GDDR4 Graphics DRAM with low power and low noise data bus inversion. *IEEE Journal of Solid-State Circuits* 43(1): 121–131.

Baek S, Cho S and Melhem R (2014) Refresh now and then. *IEEE Transactions on Computers* 63(12): 3114–3126.

Balasubramonian R, Dwarkadas S and Albonesi D (2001) Reducing the complexity of the register file in dynamic superscalar processors. In: *Proceedings 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*. IEEE, pp. 237–248.

Barroso LA, Hölzle U and Ranganathan P (2019) *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. Springer Nature.

Baskaran S, Kandemir MT and Sampson J (2022) An architecture interface and offload model for low-overhead, near-data, distributed accelerators. In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, pp. 1160–1177.

Benini L and De Micheli G (2002) Networks on chips: a new SoC paradigm. *Computer* 35(1): 70–78.

Bera A, Peng H, Lourembam J, et al. (2013) A versatile light-switchable nanorod memory: Wurtzite ZnO on perovskite SrTiO3. *Advanced Functional Materials* 23(39): 4977–4984.

Bhati I, Chang MT, Chishti Z, et al. (2016) DRAM refresh mechanisms, penalties, and Trade-Offs. *IEEE Transactions on Computers* 65(1): 108–121.

Bi J (2019) Radiation effects of Floating-Gate (FG) and Charge-Trapping (CT) Flash memory technologies. In: *2019 International Conference on IC Design and Technology (ICICDT)*. IEEE, pp. 1–3.

Bielak J, Kim EJ and Ghattas O (2005) Parallel Octree-Based finite element method for Large-Scale earthquake ground motion simulation. *Computer Modeling in Engineering and Sciences* 10(2): 99–112.

Biswas A (2021) Sapphire Rapids. In: *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, pp. 1–22.

Blalock T and Jaeger R (1992) A high-speed sensing scheme for 1T dynamic RAMs utilizing the clamped bit-line sense amplifier. *IEEE Journal of Solid-State Circuits* 27(4): 618–625.

Bolchini C, Durelli GC, Miele A, et al. (2015) An orchestrated approach to efficiently manage resources in heterogeneous system architectures. In: *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, pp. 200–207.

Boles D, Waddington D and Roberts DA (2023) CXL-Enabled enhanced memory functions. *IEEE Micro* 43(2): 58–65.

Bolotin E, Nellans D, Villa O, et al. (2015) Designing efficient heterogeneous memory architectures. *IEEE Micro* 35(4): 60–68.

Bondurant D (1994) Low latency EDRAM main memory subsystem for 66 MHz bus operation. In: *Proceedings of COMPCON '94*. IEEE, pp. 250–254.

Boukaram WH, Turkiyyah G, Ltaief H, et al. (2018) Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression. *Parallel Computing* 74: 19–33.

Bourzac K (2017) Has Intel created a universal memory technology? [News]. *IEEE Spectrum* 54(5): 9–10.

Boutros A, Nurvitadhi E, Ma R, et al. (2020) Beyond peak performance: comparing the real performance of AI-Optimized FPGAs and GPUs. In: *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, pp. 10–19.

Bradstock F (2023) Small nuclear reactors: the answer to big tech's energy crisis? OilPrice.com. Available at: https://oilprice.com/Alternative-Energy/Nuclear-Power/Small-Nuclear-Reactors-The-Answer-To-Big-Techs-Energy-Crisis.html

Bragança L, Alves F, Penha JC, et al. (2018) Simplifying HW/SW integration to deploy multiple accelerators for CPU-FPGA heterogeneous platforms. In: *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS '18*. Association for Computing Machinery, pp. 97–104.

Brox M, Balakrishnan M, Broschwitz M, et al. (2018) An 8-Gb 12-Gb/s/pin GDDR5X DRAM for cost-effective high-performance applications. *IEEE Journal of Solid-State Circuits* 53(1): 134–143.

Buck I (2007) GPU computing with NVIDIA CUDA. In: *ACM SIGGRAPH 2007 Courses, SIGGRAPH '07*. Association for Computing Machinery, p. 6.

Burstedde C, Wilcox LC and Ghattas O (2011) p4est: scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing* 33(3): 1103–1133.

Cabrera AM, Young AR and Vetter JS (2022) Design and analysis of CXL performance models for tightly-coupled heterogeneous computing. In: *Proceedings of the 1st International Workshop on Extreme Heterogeneity Solutions, ExHET '22*. Association for Computing Machinery, pp. 1–6.

Caddy B (2022) Memories on magnetic tape. *New Scientist* 256(3417): 66–67.

Castrillon J, Lieber M, Klüppelholz S, et al. (2018) A hardware/software stack for heterogeneous systems. *IEEE Transactions on Multi-Scale Computing Systems* 4(3): 243–259.

Cataldo R, Korol G, Fernandes R, et al. (2016) Architectural exploration of last-level caches targeting homogeneous multicore systems. In: *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, pp. 1–6.

Catsaros O (2023) Generative AI to become a $1.3 trillion market by 2032, research finds. Bloomberg. Available at: https://www.bloomberg.com/company/press/generative-ai-to-become-a-1-3-trillion-market-by-2032-research-finds/

Caulfield A, Costa P and Ghobadi M (2018) Beyond SmartNICs: towards a fully programmable cloud: invited paper. In: *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, pp. 1–6.

Chaitin G (2004) Register allocation and spilling via graph coloring. *ACM SIGPLAN Notices* 39(4): 66–74.

Chan CH, Cheng L, Deng W, et al. (2022) Trending IC design directions in 2022. *Journal of Semiconductors* 43(7): 071401.

Chattopadhyay B, Bhat SN, Nayak G, et al. (2018) A 12.5Gbps transmitter for multi-standard SERDES in 40nm low leakage CMOS process. In: *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*. IEEE, pp. 13–18.

Chaudhuri M, Gaur J and Subramoney S (2019) Bandwidth-Aware Last-Level caching: efficiently coordinating Off-Chip read and write bandwidth. In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, pp. 109–118.

Chen T and Chen H (1995) Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks* 6(4): 911–917.

Chen R, Shao Z and Li T (2016) Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO*. IEEE, pp. 1–12.

Chen J, Huang CK, Delgado JF, et al. (2023a) Generating subsurface earth models using discrete representation learning and deep autoregressive network. *Computational Geosciences* 27(6): 955–974.

Chen WC, Lue HT, Hsu TH, et al. (2023b) A simulation study of scaling capability toward 10nm for the 3D stackable Gate-Controlled Thyristor (GCT) DRAM device. In: *2023 IEEE International Memory Workshop (IMW)*. IEEE, pp. 1–4.

Chernicoff D (2023) Nuclear-powered data centers: modular reactors on the horizon. Data Center Frontier. Available at: https://www.datacenterfrontier.com/data-center-design/article/33002036/nuclear-powered-data-centers-modular-reactors-on-the-horizon

Chirkov G and Wentzlaff D (2023) Seizing the bandwidth scaling of On-Package interconnect in a Post-Moore's Law world. In: *Proceedings of the 37th ACM International Conference on Supercomputing, ICS '23*. Association for Computing Machinery, pp. 410–422.

Chishiro H, Suito K, Ito T, et al. (2019) Towards heterogeneous computing platforms for autonomous driving. In: *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, pp. 1–8.

Chishti Z, Powell M and Vijaykumar T (2005) Optimizing replication, communication, and capacity allocation in CMPs. In: *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, pp. 357–368.

Cho SJ, Ahn J, Choi H, et al. (2012) Performance analysis of multibank DRAM with increased clock frequency. In: *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 2477–2480.

Cho K, Lee H, Kim H, et al. (2015) Design optimization of High Bandwidth Memory (HBM) interposer considering signal integrity. In: *2015 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*. IEEE, pp. 15–18.

Cho JH, Kim J, Lee WY, et al. (2018) A 1.2V 64Gb 341GB/S HBM2 stacked DRAM with spiral point-to-point TSV structure and improved bank group data control. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. IEEE, pp. 208–210.

Chowdhury Z, Harms JD, Khatamifard SK, et al. (2018) Efficient In-Memory processing using spintronics. *IEEE Computer Architecture Letters* 17(1): 42–46.

Christiaens M, De Sutter B, De Bosschere K, et al. (1999) A fast, cache-aware algorithm for the calculation of radiological paths exploiting subword parallelism. *Journal of Systems Architecture* 45(10): 781–790.

Chu CH, Kousha P, Awan AA, et al. (2020) NV-Group: link-efficient reduction for distributed deep learning on modern dense GPU systems. In: *Proceedings of the 34th ACM International Conference on Supercomputing, ICS '20*. Association for Computing Machinery, pp. 1–12.

Chua L (2011) Resistance switching memories are memristors. *Applied Physics A* 102(4): 765–783.

Chun KC, Kim YK, Ryu Y, et al. (2021) A 16-GB 640-GB/s HBM2E DRAM with a data-bus window extension technique and a synergetic On-Die ECC scheme. *IEEE Journal of Solid-State Circuits* 56(1): 199–211.

Coll S, Frachtenberg E, Petrini F, et al. (2001) Using multirail networks in high-performance clusters. In: *Proceedings*

2001 IEEE International Conference on Cluster Computing. IEEE, pp. 15–24.

Compagnoni CM, Spinelli AS, Gusmeroli R, et al. (2007) First evidence for injection statistics accuracy limitations in NAND flash Constant-Current Fowler-Nordheim programming. In: *2007 IEEE International Electron Devices Meeting*. IEEE, pp. 165–168.

Conway P and Hughes B (2007) The AMD opteron northbridge architecture. *IEEE Micro* 27(2): 10–21.

Cosoroaba A (1995) Synchronous DRAM products evolutionize memory system design In: *Proceedings of Southcon '95*. IEEE, pp. 70–72.

Cosoroaba A (1997) Double data rate synchronous DRAMs in high performance applications. In: *WESCON/97 Conference Proceedings*. IEEE, pp. 387–391.

Cota EG, Mantovani P, Di Guglielmo G, et al. (2015) An analysis of accelerator coupling in heterogeneous architectures. In: *Proceedings of the 52nd Annual Design Automation Conference, DAC '15*. Association for Computing Machinery, pp. 1–6.

Crandall J and Chong F (2004) Minos: control data attack prevention orthogonal to memory model. In: *37th International Symposium on Microarchitecture. MICRO-37'04*. IEEE, pp. 221–232.

Crippa L and Micheloni R (2010) *MLC Storage, Chapter 10*. Springer Netherlands, pp. 261–298.

Crippa L, Micheloni R, Motta I, et al. (2008) *Nonvolatile Memories: NOR vs. NAND Architectures*. Springer Berlin Heidelberg, pp. 29–53.

Croci M and Giles MB (2022) Effects of round-to-nearest and stochastic rounding in the numerical solution of the heat equation in low precision. *IMA Journal of Numerical Analysis* 43(3): 1358–1390.

Croci M and Rosilho de Souza G (2022) Mixed-precision explicit stabilized Runge–Kutta methods for single- and multi-scale differential equations. *Journal of Computational Physics* 464: 111349.

Cruz JL, González A, Valero M, et al. (2000a) Multiple-Banked register file architectures. *ACM SIGARCH Computer Architecture News* 28(2): 316–325.

Cruz JL, González A, Valero M, et al. (2000b) Multiple-Banked register file architectures. In: *Proceedings of the 27th Annual International Symposium on Computer Architecture, ISCA '00*. Association for Computing Machinery, pp. 316–325.

Cucchiara R, Piccardi M and Prati A (1999) Exploiting cache in multimedia. *Proceedings IEEE International Conference on Multimedia Computing and Systems* 1: 345–350.

Cui J, Guo W and Yang J (2008) Analysis of DDR3 SDRAM standard technology. *Modern Applied Science* 2(2): 1–93.

Cuomo S, Di Cola VS, Giampaolo F, et al. (2022) Scientific machine learning through physics-informed neural networks: Where we are and what's next. *Journal of Scientific Computing* 92(3): 88.

Cuppu V, Jacob B, Davis B, et al. (1999a) A performance comparison of contemporary DRAM architectures. In: *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*. IEEE, pp. 222–233.

Cuppu V, Jacob B, Davis B, et al. (1999b) A performance comparison of contemporary DRAM architectures. In: *Proceedings of the 26th Annual International Symposium on Computer Architecture, ISCA '99*. IEEE Computer Society, pp. 222–233.

Cuppu V, Jacob B, Davis B, et al. (2001) High-performance DRAMs in workstation environments. *IEEE Transactions on Computers* 50(11): 1133–1153.

Dabbagh M, Hamdaoui B and Rayes A (2020) Peak power shaving for reduced electricity costs in cloud data centers: opportunities and challenges. *IEEE Network* 34(3): 148–153.

Dao TT, Pham VT and Volos C (2021) Chapter 17-Implementation of organic RRAM with ink-jet printer: from design to using in RFID-based application. In: Volos C and Pham VT (eds) *Mem-elements for Neuromorphic Circuits with Artificial Intelligence Applications, Advances in Nonlinear Dynamics and Chaos (ANDC)*. Academic Press, pp. 347–360.

Das Sharma D, Pasdast G, Qian Z, et al. (2022) Universal Chiplet Interconnect Express (UCIe): an open industry standard for innovations with chiplets at package level. *IEEE Transactions on Components, Packaging, and Manufacturing Technology* 12(9): 1423–1431.

Dastidar J (2023) FPGAs and their evolving role in domain specific architectures: a case study of the AMD 400G adaptive SmartNIC/DPU SoC. In: *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '23*. Association for Computing Machinery, p. 151.

Davis B, Jacob B and Mudge T (2000) The new DRAM interfaces: SDRAM, RDRAM and variants. In: *High Performance Computing*. Springer Berlin Heidelberg, pp. 26–31.

Dawoud DS and Dawoud P (2020) *6 Serial Peripheral Interface (SPI), Chapter 1*. River Publishers, pp. 1–44.

de Paulis F, Rabinovich R, Mellitz R, et al. (2022) Feasibility analysis of chip-to-module channel operating at 212 Gbps. In: *2022 IEEE 26th Workshop on Signal and Power Integrity (SPI)*. IEEE, pp. 1–4.

Dee RH (2008) Magnetic tape for data storage: an enduring technology. *Proceedings of the IEEE* 96(11): 1775–1785.

Deirling K (2001) InfiniBridge architecture. Mellanox Technologies. Available at: https://web.archive.org/web/20040303145802/http://www.mellanox.com/technology/shared/EPFMellanox.pdf

Deneroff MM, Shaw DE, Dror RO, et al. (2008) Anton: a specialized ASIC for molecular dynamics. In: *2008 IEEE Hot Chips 20 Symposium (HCS)*. IEEE, pp. 1–34.

Dennard RH (2018) How we made DRAM. *Nature Electronics* 1(6): 372.

Devaux F (2019) The true processing in memory accelerator. In: *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE, pp. 1–24.

Dharamjeet CYS, Chen TY, Kuan YH, et al. (2022) LLSM: A Lifetime-Aware Wear-Leveling for LSM-Tree on NAND flash

memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41(11): 3946–3956.

Ding L, Kang P, Yin W, et al. (2016) Hardware TCP offload engine based on 10-Gbps ethernet for low-latency network communication. In: *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, pp. 269–272.

Diodato P (2000) Embedded DRAM: more than just a memory. *IEEE Communications Magazine* 38(7): 118–126.

Dongarra J and Keyes D (2024) The co-evolution of computational physics and high-performance computing. *Nature Reviews Physics* 6: 621–627.

Dorren HJS, Wittebol EHM, de Kluijver R, et al. (2015) Challenges for optically enabled high-radix switches for data center networks. *Journal of Lightwave Technology* 33(5): 1117–1125.

Doweck J, Kao WF, Lu AK, et al. (2017) Inside 6th-generation Intel core: new microarchitecture code-named skylake. *IEEE Micro* 37(2): 52–62.

Drenski T and Rasmussen JC (2018) ADC & DAC-Technology trends and steps to overcome current limitations. In: *Optical Fiber Communication Conference*. Optica Publishing Group, p. M2C.1.

Dror RO, Grossman J, Mackenzie KM, et al. (2010) Exploiting 162-Nanosecond End-to-End communication latency on anton. In: *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–12.

Dror RO, Young C and Shaw DE (2011) *Anton, A Special-Purpose Molecular Simulation Machine, Chapter A*. Springer US, pp. 60–71.

Dunning D, Mooney R, Stolt P, et al. (2009) TERA-SCALE memory challenges and solutions. *Intel Technology Journal* 13(4): 80–101.

Eckert C, Wang X, Wang J, et al. (2018) Neural cache: bit-serial in-cache acceleration of deep neural networks. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 383–396.

Eddington C (2002) InfiniBridge: an InfiniBand channel adapter with integrated switch. *IEEE Micro* 22(2): 48–56.

Ehrler S (2002) Properties of new printed circuit board base materials. *Circuit World* 28(4): 38–45.

El Sabbagh MA (2018) Critical assessment of linear redriver for PCIe server channel successful implementation. In: *2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*. IEEE, pp. 253–255.

Elliott J and Jung E (2013) Ushering in the 3D memory era with V-NAND. *Proceeding Flash Memory Summit*. The Future of Memory and Storage (FMS), 13–15.

Evans J (2022) Nvidia grace. In: *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, pp. 1–20.

Farmahini-Farahani A, Gurumurthi S, Loh G, et al. (2018) Challenges of high-capacity DRAM stacks and potential directions. In: *Proceedings of the Workshop on Memory Centric High Performance Computing, MCHPC'18*. Association for Computing Machinery, pp. 4–13.

Feinberg B, Vengalam UKR, Whitehair N, et al. (2018) Enabling scientific computing on memristive accelerators. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 367–382.

Feng R, Grana D, Mukerji T, et al. (2022) Application of Bayesian generative adversarial networks to geological facies modeling. *Mathematical Geosciences* 54(5): 831–855.

Firoozshahian A, Coburn J, Levenstein R, et al. (2023) MTIA: first generation silicon targeting meta's recommendation systems. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA '23*. Association for Computing Machinery, pp. 1–13.

Fontana JRE, Decad GM and Hetzler SR (2015) Volumetric density trends (TB/in.3) for storage components: TAPE, hard disk drives, NAND, and Blu-Ray. *Journal of Applied Physics* 117(17): 17E301.

Foss R (1997) Taking DRAM from 4 MBytes/s to 4 GBytes/s. In: *Proceedings of the 23rd European Solid-State Circuits Conference*. IEEE, p. 2.

Fowers J, Ovtcharov K, Papamichael M, et al. (2018) A configurable Cloud-Scale DNN processor for Real-Time AI. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 1–14.

Frachtenberg E (2012) Holistic datacenter design in the open compute project. *Computer* 45(07): 83–85.

Frazelle J (2020) Opening up the baseboard management controller. *Communications of the ACM* 63(2): 38–40.

Friederich C (2010) *Program and Erase of NAND Memory Arrays, Chapter 3*. Springer Netherlands, pp. 55–88.

Frolova LA, Rezvanova AA, Lukyanov BS, et al. (2015a) Design of rewritable and read-only non-volatile optical memory elements using photochromic spiropyran-based salts as light-sensitive materials. *Journal of Materials Chemistry* 3: 11675–11680.

Frolova LA, Troshin PA, Susarova DK, et al. (2015b) Photoswitchable organic field-effect transistors and memory elements comprising an interfacial photochromic layer. *Chemical Communications* 51: 6130–6132.

Fujiki D, Wang X, Subramaniyan A, et al. (2021a) *Computing With DRAMs*. Springer International Publishing, 13–28.

Fujiki D, Wang X, Subramaniyan A, et al. (2021b) *Computing With Non-Volatile Memories*. Springer International Publishing, pp. 45–60.

Fujiki D, Wang X, Subramaniyan A, et al. (2021c) *Computing With SRAMs*. Springer International Publishing, pp. 29–44.

Fujisawa H, Kubouchi S, Kuroki K, et al. (2007) An 8.1-ns column-access 1.6-Gb/s/pin DDR3 SDRAM with an 8: 4 multiplexed data-transfer scheme. *IEEE Journal of Solid-State Circuits* 42(1): 201–209.

Gabriel E, Fagg GE, Bosilca G, et al. (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. IEEE, pp. 97–104.

Gao F, Tziantzioulis G and Wentzlaff D (2019) ComputeDRAM: in-memory compute using Off-the-Shelf DRAMs. In:

Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52. Association for Computing Machinery, pp. 100–113.

Gao C, Xin X, Lu Y, et al. (2021) ParaBit: processing parallel bitwise operations in NAND flash memory based SSDs. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21*. Association for Computing Machinery, pp. 59–70.

Gao Z, Yan L and Zhou T (2024) Adaptive operator learning for infinite-dimensional Bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification* 12(4): 1389–1423.

Garber L (2012) Converged Infrastructure: Addressing the Efficiency Challenge. *Computer* 45(8): 17–20.

Garcıa V, Gomez-Luna J, Grass T, et al. (2016) Evaluating the effect of last-level cache sharing on integrated GPU-CPU systems with heterogeneous applications. In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, pp. 1–10.

Garg S, Gupta H and Chakraborty S (2022) Assessment of DeepONet for time dependent reliability analysis of dynamical systems subjected to stochastic loading. *Engineering Structures* 270: 114811.

Gebhart M, Keckler SW, Khailany B, et al. (2012) Unifying primary cache, scratch, and register file memories in a throughput processor. In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, pp. 96–106.

Genc G, Sarikas A, Kesen U, et al. (2020) Luffa/Epoxy composites: electrical properties for PCB application. *IEEE Transactions on Components, Packaging, and Manufacturing Technology* 10(6): 933–940.

Geneva N and Zabaras N (2022) Transformers for modeling physical systems. *Neural Networks: The Official Journal of the International Neural Network Society* 146: 272–289.

Georgakoudis G, Jain N, Ono T, et al. (2019) Evaluating the impact of energy efficient networks on HPC workloads. In: *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, pp. 301–310.

Ghandeharizadeh S, Irani S and Lam J (2015) *Cache Replacement With Memory Allocation*. Society for Industrial and Applied Mathematics, pp. 1–9.

Ghattas O, Willcox K and Heimbach P (2021) The imperative of physics-based modeling and inverse theory in computational science. *Nature Computational Science* 1(3): 166–174.

Ghose S, Hsieh K, Boroumand A, et al. (2019) *The Processing-In-Memory Paradigm: Mechanisms to Enable Adoption*. Springer International Publishing, pp. 133–194.

Giri D, Mantovani P and Carloni LP (2018) Accelerators and coherence: an SoC perspective. *IEEE Micro* 38(6): 36–45.

Goddeke D, Strzodka R and Turek S (2007) Performance and accuracy of hardware-oriented native-emulated-and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems* 22(4): 221–256.

Gomony MD, Weis C, Akesson B, et al. (2012) DRAM selection and configuration for real-time mobile systems. In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 51–56.

Gong YH and Chung SW (2016) Exploiting refresh effect of DRAM read operations: a practical approach to low-power refresh. *IEEE Transactions on Computers* 65(5): 1507–1517.

Gong J, Wang T, Chen J, et al. (2014) An efficient and flexible host-FPGA PCIe communication library. In: *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, pp. 1–6.

Goodfellow I, Pouget-Abadie J, Mirza M, et al. (2014) Generative adversarial nets. *Advances in Neural Information Processing Systems* 1: 2672–2680.

Goto K and Van de Geijn RA (2008) Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software* 34(3): 1–25.

Gourounas D, Hanindhito B, Fathi A, et al. (2023) FAWS: FPGA acceleration of large-scale wave simulations. In: *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, pp. 76–84.

Gourounas D, James A, Hanindhito B, et al. (2025) HighWave: large-scale high-bandwidth wave simulations on FPGAs. In: *2025 33rd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, pp. 1–10.

Grady TJ, Khan R, Louboutin M, et al. (2023) Model-parallel fourier neural operators as learned surrogates for large-scale parametric PDEs. *Computers & Geosciences* 178: 105402.

Graham RL, Bureddy D, Lui P, et al. (2016) Scalable hierarchical aggregation protocol (SHArP): a hardware architecture for efficient data reduction. In: *Proceedings of the First Workshop on Optimization of Communication in HPC, COM-HPC '16*. IEEE Press, pp. 1–10.

Grant RE, Afsahi A and Balaji P (2009) Evaluation of ConnectX virtual protocol interconnect for data centers. In: *2009 15th International Conference on Parallel and Distributed Systems*. IEEE, pp. 57–64.

Gropp W, Lusk E, Doss N, et al. (1996) A high-performance, portable implementation of the MPI Message Passing Interface standard. *Parallel Computing* 22(6): 789–828.

Grossi A (2010) *Charge Trap NAND Technologies, Chapter 5*. Springer Netherlands, pp. 115–129.

Grossman JP, Young C, Bank JA, et al. (2008) Simulation and embedded software development for Anton, a parallel machine with heterogeneous multicore ASICs. In: *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '08*. Association for Computing Machinery, pp. 125–130.

Grossman J, Towles B, Greskamp B, et al. (2015) Filtering, reductions and synchronization in the Anton 2 network. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, pp. 860–870.

Gu X, Christopher I, Bai T, et al. (2009) A component model of spatial locality. In: *Proceedings of the 2009 International Symposium on Memory Management, ISMM '09*. Association for Computing Machinery, pp. 99–108.

Gummaraju J, Morichetti L, Houston M, et al. (2010) Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors. In: *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT '10*. Association for Computing Machinery, pp. 205–216.

Günther F, Mehl M, Pögl M, et al. (2006) A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves. *SIAM Journal on Scientific Computing* 28(5): 1634–1650.

Guo Z, Balasubramanian S, Zlatanovici R, et al. (2005) FinFET-Based SRAM design. In: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, ISLPED '05*. Association for Computing Machinery, pp. 2–7.

Guo J, Lai M, Pang Z, et al. (2008) Hierarchical memory system design for a heterogeneous multi-core processor. In: *Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08*. Association for Computing Machinery, pp. 1504–1508.

Guo C, Wu H, Deng Z, et al. (2016) RDMA over commodity ethernet at scale. In: *Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16*. Association for Computing Machinery, pp. 202–215.

Haag M (2018) *Hyper-Converged Infrastructure For Dummies®, VMware 2nd Special Edition*. John Wiley & Sons, Inc.

Haghighat E, Raissi M, Moure A, et al. (2021) A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering* 379: 113741.

Haidar A, Wu P, Tomov S, et al. (2017) Investigating half precision arithmetic to accelerate dense linear system solvers. In: *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '17*. Association for Computing Machinery, pp. 1–8.

Hajkazemi MH, Tavana MK and Homayoun H (2015) Wide I/O or LPDDR? Exploration and analysis of performance, power and temperature trade-offs of emerging DRAM technologies in embedded MPSoCs. In: *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, pp. 62–69.

Hamano T, Endo T and Matsuoka S (2009) Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, pp. 1–8.

Hammarlund P, Martinez AJ, Bajwa AA, et al. (2014) Haswell: the fourth-generation Intel core processor. *IEEE Micro* 34(2): 6–20.

Hamzaoglu F, Arslan U, Bisnik N, et al. (2014) 13.1 A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology. In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, pp. 230–231.

Hamzaoglu F, Arslan U, Bisnik N, et al. (2015) A 1 Gb 2 GHz 128 GB/s bandwidth embedded DRAM in 22 nm Tri-Gate CMOS technology. *IEEE Journal of Solid-State Circuits* 50(1): 150–157.

Hanindhito B, Li R, Gourounas D, et al. (2021) Wave-PIM: accelerating wave simulation using processing-in-memory. In: *50th International Conference on Parallel Processing, ICPP 2021*. Association for Computing Machinery, pp. 1–13.

Hanindhito B, Gourounas D, Fathi A, et al. (2022) GAPS: GPU-Acceleration of PDE solvers for wave simulation. In: *Proceedings of the 36th ACM International Conference on Supercomputing, ICS '22*. Association for Computing Machinery, pp. 1–13.

Hanindhito B, Fathi A, Gourounas D, et al. (2026) Technology trends in computing hardware and their impacts on high-performance scientific computing Part I: general-purpose processors and hardware accelerators. *The International Journal of High Performance Computing Applications* 0(0): 10943420251348799.

Harkness J (2023) Tracking the transition to renewable energy across data centers. Cedara Inc. Available at: https://www.cedara.io/post/tracking-the-transition-to-renewable-energy-in-the-data-center-industry#:~:text=Google_Cloud-C_Microsoft_Azure-C_and,energy_between_2025_and_2030.

Hassan M (2018) On the Off-Chip memory latency of Real-Time systems: Is DDR DRAM really the best option? In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, pp. 495–505.

Hayot-Sasson V, Brown ST and Glatard T (2020) Performance benefits of Intel® Optane™ DC persistent memory for the parallel processing of large neuroimaging data. In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, pp. 509–518.

He C (2010) *Numerical Solutions of Differential Equations on FPGA-Enhanced Computers*. (PhD Thesis). Texas A & M University.

Hemani A and Klapproth P (2006) *Trends in SoC Architecture, Chapter 2*. Springer Netherlands, pp. 59–81.

Higham NJ and Mary T (2022) Mixed precision algorithms in numerical linear algebra. *Acta Numerica* 31: 347–414.

Higuchi T, Kodama T, Kato K, et al. (2021) 30.4 A 1Tb 3b/Cell 3D-Flash memory in a 170+ word-line-layer technology. *2021 IEEE International Solid- State Circuits Conference (ISSCC)* 64: 428–430.

Hijaz F, Shi Q and Khan O (2013) A private level-1 cache architecture to exploit the latency and capacity tradeoffs in multicores operating at Near-Threshold voltages. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, pp. 85–92.

Hildebrand M, Angeles JT, Lowe-Power J, et al. (2021) A case against hardware managed DRAM caches for NVRAM based systems. In: *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, pp. 194–204.

Ho J, Jain A and Abbeel P (2020) Denoising diffusion probabilistic models. In: *Proceedings of the 34th International Conference*

on Neural Information Processing Systems, NIPS '20. Curran Associates Inc, pp. 1–12.

Hoefler T, Copik M, Beckman P, et al. (2024) XaaS: Acceleration as a service to enable productive High-Performance cloud computing. *Computing in Science & Engineering* 26(03): 40–51.

Hoffer B, Wainstein N, Neumann CM, et al. (2022) Stateful logic using phase change memory. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 8(2): 77–83.

Hollis TM, Stave E, Ovard D, et al. (2019) Recent evolution in the DRAM interface: mile-markers along memory lane. *IEEE Solid-State Circuits Magazine* 11(2): 14–30.

Hollis TM, Schneider R, Brox M, et al. (2022) An 8-Gb GDDR6X DRAM achieving 22 Gb/s/pin with single-ended PAM-4 signaling. *IEEE Journal of Solid-State Circuits* 57(1): 224–235.

Holzinger P, Reiser D, Hahn T, et al. (2021) Fast HBM access with FPGAs: analysis, architectures, and applications. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, pp. 152–159.

Horowitz M, Yang CKK and Sidiropoulos S (1998) High-speed electrical signaling: overview and limitations. *IEEE Micro* 18(1): 12–24.

Hsieh MC (2016) Advanced flip chip package on package technology for mobile applications. In: *2016 17th International Conference on Electronic Packaging Technology (ICEPT)*. IEEE, pp. 486–491.

Hu X, Stow D and Xie Y (2018) Die stacking is happening. *IEEE Micro* 38(1): 22–28.

Huang CC and Nagarajan V (2014) Increasing cache capacity via critical-words-only cache. In: *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, pp. 125–132.

Huang DS, Lee JH, Tsai YS, et al. (2018) Comprehensive device and product level reliability studies on advanced CMOS technologies featuring 7nm high-k metal gate FinFET transistors. In: *2018 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, pp. 6F.7–1–6F.7–5.

Huang X, Wang F and Alkhalifah T (2024) Diffusion-based subsurface multiphysics monitoring and forecasting. arXiv preprint arXiv:2407.18426. https://arxiv.org/html/2407.18426v2

Hur JY, Rhim SW, Lee BH, et al. (2019) Adaptive linear address map for bank interleaving in DRAMs. *IEEE Access* 7: 129604–129616.

Hylick A, Sohan R, Rice A, et al. (2008) An analysis of hard drive energy consumption. In: *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*. IEEE, pp. 1–10.

Imani M, Gupta S, Kim Y, et al. (2019a) FloatPIM: in-memory acceleration of deep neural network training with high precision. In: *Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19*. Association for Computing Machinery, pp. 802–815.

Imani M, Gupta S, Kim Y, et al. (2019b) FloatPIM: in-memory acceleration of deep neural network training with high precision. In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 802–815.

Inoue H and Taura K (2015) SIMD- and Cache-Friendly algorithm for sorting an array of structures. *Proceedings of the VLDB Endowment* 8(11): 1274–1285.

Ishimaru K (2019) Future of Non-Volatile memory from storage to computing. In: *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, pp. 1.3.1–1.3.6.

Islam MA, Arafath MY and Hasan MJ (2014) Design of DDR4 SDRAM controller. In: *8th International Conference on Electrical and Computer Engineering*. IEEE, pp. 148–151.

Itoh K (2008) The history of DRAM circuit designs – at the Forefront of DRAM development. *IEEE Solid-State Circuits Newsletter* 13(1): 27–31.

Itoh K (2011) Embedded memories: progress and a look into the future. *IEEE Design & Test of Computers* 28(1): 10–13.

Iwasaki S (1984) Perpendicular magnetic recording – Evolution and future. *IEEE Transactions on Magnetics* 20(5): 657–662.

Iyer S and Kalter H (1999) Embedded DRAM technology: opportunities and challenges. *IEEE Spectrum* 36(4): 56–64.

Jacob B (2003) A case for studying DRAM issues at the system level. *IEEE Micro* 23(4): 44–56.

Jacob B, Ng S and Wang D (2007) *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc.

Jaffer S, Mahdaviani K and Schroeder B (2022) Improving the endurance of next generation SSD's using WOM-v codes. *ACM Transactions on Storage* 18(4): 1–32.

Jagtap AD, Kharazmi E and Karniadakis GE (2020) Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering* 365: 113028.

Jahed M (1995) Interfacing synchronous DRAMs to Pentium processors. In: *Proceedings of WESCON'95*. IEEE, p. 25.

Jaleel A, Theobald KB, Steely SC, et al. (2010) High performance cache replacement using Re-Reference Interval Prediction (RRIP). In: *Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA '10*. Association for Computing Machinery, pp. 60–71.

James D (2009) Design-for-manufacturing features in nanometer processes - A reverse engineering perspective. In: *2009 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*. IEEE, pp. 56–61.

Jang J, Kim HS, Cho W, et al. (2009) Vertical cell array using TCAT(Terabit Cell Array Transistor) technology for ultra high density NAND flash memory *2009 Symposium on VLSI Technology*, 192–193.

Jeddeloh J and Keeth B (2012) Hybrid memory cube new DRAM architecture increases density and performance. In: *2012 Symposium on VLSI Technology (VLSIT)*. IEEE, pp. 87–88.

Jeong YJ, Yoo EJ, Kim LH, et al. (2016) Light-responsive spiropyran based polymer thin films for use in organic field-effect transistor memories. *Journal of Materials Chemistry C* 4: 5398–5406.

Ji SY, Loop B, James PD, et al. (2010) An empirical study of performance and power scaling of low voltage DDR3. In: *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*. IEEE, pp. 9–12.

Ji Y, Yang Y, Lee SK, et al. (2016) Flexible nanoporous WO3–x nonvolatile memory device. *ACS Nano* 10(8): 7598–7603.

Joshi AD and Ramasubramanian N (2015) Comparison of significant issues in multicore cache coherence. In: *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*. IEEE, pp. 108–112.

Jouppi NP, Young C, Patil N, et al. (2017) In-Datacenter performance analysis of a tensor processing unit. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17*. Association for Computing Machinery, pp. 1–12.

Jouppi NP, Yoon DH, Kurian G, et al. (2020) A domain-specific supercomputer for training deep neural networks. *Communications of the ACM* 63(7): 67–78.

Jouppi NP, Hyun Yoon D, Ashcraft M, et al. (2021) Ten lessons from three generations shaped Google's TPUv4i: industrial product. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 1–14.

Jouppi N, Kurian G, Li S, et al. (2023) TPU v4: an optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA '23*. Association for Computing Machinery, pp. 1–14.

Jun H, Cho J, Lee K, et al. (2017) HBM (High Bandwidth Memory) DRAM technology and architecture. In: *2017 IEEE International Memory Workshop (IMW)*. IEEE, pp. 1–4.

Jung M (2022) Hello bytes, bye blocks: PCIe storage meets Compute Express Link for memory expansion (CXL-SSD). In: *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems, HotStorage '22*. Association for Computing Machinery, pp. 45–51.

Kadeethum T, Ballarin F, Choi Y, et al. (2022) Non-intrusive reduced order modeling of natural convection in porous media using convolutional autoencoders: comparison with linear subspace techniques. *Advances in Water Resources* 160: 104098.

Kalavade P (2020) 4 bits/cell 96 layer floating gate 3D NAND with CMOS under array technology and SSDs. In: *2020 IEEE International Memory Workshop (IMW)*. IEEE, pp. 1–4.

Kandemir M, Ramanujam J and Choudhary A (1999) Improving cache locality by a combination of loop and data transformations. *IEEE Transactions on Computers* 48(2): 159–167.

Kang U, Yu H, Park C, et al. (2014) Co-Architecting controllers and DRAM to enhance DRAM process scaling. *The Memory Forum* 1: 1–12.

Karniadakis GE, Kevrekidis IG, Lu L, et al. (2021) Physics-informed machine learning. *Nature Reviews Physics* 3(6): 422–440.

Karp M, Podobas A, Jansson N, et al. (2021) High-Performance spectral element methods on Field-Programmable gate arrays: implementation, evaluation, and future projection. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 1077–1086.

Kasperek D, Antonowicz P, Baranowski M, et al. (2023) Comparison of the usability of Apple M2 and M1 processors for various machine learning tasks. *Sensors* 23(12): 5424.

Katsumata R, Kito M, Fukuzumi Y, et al. (2009) Pipe-shaped BiCS flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices. In: *2009 Symposium on VLSI Technology*. IEEE, pp. 136–137.

Kavanagh R, Armstrong D and Djemame K (2016) Accuracy of energy model calibration with IPMI. In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, pp. 648–655.

Ke L, Zhang X, So J, et al. (2022) Near-Memory processing in action: accelerating personalized recommendation with AxDIMM. *IEEE Micro* 42(1): 116–127.

Keckler SW, Dally WJ, Khailany B, et al. (2011) GPUs and the future of parallel computing. *IEEE Micro* 31(5): 7–17.

Keitel-Schulz D and Wehn N (2001) Embedded DRAM development: Technology, physical design, and application issues. *IEEE Design & Test of Computers* 18(3): 7–15.

Kenter T, Mahale G, Alhaddad S, et al. (2018) OpenCL-based FPGA design to accelerate the nodal discontinuous Galerkin method for unstructured meshes. In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, pp. 189–196.

Kenter T, Shambhu A, Faghih-Naini S, et al. (2021) Algorithm-hardware co-design of a discontinuous Galerkin shallow-water model for a dataflow architecture on FPGA. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. IEEE, pp. 1–11.

Kenyon C and Capano C (2022) Apple silicon performance in scientific computing. In: *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, pp. 1–10.

Kestor G, Gioiosa R, Kerbyson DJ, et al. (2013) Quantifying the energy cost of data movement in scientific applications. In: *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, pp. 56–65.

Keyes D (2023) Efficient computation through tuned approximation. Online power point presentation: https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-2b-Dinnerspeaker-Keyes-Efficient-Computation.pdf

Keyes DE, Ltaief H and Turkiyyah G (2020) Hierarchical algorithms on hierarchical architectures. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences* 378: 20190055.

Khan S, Lee D, Kim Y, et al. (2014) The efficacy of error mitigation techniques for DRAM retention failures: a comparative experimental study. *ACM Sigmetrics Performance Evaluation Review* 42(1): 519–532.

Kharazmi E, Zhang Z and Karniadakis GE (2021) hp-VPINNs: variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering* 374: 113547.

Kho R, Boursin D, Brox M, et al. (2010) A 75 nm 7 Gb/s/pin 1 Gb GDDR5 graphics memory device with bandwidth improvement techniques. *IEEE Journal of Solid-State Circuits* 45(1): 120–133.

Khokhar A, Prasanna V, Shaaban M, et al. (1993) Heterogeneous computing: challenges and opportunities. *Computer* 26(6): 18–27.

Khoram S, Zha Y, Zhang J, et al. (2017) Challenges and opportunities: from near-memory computing to in-memory computing. In: *Proceedings of the 2017 ACM on International Symposium on Physical Design, ISPD '17*. Association for Computing Machinery, pp. 43–46.

Kim J (2015) Active Si interposer for 3D IC integrations. In: *2015 International 3D Systems Integration Conference*. IEEE, pp. TS11.1.1–TS11.1.3.

Kim K (2021) The smallest engine transforming humanity: the past, present, and future. In: *2021 IEEE International Electron Devices Meeting (IEDM)*. IEEE, pp. 1.1.1–1.1.8.

Kim NS and Mudge T (2003) The microarchitecture of a low power register file. In: *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, ISLPED '03*. Association for Computing Machinery, pp. 384–389.

Kim J, Hong AJ, Kim SM, et al. (2009a) Novel Vertical-Stacked-Array-Transistor (VSAT) for ultra-high-density and cost-effective NAND flash memory devices and SSD (Solid State Drive). In: *2009 Symposium on VLSI Technology*. IEEE, pp. 186–187.

Kim W, Choi S, Sung J, et al. (2009b) Multi-layered Vertical Gate NAND flash overcoming stacking limit for terabit density storage. In: *2009 Symposium on VLSI Technology*. IEEE, pp. 188–189.

Kim C, Song J and Lee HW (2014) *An I/O Line Configuration and Organization of DRAM, Chapter 2*. Springer International Publishing, pp. 13–24.

Kim C, Lee HW and Song J (2016) Memory interfaces: past, present, and future. *IEEE Solid-State Circuits Magazine* 8(2): 23–34.

Kim NS, Chen D, Xiong J, et al. (2017) Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era. *IEEE Micro* 37(4): 10–18.

Kim D, Park M, Jang S, et al. (2019) 23.2 A 1.1V 1ynm 6.4Gb/s/pin 16Gb DDR5 SDRAM with a phase-rotator-based DLL, high-speed SerDes and RX/TX equalization scheme. In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. IEEE, pp. 380–382.

Kim D, Park M, Jang S, et al. (2020) A 1.1-V 10-nm class 6.4-Gb/s/Pin 16-Gb DDR5 SDRAM with a phase Rotator-ILO DLL, High-Speed SerDes, and DFE/FFE equalization scheme for Rx/Tx. *IEEE Journal of Solid-State Circuits* 55(1): 167–177.

Kim J, Jang I, Reda W, et al. (2021a) LineFS: efficient SmartNIC offload of a distributed file system with pipeline parallelism. In: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21*. Association for Computing Machinery, pp. 756–771.

Kim JH, Kang S, Lee S, et al. (2021b) Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond. In: *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, pp. pp. 1–26.

Kim K, Chae JH, Yang J, et al. (2021c) 25.1 A 24Gb/s/pin 8Gb GDDR6 with a Half-Rate Daisy-Chain-Based clocking architecture and IO Circuitry for Low-Noise Operation. *2021 IEEE International Solid- State Circuits Conference (ISSCC)* 64: 344–346.

Kim S, Park S and Park CS (2021d) System-Level communication performance estimation for DMA-controlled accelerators. *IEEE Access* 9: 141389–141402.

Kim S, Shin T, Park H, et al. (2021e) Signal integrity design and analysis of a spiral Through-Silicon Via (TSV) array channel for High Bandwidth Memory (HBM). In: *2021 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*. IEEE, pp. 1–3.

Kim JH, Kang SH, Lee S, et al. (2022a) Aquabolt-XL HBM2-PIM, LPDDR5-PIM with In-Memory processing, and AXDIMM with acceleration buffer. *IEEE Micro* 42(3): 20–30.

Kim SS, Yong SK, Kim W, et al. (2022b) Review of semiconductor flash memory devices for material and process issues. *Advanced Materials* 35(n/a): 2200659.

Kim Y, Choi Y, Widemann D, et al. (2022c) A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. *Journal of Computational Physics* 451: 110841.

Kim T, Lee J, Kim Y, et al. (2023) Thermal improvement of HBM with joint thermal resistance reduction for scaling 12 stacks and beyond. In: *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, pp. 767–771.

King YC, King TJ and Hu C (2001) Charge-trap memory device fabricated by oxidation of Si/sub 1-x/Ge/sub x. *IEEE Transactions on Electron Devices* 48(4): 696–700.

Klein D (2016) The history of semiconductor memory: from magnetic tape to NAND flash memory. *IEEE Solid-State Circuits Magazine* 8(2): 16–22.

Kobota T (2004) HD DVD - overview of next generation optical disc format. In: *The IEEE 2-Day Seminar on IT to HD: Visions of Broadcasting in the 21st Century, (Ref. No. 2004/10760)*. IEEE, pp. 213–224.

Kobyzev I, Prince SJD and Brubaker MA (2020) Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42: 1–16.

Kogge P and Shalf J (2013) Exascale computing trends: Adjusting to the new normal for computer architecture. *Computing in Science & Engineering* 15(6): 16–26.

Koh Y (2009) NAND flash scaling beyond 20nm. In: *2009 IEEE International Memory Workshop*. IEEE, pp. 1–3.

Köhler J, Klein L and Noé F (2020) Equivariant flows: exact likelihood generative learning for symmetric densities. In: *Proceedings of the 37th International Conference on Machine Learning, ICML'20*. International Conference on Machine Learning (ICML), pp. 1–10.

Komatitsch D, Erlebacher G, Göddeke D, et al. (2010) High-Order Finite-Element seismic wave propagation modeling with MPI

on a large GPU cluster. *Journal of Computational Physics* 229(20): 7692–7714.

Kondo M and Nakamura H (2005) A small, fast and low-power register file by bit-partitioning. In: *11th International Symposium on High-Performance Computer Architecture*. IEEE, pp. 40–49.

Kong X, Lou J, Bai W, et al. (2023) Towards a manageable intra-host network. In: *Proceedings of the 19th Workshop on Hot Topics in Operating Systems, HOTOS '23*. Association for Computing Machinery, pp. 206–213.

Koo K, Ok S, Kang Y, et al. (2012) A 1.2V 38nm 2.4Gb/s/pin 2Gb DDR4 SDRAM with bank group and × 4 half-page architecture. In: *2012 IEEE International Solid-State Circuits Conference*. IEEE, pp. 40–41.

Koot M and Wijnhoven F (2021) Usage impact on data center electricity needs: A system dynamic forecasting model. *Applied Energy* 291: 116798.

Koronen C, Åhman M and Nilsson LJ (2020) Data centres in future European energy systems—energy efficiency, integration and policy. *Energy Efficiency* 13(1): 129–144.

Kouchi T, Kumazaki N, Yamaoka M, et al. (2020) 13.5 A 128Gb 1b/Cell 96-Word-Line-Layer 3D flash memory to improve random read latency with tPROG=75µs and tR=4µs. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. IEEE, pp. 226–228.

Kouchi T, Kakoi M, Kumazaki N, et al. (2021) A 128Gb 1-bit/Cell 96-Word-Line-Layer 3D flash memory to improve the random read latency with tProg = 75 µs and tR = 4 µs. *IEEE Journal of Solid-State Circuits* 56(1): 225–234.

Kovachki N, Li Z, Liu B, et al. (2023) Neural operator: learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research* 24(89): 1–97.

Krause PJ, Shastri A and Kahn JM (2018) Data center links beyond 100Gbit/s per wavelength. *Optical Fiber Technology* 44: 69–85.

Krishnapriyan AS, Gholami A, Zhe S, et al. (2021) Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf

Krueger J, Donofrio D, Shalf J, et al. (2011) Hardware/software co-design for energy-efficient seismic modeling. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. IEEE, pp. 1–12.

Kryder MH, Gage EC, McDaniel TW, et al. (2008) Heat assisted magnetic recording. *Proceedings of the IEEE* 96(11): 1810–1835.

Kumar S, Shriraman A and Vedula N (2015) Fusion: design tradeoffs in coherent cache hierarchies for accelerators. In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA '15*. Association for Computing Machinery, pp. 733–745.

Kumar S, Vasa M, Bandi S, et al. (2023) Demystifying the influence of channel components on Signal Integrity (SI) margin. In: *2023 Joint Asia-Pacific International Symposium on Electromagnetic Compatibility and International Conference on ElectroMagnetic Interference & Compatibility (APEMC/INCEMIC)*. IEEE, pp. 1–4.

Kumari K and Donzis DA (2020) Direct numerical simulations of turbulent flows using high-order asynchrony-tolerant schemes: Accuracy and performance. *Journal of Computational Physics* 419: 109626.

Kundu S (2014) *Network-on-Chip: The Next Generation of System-on-Chip Integration*. Taylor & Francis.

Kuskin JS, Young C, Grossman J, et al. (2008) Incorporating flexibility in Anton, a specialized machine for molecular dynamics simulation. In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, pp. 343–354.

Kyung KH, Kim CW, Lee JY, et al. (2005) A 800Mb/s/pin 2GB DDR2 SDRAM using an 80nm triple metal technology. In: *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005*. IEEE, pp. 468–610.

Lage C, Hayden J and Subramanian C (1996) Advanced SRAM technology-the race between 4T and 6T cells. In: *International Electron Devices Meeting*. IEEE, pp. 271–274.

Larson A (2023) NuScale gets a win with SMRs for data centers in Ohio and Pennsylvania. https://www.powermag.com/nuscale-gets-a-win-with-smrs-for-data-centers-in-ohio-and-pennsylvania/

Larson RH, Salmon JK, Dror RO, et al. (2008) High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation. In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, pp. 331–342.

Lauterbach G (2021) The path to successful wafer-scale integration: the cerebras story. *IEEE Micro* 41(6): 52–57.

Lee JH, Lee JS and Kim SD (2000) A new cache architecture based on temporal and spatial locality. *Journal of Systems Architecture* 46(15): 1451–1467.

Lee DU, Lee HW, Kwean KC, et al. (2006) A 2.5Gb/s/pin 256Mb GDDR3 SDRAM with series pipelined CAS latency control and dual-loop digital DLL. In: *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*. IEEE, pp. 547–556.

Lee CJ, Mutlu O, Narasiman V, et al. (2008) Prefetch-Aware DRAM controllers. In: *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, pp. 200–209.

Lee J, Lee S and Nam S (2010a) Multi-Slot main memory system for Post DDR3. *IEEE Transactions on Circuits and Systems II: Express Briefs* 57(5): 334–338.

Lee T, Kim SU, Min J, et al. (2010b) Multilevel biomemory device consisting of recombinant Azurin/Cytochrome c. *Advanced Materials* 22(4): 510–514.

Lee DU, Kim KW, Kim KW, et al. (2015a) A 1.2 V 8 Gb 8-Channel 128 GB/s High-Bandwidth Memory (HBM) stacked DRAM with effective I/O test circuits. *IEEE Journal of Solid-State Circuits* 50(1): 191–203.

Lee H, Cho K, Kim H, et al. (2015b) Electrical performance of high bandwidth memory (HBM) interposer channel in terabyte/s

bandwidth graphics module. In: *2015 International 3D Systems Integration Conference*. IEEE, pp. TS2.2.1–TS2.2.4.

Lee D, Hwang E, Lee Y, et al. (2016a) Multibit MoS2 photoelectronic memory with ultrahigh sensitivity. *Advanced Materials* 28(41): 9196–9202.

Lee S, Lee J, Park I, et al. (2016b) 7.5 A 128Gb 2b/cell NAND flash memory in 14nm technology with tPROG=640µs and 800MB/s I/O rate. In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, pp. 138–139.

Lee J, Pak S, Lee YW, et al. (2017) Monolayer optical memory cells based on artificial trap-mediated charge storage and release. *Nature Communications* 8(1): 14734.

Lee C, Shin W, Kim DJ, et al. (2020a) NVDIMM-C: a byte-addressable non-volatile memory module for compatibility with standard ddr memory interfaces. In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 502–514.

Lee DU, Cho HS, Kim J, et al. (2020b) 22.3 A 128Gb 8-High 512GB/s HBM2E DRAM with a pseudo quarter bank structure, power dispersion and an instruction-based at-speed PMBIST. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. IEEE, pp. 334–336.

Lee JY, Cho S and Hwang HJ (2023a) HyperDeepONet: learning operator with complex target function space using the limited resources via hypernetwork. In: *The Eleventh International Conference on Learning Representations*. IEEE, pp. 1–26.

Lee S, Park J, Moon JK, et al. (2023b) A study on the advanced chip to wafer stack for better thermal dissipation of high bandwidth memory. In: *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, pp. 878–882.

Lehmann F, Gatti F, Bertin M, et al. (2024) 3D elastic wave propagation with a factorized Fourier Neural Operator (F-FNO). *Computer Methods in Applied Mechanics and Engineering* 420: 116718.

Leinonen J, Bonev B, Kurth T, et al. (2024) Modulated adaptive Fourier neural operators for temporal interpolation of weather forecasts. arXiv preprint arXiv:2410.18904. https://arxiv.org/abs/2410.18904

Lencer D, Salinga M, Grabowski B, et al. (2008) A map for Phase-Change materials. *Nature Materials* 7(12): 972–977.

Li Y (2020) 3D NAND memory and its application in Solid-State drives: architecture, reliability, flash management techniques, and current trends. *IEEE Solid-State Circuits Magazine* 12(4): 56–65.

Li H, Zhou Y, Tu W, et al. (2015a) State-of-the-Art progress in diverse heterostructured photocatalysts toward promoting photocatalytic performance. *Advanced Functional Materials* 25(7): 998–1013.

Li X, Lung CH and Majumdar S (2015b) Energy aware green spine switch management for Spine-Leaf datacenter networks. In: *2015 IEEE International Conference on Communications (ICC)*. IEEE, pp. 116–121.

Li A, Song SL, Kumar A, et al. (2016) Critical points based register-concurrency autotuning for GPUs. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1273–1278.

Li S, Reddy D and Jacob B (2018) A performance & power comparison of modern High-Speed DRAM architectures. In: *Proceedings of the International Symposium on Memory Systems, MEMSYS '18*. Association for Computing Machinery, pp. 341–353.

Li H, Bhargava M, Whatmough PN, et al. (2019a) On-Chip memory technology design space explorations for mobile deep neural network accelerators. In: *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*. Association for Computing Machinery, pp. 1–6.

Li Y, Kashyap A, Guo Y, et al. (2023a) Compression analysis for BlueField-2/3 DPUs: lossy and lossless perspectives. *IEEE Micro* 1: 1–9.

Li Q, Li H and Zhang K (2019b) A survey of SSD lifecycle prediction. In: *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, pp. 195–198.

Li A, Song SL, Chen J, et al. (2020) Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems* 31(1): 94–110.

Li Z, Meidani K and Farimani AB (2023c) Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research* 1: 1–19.

Li Y, Qi H, Lu G, et al. (2022) Understanding hot interconnects with an extensive benchmark survey. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations* 2(3): 100074.

Li Z, Huang DZ, Liu B, et al. (2023b) Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research* 24(388): 1–26.

Li Z, Zheng H, Kovachki N, et al. (2024) Physics-informed neural operator for learning partial differential equations. *ACM/IMS Journal of Data Science* 1(3): 1–27.

Liang Y and Wang S (2016) Performance-centric optimization for racetrack memory based register file on GPUs. *Journal of Computer Science and Technology* 31(1): 36–49.

Liao J, Zhang F, Li L, et al. (2015) Adaptive wear-leveling in flash-based memory. *IEEE Computer Architecture Letters* 14(1): 1–4.

Lie S (2022) Cerebras architecture deep dive: first look inside the HW/SW co-design for deep learning: cerebras systems. In: *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, pp. 1–34.

Ligorio G, Nardi MV and Koch N (2017) Lithography-free miniaturization of resistive nonvolatile memory devices to the 100 nm scale by glancing angle deposition. *Nano Letters* 17(2): 1149–1153.

Lim J and Kim H (2012) Design space exploration of memory model for heterogeneous computing. In: *Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, MSPC '12*. Association for Computing Machinery, pp. 74–75.

Lin S, Liao M, Lan A, et al. (2014) Alternative package-on-package with organic substrate interposer for stacking packaging solution. In: *2014 IEEE 16th Electronics Packaging Technology Conference (EPTC)*. IEEE, pp. 250–253.

Lin H, Boecker C, Hossain M, et al. (2021) ADC-DSP-based 10-to-112-Gb/s Multi-Standard receiver in 7-nm FinFET. *IEEE Journal of Solid-State Circuits* 56(4): 1265–1277.

Lindtjorn O, Clapp R, Pell O, et al. (2011) Beyond traditional microprocessors for geoscience high-performance computing applications. *IEEE Micro* 31(2): 41–49.

Lingambudi A, Vijay S, Becker WD, et al. (2016) Timing margin analysis and power measurement with DDR4 memory. In: *2016 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*. IEEE, pp. 51–53.

Litvinenko A, Sun Y, Genton MG, et al. (2019) Likelihood approximation with hierarchical matrices for large spatial datasets. *Computational Statistics & Data Analysis* 137: 115–132.

Liu L (1994) Issues in multi-level cache designs. In: *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, pp. 46–52.

Liu J, Jaiyen B, Kim Y, et al. (2013) An experimental study of data retention behavior in modern DRAM devices: implications for retention time profiling mechanisms. *ACM SIGARCH Computer Architecture News* 41(3): 60–71.

Liu D, Zhong K, Zhu X, et al. (2017) Non-volatile memory based page swapping for building high-performance mobile devices. *IEEE Transactions on Computers* 66(11): 1918–1931.

Liu H, Zheng L, Huang Y, et al. (2023) Accelerating personalized recommendation with cross-level near-memory processing. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA '23*. Association for Computing Machinery, pp. 1–13.

Loh GH, Jerger NE, Kannan A, et al. (2015) Interconnect-memory challenges for multi-chip, silicon interposer systems. In: *Proceedings of the 2015 International Symposium on Memory Systems, MEMSYS '15*. Association for Computing Machinery, pp. 3–10.

Lojek B (2007) *The MOS Transistor, Chapter The MOS Transistor*. Springer Berlin Heidelberg, pp. 317–374.

Lopez-Alvis J, Laloy E, Nguyen F, et al. (2021) Deep generative models in inversion: The impact of the generator's nonlinearity and development of a new approach based on a variational autoencoder. *Computers & Geosciences* 152: 104762.

Lu CY (2012) Future prospects of NAND flash memory technology - the evolution from floating gate to charge trapping to 3D stacking. *Journal of Nanoscience and Nanotechnology* 12(10): 7604–7618.

Lu Y and Xu W (2024) Generative downscaling of PDE solvers with physics-guided diffusion models. *Journal of Scientific Computing* 101(71): 71.

Lu L, Jin P, Pang G, et al. (2021a) Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence* 3(3): 218–229.

Lu L, Pestourie R, Yao W, et al. (2021b) Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing* 43(6): B1105–B1132.

Lu PJ, Lai MC and Chang JS (2022) A survey of high-performance interconnection networks in high-performance computer systems. *Electronics* 11(9): 1369.

Lutz C, Breß S, Zeuch S, et al. (2020) Pump up the volume: processing large data on GPUs with fast interconnects. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20*. Association for Computing Machinery, pp. 1633–1649.

Ma S, Huang L, Lei Y, et al. (2019) An efficient Direct Memory Access (DMA) controller for scientific computing accelerators. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1–5.

Maccabe A, Zhu W, Otto J, et al. (2002) Experience in offloading protocol processing to a programmable NIC. In: *Proceedings. IEEE International Conference on Cluster Computing*. IEEE, pp. 67–74.

Macri J (2015) AMD's next generation GPU and high bandwidth memory architecture: fury. In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE, pp. 1–26.

Maeda T and Ukita N (2023) Fast inference and update of probabilistic density estimation on trajectory prediction. *Proceedings of the IEEE/CVF International Conference on Computer Vision* (ICCV), 9795–9805. doi: 10.48550/arXiv.2308.08824.

Mahanta S, Prusti S and Patnaik S (2022) Design of ultra low power 6T SRAM cell using 180 nm CMOS technology for access enhancement. In: *2022 International Interdisciplinary Conference on Mathematics, Engineering and Science (MESIICON)*. IEEE, pp. 1–6.

Mahapatra NR and Venkatrao B (1999) The processor-memory bottleneck: problems and solutions. *XRDS: Crossroads, The ACM Magazine For Students* 5(3es): 2.

Maher SJ, Ralphs TK and Shinano Y (2021) Assessing the effectiveness of (Parallel) branch-and-bound algorithms. CoRR abs/2104.10025. https://arxiv.org/abs/2104.10025

Malas T, Hager G, Ltaief H, et al. (2015) Multicore-optimized wavefront diamond blocking for optimizing stencil updates. *SIAM Journal on Scientific Computing* 37(4): C439–C464.

Mandl L, Goswami S, Lambers L, et al. (2025) Separable physics-informed DeepONet: Breaking the curse of dimensionality in physics-informed machine learning. *Computer Methods in Applied Mechanics and Engineering* 434: 117586.

Maniotis P, Schares L, Lee BG, et al. (2020) Scaling HPC networks with co-packaged optics. In: *Optical Fiber Communication Conference (OFC) 2020*. Optica Publishing Group.

Maniotis P, Schares L, Kuchta DM, et al. (2022) Toward higher-radix switches with co-packaged optics for improved network locality in data center and HPC networks. *Journal of Optical Communications and Networking* 14(6): C1–C10.

Margala M (1999) Low-power SRAM circuit design. In: *Records of the 1999 IEEE International Workshop on Memory Technology, Design and Testing*. IEEE, pp. 115–122.

Marongiu A, Capotondi A, Tagliavini G, et al. (2015) Simplifying many-core-based heterogeneous SoC programming with offload directives. *IEEE Transactions on Industrial Informatics* 11(4): 957–967.

Martin MMK, Hill MD and Sorin DJ (2012) Why On-Chip cache coherence is here to stay. *Communications of the ACM* 55(7): 78–89.

Martwick A and Drew J (2015) Silicon interposer and TSV signaling. In: *2015 IEEE 65th Electronic Components and Technology Conference (ECTC)*. IEEE, pp. 266–275.

Masuoka F, Asano M, Iwahashi H, et al. (1984) A new flash E2PROM cell using triple polysilicon technology. In: *1984 International Electron Devices Meeting*. IEEE, pp. 464–467.

Matani D, Shah K and Mitra A (2021) An O(1) algorithm for implementing the LFU cache eviction scheme. https://arxiv.org/abs/2110.11602

Matsui C, Sun C and Takeuchi K (2017) Design of hybrid SSDs with storage class memory and NAND flash memory. *Proceedings of the IEEE* 105(9): 1812–1821.

Matsuoka S, Domke J, Wahib M, et al. (2023) Myths and legends in high-performance computing. *The International Journal of High Performance Computing Applications* 37(3-4): 245–259.

Maurice C, Le Scouarnec N, Neumann C, et al. (2015) Reverse engineering intel last-level cache complex addressing using performance counters. In: Bos H, Monrose F and Blanc G (eds) *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, pp. 48–65.

Mayhew D and Krishnan V (2003) PCI express and advanced switching: evolutionary path to building next generation interconnects. In: *11th Symposium on High Performance Interconnects, 2003. Proceedings*. IEEE, pp. 21–29.

Mazumdar C, Mitra P and Basu A (2021) Dead page and dead block predictors: cleaning TLBs and caches together. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 507–519.

McAuley A and Francis P (1993) Fast routing table lookup using CAMs. *IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings* 3: 1382–1391.

McFarling S (1989) Program optimization for instruction caches. *SIGARCH Comput. Archit. News* 17(2): 183–191.

Mekkat V, Holey A, Yew PC, et al. (2013) Managing shared last-level cache in a heterogeneous multicore processor. In: *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. IEEE, pp. 225–234.

Mellor C (2020) Why DRAM is stuck in a 10nm trap. https://blocksandfiles.com/2020/04/13/dram-is-stuck-in-a-10nm-process-trap/

Michelogiannakis G, Arafa Y, Cook B, et al. (2023) Efficient intra-rack resource disaggregation for HPC using co-packaged DWDM photonics. In: *2023 IEEE International Conference on Cluster Computing (Cluster)*. IEEE Computer Society, pp. 158–172.

Micheloni R and Crippa L (2010) *XLC storage, Chapter 16*. Springer Netherlands, pp. 455–482.

Micheloni R, Ravasio R, Marelli A, et al. (2006) A 4Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36MB/s system read throughput. In: *2006 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*. IEEE, pp. 497–506.

Miele R and Azevedo L (2024) Physics-informed W-net GAN for the direct stochastic inversion of fullstack seismic data into facies models. *Scientific Reports* 14(1): 5122.

Mirosanlou R, Guo D, Hassan M, et al. (2020) MCsim: an extensible DRAM memory controller simulator. *IEEE Computer Architecture Letters* 19(2): 105–109.

Mishra N, Mukherjee B, Xing G, et al. (2016) Cation exchange synthesis of uniform PbSe/PbS core/shell tetra-pods and their use as near-infrared photodetectors. *Nanoscale* 8: 14203–14212.

Mittal S (2017) A survey of techniques for architecting and managing GPU register file. *IEEE Transactions on Parallel and Distributed Systems* 28(1): 16–28.

Miyagawa N (2014) Overview of Blu-Ray disc™ recordable/rewritable media technology. *Frontiers of Optoelectronics* 7(4): 409–424.

Modave A, St-Cyr A, Mulder W, et al. (2015) A nodal discontinuous Galerkin method for reverse-time migration on GPU clusters. *Geophysical Journal International* 203(2): 1419–1435.

Mormann A (1996) Burst and latency requirements drive EDO and BEDO DRAM standards. In: *COMPCON '96. Technologies for the Information Superhighway Digest of Papers*. IEEE, pp. 356–359.

Munger B, Wilcox K, Sniderman J, et al. (2023) "Zen 4": The AMD 5nm 5.7GHz x86-64 microprocessor core. In: *2023 IEEE International Solid- State Circuits Conference (ISSCC)*. IEEE, pp. 38–39.

Muthukrishnan H, Nellans D, Lustig D, et al. (2021) Efficient Multi-GPU shared memory via automatic optimization of fine-grained transfers. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 139–152.

Mutlu O, Ghose S, Gómez-Luna J, et al. (2019) Enabling practical processing in and near memory for data-intensive computing. In: *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*. Association for Computing Machinery, pp. 1–4.

Na N, Wang J, Long S, et al. (2017) Exploring DDR4 address bus design for high speed memory interface. In: *2017 IEEE 67th Electronic Components and Technology Conference (ECTC)*. IEEE, pp. 1843–1848.

Nair PJ, Chou CC and Qureshi MK (2014) Refresh pausing in DRAM memory systems. *ACM Transactions on Architecture and Code Optimization* 11(1): 1–26.

Nakamura R and Kuga Y (2023) Multi-Threaded Scp: easy and fast file transfer over SSH. In: *Practice and Experience in*

Advanced Research Computing, PEARC '23. Association for Computing Machinery, pp. 320–323.

Narayan KS and Kumar N (2001) Light responsive polymer field-effect transistor. *Applied Physics Letters* 79(12): 1891–1893.

Nguyen K, Lyu K, Meng X, et al. (2019) Nonblocking DRAM refresh. *IEEE Micro* 39(3): 103–109.

Nicolas Bailon D, Thiers JP and Freudenberger J (2022) Error correction for TLC and QLC NAND flash memories using cell-wise encoding. *Electronics* 11(10): 1585.

Nishi Y, Abe K, Ribo J, et al. (2008) An ASIC-Ready 1.25–6.25Gb/s SerDes in 90nm CMOS with multi-standard compatibility. In: *2008 IEEE Asian Solid-State Circuits Conference*. IEEE, pp. 37–40.

Nitayama A and Aochi H (2011) Vertical 3D NAND flash memory technology. *ECS Transactions* 41(7): 15–25.

Nitin B, Randy W, Shinichiro I, et al. (2018) DDR5 design challenges. In: *2018 IEEE 22nd Workshop on Signal and Power Integrity (SPI)*. IEEE, pp. 1–4.

Noe F, Olsson S, Kohler J, et al. (2019) Boltzmann generators: sampling equilibrium states of many-body systems with deep learning. *Science* 365(6457): eaaw1147.

Noh YY, Kim DY, Yoshida Y, et al. (2005) High-photosensitivity p-channel organic phototransistors based on a biphenyl end-capped fused bithiophene oligomer. *Applied Physics Letters* 86(4): 043501.

Nordrum A (2019) The fight for the future of the disk drive. *IEEE Spectrum* 56(1): 44–47.

Nuth P and Dally W (1995) The Named-State register file: implementation and performance. In: *Proceedings of 1995 1st IEEE Symposium on High Performance Computer Architecture*. IEEE, pp. 4–13.

Olanrewaju RF, Baba A, Khan BUI, et al. (2016) A study on performance evaluation of conventional cache replacement algorithms: A review. In: *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. IEEE, pp. 550–556.

Olson LE, Hill MD and Wood DA (2017) Crossing guard: mediating host-accelerator coherence interactions. In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '17*. Association for Computing Machinery, pp. 163–176.

Ortiz S (2001) Embedded OSs gain the inside track. *Computer* 34(11): 14–16.

Owen J and Steinman M (2008) Northbridge architecture of AMD's griffin microprocessor family. *IEEE Micro* 28(2): 10–18.

Páll S, Abraham MJ, Kutzner C, et al. (2015) Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In: Markidis S and Laure E (eds) *Solving Software Challenges for Exascale*. Springer International Publishing, pp. 3–27.

Pambudi NA, Sarifudin A, Firdaus RA, et al. (2022) The immersion cooling technology: Current and future development in energy saving. *Alexandria Engineering Journal* 61(12): 9509–9527.

Panda P, Patil G and Raveendran B (2016) A survey on replacement strategies in cache memory for embedded systems. In: *2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. IEEE, pp. 12–17.

Panda DK, Subramoni H, Chu CH, et al. (2021) The MVAPICH project: transforming research into high-performance MPI library for HPC community. *Journal of Computational Science* 52: 101208.

Panneerselvam S and Swift M (2016) Rinnegan: efficient resource use in heterogeneous architectures. In: *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT '16*. Association for Computing Machinery, pp. 373–386.

Papamakarios G, Nalisnick E, Rezende DJ, et al. (2021) Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research* 22: 1–64.

Parat K and Goda A (2018) Scaling trends in NAND flash. In: *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, pp. 2.1.1–2.1.4.

Park C, Chung H, Lee YS, et al. (2005) A 512 Mbit, 1.6 Gbps/pin DDR3 SDRAM prototype with C/sub 10/minimization and self-calibration techniques. *Digest of Technical Papers. 2005 Symposium on VLSI Circuits* 2005: 370–373.

Park J, Lee S and Yong K (2012) Photo-stimulated resistive switching of ZnO nanorods. *Nanotechnology* 23(38): 385707.

Park SJ, Kim H, Kim KS, et al. (2022) Scaling of memory performance and capacity with CXL memory expander. In: *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, pp. 1–27.

Park MJ, Lee J, Cho K, et al. (2023) A 192-Gb 12-High 896-GB/s HBM3 DRAM with a TSV Auto-Calibration scheme and Machine-Learning-Based layout optimization. *IEEE Journal of Solid-State Circuits* 58(1): 256–269.

Parvathy N, Upadhyay BR and Sudarshan TSB (2016) Cache coherence: A walkthrough of mechanisms and challenges. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, pp. 2251–2256.

Pasricha S and Dutt N (2010) *On-Chip Communication Architectures: System On Chip Interconnect*. Morgan Kaufmann.

Patel D, Cozma G and Wong G (2023) AMD MI300 – taming- the Hype – AI performance, volume ramp, customers, cost, IO, networking software. https://newsletter.semianalysis.com/p/amd-mi300-taming-the-hype-ai-performance

Paulson L (2005) Will hard drives finally stop shrinking? *Computer* 38(5): 14–16.

Pavan P, Bez R, Olivo P, et al. (1997) Flash memory cells-an overview. *Proceedings of the IEEE* 85(8): 1248–1271.

Peccerillo B, Mannino M, Mondelli A, et al. (2022) A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture* 129: 102561.

Penwarden M, Zhe S, Narayan A, et al. (2023) A metalearning approach for physics-informed neural networks (PINNs): application to parameterized pdes. *Journal of Computational Physics* 477: 111912.

Pernice WHP and Bhaskaran H (2012) Photonic non-volatile memories using phase change materials. *Applied Physics Letters* 101(17): 171101.

Pham-Quoc C, Al-Ars Z and Bertels K (2013) Heterogeneous hardware accelerators interconnect: An overview. In: *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*. IEEE, pp. 189–197.

Phillips JC, Braun R, Wang W, et al. (2005) Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry* 26(16): 1781–1802.

Pinchetti V, Meinardi F, Camellini A, et al. (2016) Effect of core/shell interface on carrier dynamics and optical gain properties of dual-color emitting CdSe/CdS nanocrystals. *ACS Nano* 10(7): 6877–6887.

Pismenny B, Eran H, Yehezkel A, et al. (2021a) Autonomous NIC offloads. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '21*. Association for Computing Machinery, pp. 18–35.

Pismenny B, Olmos JJV, Zack Y, et al. (2021b) Securitization of cloud, edge and IoT communications through hardware accelerations/offloadings. In: *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, pp. 61–64.

Pohl C and Sattler KU (2018) Joins in a heterogeneous memory hierarchy: exploiting high-bandwidth memory. In: *Proceedings of the 14th International Workshop on Data Management on New Hardware, DAMON '18*. Association for Computing Machinery, pp. 1–10.

Prabhakar R, Jairath S and Shin JL (2022) SambaNova SN10 RDU: A 7nm dataflow architecture to accelerate software 2.0. *2022 IEEE International Solid- State Circuits Conference (ISSCC)* 65: 350–352.

Pratoomtong A and Hu YH (2005) On-chip cache algorithm design for multimedia SOC. In: *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005*. IEEE, pp. ii/337–ii/340.

Precedence Research (2024) *AI In Hardware Global Market Opportunities & Strategies*. Report, Precedence Research.

Prince B (1999) A tribute to graphics DRAMs. In: *Records of the 1999 IEEE International Workshop on Memory Technology, Design and Testing*. IEEE, pp. 123–130.

Prince B (2003) Application specific DRAMs today. In: *Records of the 2003 International Workshop on Memory Technology, Design and Testing*. IEEE, pp. 7–13.

Puzyrev V, Salles T, Surma G, et al. (2022) Geophysical model generation with generative adversarial networks. *Geoscience Letters* 9(1): 32.

Qiu Y, Kang Q, Liu M, et al. (2020) Clara: performance clarity for SmartNIC offloading. In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks, HotNets '20*. Association for Computing Machinery, pp. 16–22.

Racunas P and Patt YN (2003) Partitioned first-level cache design for clustered microarchitectures. In: *Proceedings of the 17th Annual International Conference on Supercomputing, ICS '03*. Association for Computing Machinery, pp. 22–31.

Radhakrishnan S, Chinthamani S and Cheng K (2007) The blackford northbridge chipset for the Intel 5000. *IEEE Micro* 27(2): 22–33.

Raissi M, Perdikaris P and Karniadakis G (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 378: 686–707.

Raoux S, Jordan-Sweet JL and Kellock AJ (2008) Crystallization properties of ultrathin phase change films. *Journal of Applied Physics* 103(11): 114310.

Rashdan M, Yousif A, Haslett J, et al. (2009) A new time-based architecture for serial communication links. In: *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*. IEEE, pp. 531–534.

Rashdan M, El-Sayed F and Salman M (2020) Performance comparison between SerDes and Time-Based serial links. In: *2020 7th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, pp. 37–41.

Ravasi M (2025) Geophysical inverse problems with measurement-guided diffusion models. https://arxiv.org/abs/2501.04881

Reed D, Gannon D and Dongarra J (2023) HPC forecast: Cloudy and uncertain. *Communications of the ACM* 66(2): 82–90.

Ren X and Lis M (2017) Efficient sequential consistency in GPUs via relativistic cache coherence. In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 625–636.

Rezende DJ and Mohamed S (2015) Variational inference with normalizing flows. Proceedings of the 32nd International Conference on Machine Learning. In: Bach F and Blei D (eds). *Proceedings of Machine Learning Research* 37: 1530–1538.

Rhu M, Sullivan M, Leng J, et al. (2013) A locality-aware memory hierarchy for energy-efficient GPU architectures. In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46*. Association for Computing Machinery, pp. 86–98.

Rios C, Hosseini P, Wright CD, et al. (2014) On-Chip photonic memory elements employing phase-change materials. *Advanced Materials* 26(9): 1372–1377.

Ríos C, Stegmaier M, Hosseini P, et al. (2015) Integrated all-photonic non-volatile multi-level memory. *Nature Photonics* 9(11): 725–732.

Rixner S (2004) Memory controller optimizations for web servers. In: *37th International Symposium on Microarchitecture (MICRO-37'04)*. IEEE, pp. 355–366.

Rodríguez-Ibarra I, Woo-García RM, Algredo-Badillo I, et al. (2022) Comparative of performance and delays between topologies cells 2T1C, 3T1C, and 4T2C for DRAM. In: *2022 IEEE International Conference on Engineering Veracruz (ICEV)*. IEEE, pp. 1–4.

Rohbani N, Darabi S and Sarbazi-Azad H (2021) PF-DRAM: a Precharge-Free DRAM structure. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 126–138.

Ros A, Davari M and Kaxiras S (2015) Hierarchical private/shared classification: The key to simple and efficient coherence for clustered cache hierarchies. In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 186–197.

Roshan-Zamir A, Elhadidy O, Yang HW, et al. (2017) A Reconfigurable 16/32 Gb/s Dual-Mode NRZ/PAM4 SerDes in 65-nm CMOS. *IEEE Journal of Solid-State Circuits* 52(9): 2430–2447.

Rotem E, Yoaz A, Rappoport L, et al. (2022) Intel alder lake cpu architectures. *IEEE Micro* 42.

Rottmayer R, Batra S, Buechel D, et al. (2006) Heat-Assisted magnetic recording. *IEEE Transactions on Magnetics* 42(10): 2417–2421.

Rudi J, Malossi ACI, Isaac T, et al. (2015) An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15*. Association for Computing Machinery, pp. 1–12.

Rumley S, Bergman K, Seyedi MA, et al. (2020) *Evolving Requirements and Trends of HPC, Chapter 1*. Springer International Publishing, pp. 725–755.

Ryu Y, Ahn SG, Lee JH, et al. (2023) A 16 GB 1024 GB/s HBM3 DRAM with source-synchronized bus design and on-die error control scheme for enhanced RAS features. *IEEE Journal of Solid-State Circuits* 58(4): 1051–1061.

Sabin G and Rashti M (2015) Security offload using the SmartNIC, a programmable 10 Gbps ethernet NIC. In: *2015 National Aerospace and Electronics Conference (NAECON)*. IEEE, pp. 273–276.

Sadasivam SK, Thompto BW, Kalla R, et al. (2017) IBM Power9 processor architecture. *IEEE Micro* 37(2): 40–51.

Saito S, Fujii S, Okada Y, et al. (1985) A 1-Mbit CMOS DRAM with fast page mode and static column mode. *IEEE Journal of Solid-State Circuits* 20(5): 903–908.

Santo B (1988) 1970 1k-Bit DRAM: the LSI era was born in a technical competition between semiconductor and core memories. *IEEE Spectrum* 25(11): 108–112.

Sardashti S and Wood DA (2013) Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching. In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46*. Association for Computing Machinery, pp. 62–73.

Seshadri V, Mutlu O, Kozuch MA, et al. (2012) The evicted-address filter: a unified mechanism to address both cache pollution and thrashing. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, PACT '12*. Association for Computing Machinery, pp. 355–366.

Seshadri V, Lee D, Mullins T, et al. (2017) Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-50 '17*. Association for Computing Machinery, pp. 273–287.

Shafique M and Garg S (2017) Computing in the dark silicon era: current trends and research challenges. *IEEE Design & Test* 34(2): 8–23.

Shah M, Kodali S, Rani A, et al. (2022) Seamless interfacing with ThunderboltTM 4.0. In: *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE, pp. 1–6.

Shalf J (2020) The future of computing beyond Moore's law. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences* 378(2166): 20190061.

Shanbhag A, Tatbul N, Cohen D, et al. (2020) Large-Scale in-memory analytics on Intel® Optane™ DC persistent memory. In: *Proceedings of the 16th International Workshop on Data Management on New Hardware, DaMoN '20*. Association for Computing Machinery, pp. 1–8.

Shao X, Zhang Z, Song P, et al. (2022) A review of energy efficiency evaluation metrics for data centers. *Energy and Buildings* 271: 112308.

Sharma DD (2019) Compute express link. *Whitepaper, Compute Express LinkTM Consortium* 1: 1–3.

Sharma DD (2020a) *Compute Express Link™ 2.0 White Paper*. Whitepaper: Compute Express Link™ Consortium.

Sharma DD (2020b) PCI Express® 6.0 specification at 64.0 GT/s with PAM-4 signaling: a low latency, high bandwidth, high reliability and cost-effective interconnect. In: *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, pp. 1–8.

Sharma DD (2022a) A low-latency and low-power approach for coherency and memory protocols on PCI express 6.0 PHY at 64.0 GT/s with PAM-4 signaling. *IEEE Micro* 42(2): 37–43.

Sharma DD (2022b) *Compute express link 3.0*. Whitepaper, Compute Express LinkTM Consortium.

Sharma DD (2022c) Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In: *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, pp. 5–12.

Sharma DD (2023a) Compute Express Link (CXL): enabling heterogeneous data-centric computing with heterogeneous memory hierarchy. *IEEE Micro* 43(2): 99–109.

Sharma DD (2023b) System on a package innovations with Universal Chiplet Interconnect Express (UCIe) interconnect. *IEEE Micro* 43(2): 76–85.

Shaw DE, Deneroff MM, Dror RO, et al. (2007a) Anton, a special-purpose machine for molecular dynamics simulation. In: *Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07*. Association for Computing Machinery, pp. 1–12.

Shaw DE, Deneroff MM, Dror RO, et al. (2007b) Anton, a special-purpose machine for molecular dynamics simulation. *ACM SIGARCH Computer Architecture News* 35(2): 1–12.

Shaw DE, Dror RO, Salmon JK, et al. (2009a) Millisecond-scale molecular dynamics simulations on Anton. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*. Association for Computing Machinery, pp. 1–11.

Shaw DE, Dror RO, Salmon JK, et al. (2009b) Millisecond-scale molecular dynamics simulations on Anton. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*. Association for Computing Machinery, pp. 1–11.

Shaw DE, Adams PJ, Azaria A, et al. (2021) Anton 3: twenty microseconds of molecular dynamics simulation before Lunch. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21*. Association for Computing Machinery, pp. 1–11.

Shi R, Kara K, Hagleitner C, et al. (2022) Exploiting HBM on FPGAs for data processing. *ACM Transactions on Reconfigurable Technology and Systems* 15(4): 1–27.

Shibata N, Kanda K, Shimizu T, et al. (2019) 13.1 A 1.33Tb 4-bit/Cell 3D-Flash memory on a 96-Word-Line-Layer technology. In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. IEEE, pp. 210–212.

Shih B, Peyvan A, Zhang Z, et al. (2025) Transformers as neural operators for solutions of differential equations with finite regularity. *Computer Methods in Applied Mechanics and Engineering* 434: 117560.

Shim S, Kim S, Bae J, et al. (2018) A 16Gb 1.2V 3.2Gb/s/pin DDR4 SDRAM with improved power distribution and repair strategy. In; *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. IEEE, pp. 212–214.

Shiratake S (2020) Scaling and performance challenges of future DRAM. In: *2020 IEEE International Memory Workshop (IMW)*. IEEE, pp. 1–3.

Shiroishi Y, Fukuda K, Tagawa I, et al. (2009) Future options for HDD storage. *IEEE Transactions on Magnetics* 45(10): 3816–3822.

Shivam P and Chase JS (2003) On the elusive benefits of protocol offload. In: *Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications, NICELI '03*. Association for Computing Machinery, pp. 179–184.

Shuang-yan C, Dong-hui W, Rui S, et al. (2005) An innovative design of the DDR/DDR2 SDRAM compatible controller. *2005 6th International Conference on ASIC* 1: 62–66.

Si M, Cheng HY, Ando T, et al. (2021) Overview and outlook of emerging non-volatile memories. *MRS Bulletin* 46(10): 946–958.

Siau C, Kim KH, Lee S, et al. (2019) 13.5 A 512Gb 3-bit/Cell 3D flash memory on 128-Wordline-Layer with 132MB/s write performance featuring circuit-under-array technology. In: *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*. IEEE, pp. 218–220.

Silvagni A (2016) *3D VG-Type NAND Flash Memories*. Springer Netherlands, pp. 197–222.

Simão C, Mas-Torrent M, Casado-Montenegro J, et al. (2011) A three-state surface-confined molecular switch with multiple channel outputs. *Journal of the American Chemical Society* 133(34): 13256–13259.

Singh I, Shriraman A, Fung WWL, et al. (2013) Cache coherence for GPU architectures. In; *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 578–590.

Skamarock WC, Klemp JB, Dudhia J, et al. (2019) *A Description of the Advanced Research WRF Version 4*. Technical Report NCAR/TN-556+STR. National Center for Atmospheric Research.

Smith AJ (1982) Cache memories. *ACM Computing Surveys* 14(3): 473–530.

Sodani A (2015) Knights Landing (KNL): 2nd generation Intel® Xeon Phi processor. In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE, pp. 1–24.

Sohl-Dickstein J, Weiss EA, Maheswaranathan N, et al. (2015) Deep unsupervised learning using nonequilibrium thermodynamics. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15*. IEEE, pp. 2256–2265.

Sohn K, Na T, Song I, et al. (2013) A 1.2 V 30 nm 3.2 Gb/s/pin 4 Gb DDR4 SDRAM with dual-error detection and PVT-Tolerant data-fetch scheme. *IEEE Journal of Solid-State Circuits* 48(1): 168–177.

Solihin Y, Lee J and Torrellas J (2003) Correlation prefetching with a user-level memory thread. *IEEE Transactions on Parallel and Distributed Systems* 14(6): 563–580.

Sommer L, Korinth J and Koch A (2017) OpenMP device offloading to FPGA accelerators. In: *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, pp. 201–205.

Stanzione D (2022) Thoughts from TACC on next generation academic supercomputing systems. *OU Supercomputing Symposium*. https://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/22/Wednesday_TACC_Stanzione.pdf

Stanzione D, Barth B, Gaffney N, et al. (2017) Stampede 2: the evolution of an XSEDE Supercomputer. In: *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, PEARC17*. Association for Computing Machinery, pp. 1–8.

Stanzione D, West J, Evans RT, et al. (2020) Frontera: the evolution of leadership computing at the national science foundation. In: *Practice and Experience in Advanced Research Computing, PEARC '20*. Association for Computing Machinery, pp. 106–111.

Stegmaier M, Ríos C, Bhaskaran H, et al. (2017) Nonvolatile all-optical 1 × 2 switch for chipscale photonic networks. *Advanced Optical Materials* 5(1): 1600346.

Stojkovic D, Shringarpure M, Kim Y, et al. (2024) Improving reservoir characterization and prediction via machine

learning-driven integration of subseismic geologic concepts, geophysical attributes, and wells. *SPE Journal* 29(04): 1967–1973.

Stuecheli J, Blaner B, Johns CR, et al. (2015) CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development* 59(1): 7:1–7:7.

Su W, Wang L, Su M, et al. (2011) A processor-DMA-Based memory copy hardware accelerator. In: *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*. IEEE, pp. 225–229.

Suggs D, Subramony M and Bouvier D (2020) The AMD "Zen 2" processor. *IEEE Micro* 40(2): 45–52.

Sun Y, Mukherjee S, Baruah T, et al. (2018) Evaluating performance tradeoffs on the radeon open compute platform. In: *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, pp. 209–218.

Sundar H, Biros G, Burstedde C, et al. (2012) Parallel geometric-algebraic multigrid on unstructured forests of octrees. In: *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–11.

Sur S, Koop MJ, Chai L, et al. (2007) Performance analysis and evaluation of mellanox ConnectX InfiniBand architecture with multi-core platforms. In: *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*. IEEE, pp. 125–134.

Swirbel T, Naujoks A and Watkins M (1999) Electrical design and simulation of high density printed circuit boards. *IEEE Transactions on Advanced Packaging* 22(3): 416–423.

Talati N, Gupta S, Mane P, et al. (2016) Logic design within memristive memories using memristor-Aided loGIC (MAGIC). *IEEE Transactions on Nanotechnology* 15(4): 635–650.

Tan Y, Wang B, Yan Z, et al. (2019) UIMigrate: adaptive data migration for hybrid non-volatile memory systems. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 860–865.

Tanaka H, Kido M, Yahashi K, et al. (2007) Bit cost scalable technology with punch and plug process for ultra high density flash memory. In: *2007 IEEE Symposium on VLSI Technology*. IEEE, pp. 14–15.

Tang T, Wray B and Murugan R (2020) Die-Package-PCB signal integrity performance debug of a high-speed (25Gbps) retimer: simulation to measurement correlation. In: *2020 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)*. IEEE, pp. 170–175.

Tao J and Karl W (2006) Detailed cache simulation for detecting bottleneck, miss reason and optimization potentialities. In: *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools, valuetools '06*. Association for Computing Machinery, p. 62.

Tasdemir K, Khan R, Siddiqui F, et al. (2023) An investigation of machine learning algorithms for high-bandwidth SQL injection detection utilising BlueField-3 DPU technology. In: *2023 IEEE 36th International System-on-Chip Conference (SOCC)*. IEEE, pp. 1–6.

Teh MY, Wu Z, Glick M, et al. (2022) Performance trade-offs in reconfigurable networks for HPC. *Journal of Optical Communications and Networking* 14(6): 454–468.

The Business Research Company (2023) *AI In Hardware Global Market Opportunities & Strategies*. Report, The Business Research Company.

Thompson DA and Best JS (2000) The future of magnetic data storage techology. *IBM Journal of Research and Development* 44(3): 311–322.

Thompson NC and Spanuth S (2021) The decline of computers as a general purpose technology. *Communications of the ACM* 64(3): 64–72.

Thompson AP, Aktulga HM, Berger R, et al. (2022) LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications* 271: 108171.

Tirumala A and Wong R (2024) NVIDIA blackwell platform: advancing Generative AI and Accelerated Computing. In: *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, pp. 1–33.

Torres EF, Ibañez P, Viñals V, et al. (2004) Contents management in first-level multibanked data caches. In: Danelutto M, Vanneschi M and Laforenza D (eds) *Euro-Par 2004 Parallel Processing*. Springer Berlin Heidelberg, pp. 516–524.

Towles B, Grossman JP, Greskamp B, et al. (2014) Unifying on-chip and inter-node switching within the Anton 2 network. In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture, ISCA '14*. IEEE Press, pp. 1–12.

Tran TT (2023) *Transmission Line (TL) Effects in Frequency Domain*. Springer International Publishing, 95–102.

Trancoso P (2005) Dynamic split: flexible border between instruction and data cache. In: *8th Euromicro Conference on Digital System Design (DSD'05)*. IEEE, pp. 476–483.

Trivedi A and Brunella MS (2023) CPU-free computing: a vision with a blueprint. In: *Proceedings of the 19th Workshop on Hot Topics in Operating Systems, HOTOS '23*. Association for Computing Machinery, pp. 1–14.

Trott CR, Lebrun-Grandié D, Arndt D, et al. (2022) Kokkos 3: programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems* 33(4): 805–817.

Tsai PA, Gan YL and Sanchez D (2018) Rethinking the memory hierarchy for modern languages. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, pp. 203–216.

Tsog N, Mubeen S, Bruhn F, et al. (2021) Offloading accelerator-intensive workloads in CPU-GPU heterogeneous processors. In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, pp. 1–8.

Tsuji H and Nakamura E (2017) Design and functions of semiconducting fused polycyclic furans for optoelectronic applications. *Accounts of Chemical Research* 50(2): 396–406.

Turner RE (2023) An introduction to Transformers. https://arxiv.org/abs/2304.10557

Usui H, Subramanian L, Chang KKW, et al. (2016) DASH: Deadline-Aware High-Performance memory scheduler for heterogeneous systems with hardware accelerators. *ACM Transactions on Architecture and Code Optimization* 12(4): 1–28.

Valero A, Petit S, Sahuquillo J, et al. (2012) Design, performance, and energy consumption of eDRAM/SRAM macrocells for L1 data caches. *IEEE Transactions on Computers* 61(9): 1231–1242.

Vamvakos SD, Gauthier CR, Rao C, et al. (2012) A 2.488–11.2 Gb/s multi-protocol SerDes in 40nm low-leakage CMOS for FPGA applications. In: *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, pp. 5–8.

van Houdt J (2006) Flash memory: a challenged memory technology. In: *2006 IEEE International Conference on IC Design and Technology*. IEEE, pp. 1–4.

Vaswani A, Shazeer N, Parmar N, et al. (2017) Attention is all you need. *Advances in Neural Information Processing Systems*. Curran Associates Inc., 5998–6008.

Villar JA, Andújar FJ, Sánchez JL, et al. (2013) Obtaining the optimal configuration of high-radix combined switches. *Journal of Parallel and Distributed Computing* 73(9): 1239–1250.

Villela R (2020) *Memory Management, Chapter 1*. Apress, pp. 1–12.

Vizitiu A, Itu L, Niţă C, et al. (2014) Optimized three-dimensional stencil computation on Fermi and Kepler GPUs. In: *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, pp. 1–6.

Waddington D, Kunitomi M, Dickey C, et al. (2019) Evaluation of Intel 3D-Xpoint NVDIMM technology for memory-intensive genomic workloads. In: *Proceedings of the International Symposium on Memory Systems, MEMSYS '19*. Association for Computing Machinery, pp. 277–287.

Wang W and Buckwalter JF (2011) Double-edge pulsewidth modulation for multidrop backplanes. In: *2011 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*. IEEE, pp. 1–4.

Wang Z and Lee RB (2008) A novel cache architecture with enhanced performance and security. In: *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, pp. 83–93.

Wang W, Wang Q, Wei W, et al. (2008) Modeling and evaluating heterogeneous memory architectures by trace-driven simulation. In: *Proceedings of the 2008 Workshop on Memory Access on Future Processors: A Solved Problem? MAW '08*. Association for Computing Machinery, pp. 369–376.

Wang S, Prakash A and Mitra T (2018) Software support for heterogeneous computing. In: *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, pp. 756–762.

Wang Y, Yang Y, Han C, et al. (2019) LR-LRU: A PACS-oriented intelligent cache replacement policy. *IEEE Access* 7: 58073–58084.

Wang J, Park S and Park CS (2021a) Optimization of communication schemes for DMA-controlled accelerators. *IEEE Access* 9: 139228–139247.

Wang K, Chen Y, Mehana M, et al. (2021b) A physics-informed and hierarchically regularized data-driven model for predicting fluid flow through porous media. *Journal of Computational Physics* 443: 110526.

Wang Q, Ihme M, Chen YF, et al. (2022) A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units. *Computer Physics Communications* 274: 108292.

Wehn N and Hein S (1998) Embedded DRAM architectural trade-offs. *Proceedings Design, Automation and Test in Europe*. IEEE, 704–708.

Wei Y, Huang YC, Tang H, et al. (2023) 9.3 NVLink-C2C: a coherent off package Chip-To-Chip interconnect with 40Gbps/pin single-ended signaling. In: *2023 IEEE International Solid- State Circuits Conference (ISSCC)*. IEEE, pp. 160–162.

Weiland M, Brunst H, Quintino T, et al. (2019) An early evaluation of intel's optane DC persistent memory module and its impact on high-performance scientific applications. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19*. Association for Computing Machinery, pp. 1–19.

Weste N and Harris D (2010) *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th edition. Addison-Wesley Publishing Company.

Westerink J and Luettich R (2024) Advanced circulation (ADCIRC) version 56.0.3.

Winterberg D, Kumar V, Chen T, et al. (2023) DDR5 electrical challenges in high-speed server design. *2023 Joint Asia-Pacific International Symposium on Electromagnetic Compatibility and International Conference on ElectroMagnetic Interference & Compatibility (APEMC/INCEMIC)*. IEEE, 1–4.

Wolf ME and Lam MS (1991) A data locality optimizing algorithm. *ACM SIGPLAN Notices* 26(6): 30–44.

Wong G (2010) *Market and Applications for NAND Flash Memories, Chapter 1*. Springer Netherlands, pp. 1–18.

Woo NS (2010) High performance SOC for mobile applications. In: *2010 IEEE Asian Solid-State Circuits Conference*. IEEE, pp. 1–4.

Woo Y, Kim T and Jung S (2020) Persistent memory needs lifetime extension techniques. In: *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, pp. 1439–1441.

Wood R (2000) The feasibility of magnetic recording at 1 Terabit per square inch. *IEEE Transactions on Magnetics* 36(1): 36–42.

Wu H, Liao Y, Gao B, et al. (2016) *RRAM Cross-Point Arrays, Chapter 8*. Springer Netherlands, pp. 223–260.

Yahata H, Okuda Y, Miyashita H, et al. (2000) A 256-Mb double-data-rate SDRAM with a 10-mW analog DLL circuit. In: *2000 Symposium on VLSI Circuits. Digest of Technical Papers (Cat. No.00CH37103)*. IEEE, pp. 74–75.

Yan F, Yuan C, Li C, et al. (2021) FOSquare: a novel optical HPC interconnect network architecture based on fast optical switches with distributed optical flow control. *Photonics* 8(1): 11.

Yang G and Sommer S (2023) A denoising diffusion model for fluid field prediction. arXiv preprint arXiv:2301.11661. https://arxiv.org/abs/2301.11661

Yang J, Li B and Lilja DJ (2020) Exploring performance characteristics of the optane 3D Xpoint storage technology. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 5(1): 1–28.

Yang L, Meng X and Karniadakis GE (2021) B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics* 425: 109913.

Yazdanbakhsh A, Thwaites B, Esmaeilzadeh H, et al. (2016) Mitigating the memory bottleneck with approximate load value prediction. *IEEE Design & Test* 33(1): 32–42.

Ye C, Peng Q, Li M, et al. (2012) Multilevel conductance switching of memory device through photoelectric effect. *Journal of the American Chemical Society* 134(49): 20053–20059.

Yi B, Xia J, Chen L, et al. (2017) Towards zero copy dataflows using RDMA. In: *Proceedings of the SIGCOMM Posters and Demos, SIGCOMM Posters and Demos '17*. Association for Computing Machinery, pp. 28–30.

Yin S, Kim Y, Han X, et al. (2019) Monolithically integrated RRAM- and CMOS-Based In-Memory computing optimizations for efficient deep learning. *IEEE Micro* 39(6): 54–63.

Yogatama BW, Gong W and Yu X (2022) Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS. *Proceedings of the VLDB Endowment* 15(11): 2491–2503.

Yoo HJ (1997) A study of pipeline architectures for high-speed synchronous DRAMs. *IEEE Journal of Solid-State Circuits* 32(10): 1597–1603.

Yoon JB and Malone N (2022) *Application Note: PCIe Link Training Overview*. In: Application Note, Texas Instruments Incorporated.

Yoon CW, Im YK, Han SH, et al. (1999a) A fast synchronous pipelined DRAM (SP-DRAM) architecture with SRAM buffers. In: *ICVC '99. 6th International Conference on VLSI and CAD (Cat. No.99EX361)*. IEEE, pp. 285–288.

Yoon H, Cha GW, Yoo C, et al. (1999b) A 2.5-V, 333-Mb/s/pin, 1-Gbit, double-data-rate synchronous DRAM. *IEEE Journal of Solid-State Circuits* 34(11): 1589–1599.

Young C, Bank JA, Dror RO, et al. (2009) A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*. Association for Computing Machinery, pp. 1–11.

Yu Y, Beyls K and D'Hollander E (2001) Visualizing the impact of the cache on program execution. In: *Proceedings Fifth International Conference on Information Visualisation*. IEEE, pp. 336–341.

Yu J, Lu L, Meng X, et al. (2022) Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Computer Methods in Applied Mechanics and Engineering* 393: 114823.

Yue P and Shekhar S (2022) F4: paving the way to 200Gb/s transceivers. *2022 IEEE International Solid- State Circuits Conference (ISSCC)* 65: 537–539.

Zahran M (2017) Heterogeneous computing: here to stay. *Communications of the ACM* 60(3): 42–45.

Zambelli C, Chimenton A and Olivo P (2010) *Reliability Issues of NAND Flash Memories*. Springer Netherlands, pp. 89–113.

Zhai Y, Yang JQ, Zhou Y, et al. (2018) Toward non-volatile photonic memory: concept, material and design. *Materials Horizons* 5: 641–654.

Zhang Z (2021) Analysis of the Advantages of the M1 CPU and Its Impact on the Future Development of Apple. In: *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. IEEE, pp. 732–735.

Zhang J, Li P, Liu B, et al. (2018) Performance analysis of 3D XPoint SSDs in virtualized and non-virtualized environments. In: *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, pp. 1–10.

Zhang R, Liu T, Yang K, et al. (2020a) SRAM stability analysis and performance–reliability tradeoff for different cache configurations. *IEEE Transactions on Very Large Scale Integration Systems* 28(3): 620–633.

Zhang Y, Jia Z, Pan Y, et al. (2020b) PattPIM: a practical ReRAM-Based DNN accelerator by reusing weight pattern repetitions. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

Zhang C, Song X and Azevedo L (2021) U-net generative adversarial network for subsurface facies modeling. *Computational Geosciences* 25(1): 553–573.

Zhang Y, Zhao Y, Dai S, et al. (2022) Cooling technologies for data centres and telecommunication base stations – A comprehensive review. *Journal of Cleaner Production* 334: 130280.

Zhang J, Huang H, Zhu L, et al. (2023) SmartDS: middle-tier-centric SmartNIC enabling application-aware message split for disaggregated block storage. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 23*. Association for Computing Machinery, pp. 1–13.

Zhang H, Li Y and Huang J, et al. (2024) Diffusionvel: multi-information integrated velocity inversion using generative diffusion models. arXiv preprint arXiv:2410.21776. https://arxiv.org/abs/2410.21776

Zhao J, Zou Q and Xie Y (2017) Overview of 3-D architecture design opportunities and techniques. *IEEE Design & Test* 34(4): 60–68.

Zhong G, Dubey A, Tan C, et al. (2019) Synergy: an HW/SW framework for high throughput CNNs on embedded heterogeneous SoC. *ACM Transactions on Embedded Computing Systems* 18(2): 1–23.

Zhou H, Ma Y, Wu H, et al. (2024) Transolver: a fast transformer solver for PDEs on general geometries. Proceedings of the 41st International Conference on Machine Learning. *Proceedings of Machine Learning Research*, 235. https://arxiv.org/abs/2402.02366

Zhu Z, Zhang Z and Zhang X (2002) Fine-grain priority scheduling on multi-channel memory systems. In; *Proceedings Eighth International Symposium on High Performance Computer Architecture*. IEEE, pp. 107–116.

Zhu JG, Zhu X and Tang Y (2008) Microwave assisted magnetic recording. *IEEE Transactions on Magnetics* 44(1): 125–131.

Zhuang J, Yang Z and Zhou P (2023) High performance, low power matrix multiply design on ACAP: from architecture, design challenges and DSE perspectives. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

Zuckerman J, Giri D, Kwon J, et al. (2021) Cohmeleon: learning-based orchestration of accelerator coherence in heterogeneous SoCs. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21*. Association for Computing Machinery, pp. 350–365.

Author biographies

Bagus Hanindhito was a PhD student at the Laboratory of Computer Architecture, Department of Electrical and Computer Engineering, The University of Texas at Austin. His research interests include workload characterization, performance evaluation, and architecture-aware optimizations of HPC and AI/ML applications on GPU-accelerated computing clusters and emerging accelerators. After graduating with his PhD in 2024, he works as a Principal Engineer in the Chief Technology Office of the Infrastructure Solutions Group at Dell Technologies. He received his MSE in Computer Engineering from The University of Texas at Austin in 2020, and earned a BS and an MS in Electrical Engineering from Institut Teknologi Bandung, Indonesia, in 2015 and 2019, respectively.

Arash Fathi is a computational scientist at ExxonMobil with broad interests in computational engineering, mathematics, and HPC. At ExxonMobil Corporate Strategic Research, he led a hardware–algorithm co-design project, developing specialized hardware and tailored algorithms to maximize the computing efficiency of wave simulations. He has also explored the potential of quantum computing for solving PDEs prevalent in the oil and gas industry. Arash has organized several symposiums on novel computational algorithms for future computing platforms and played a key role in projects involving PDE-constrained optimization, uncertainty quantification, scientific machine learning, emission detection, and digital twins.

Dimitrios Gourounas is pursuing his PhD in the System-Level Architecture and Modeling (SLAM) group at the University of Texas at Austin. His research interests lie in reconfigurable architectures targeting high-performance computing and machine learning workloads. His work includes the design of a reconfigurable accelerator for memory-bound, discontinuous-Galerkin–based PDE solvers and automated frameworks for generating high-efficiency matrix-multiplication units on modern AI-optimized FPGA architectures. He holds a BSc in Electrical and Computer Engineering from the National Technical University of Athens.

Dimitar Trenev is a Computational Scientist at ExxonMobil Technology and Engineering Company, focusing on high-performance computing, quantum computing, and numerical analysis. His work centers on developing and applying advanced computational methods to challenging problems in the energy industry. Dimitar has played key roles in projects involving large-scale seismic inversion, reservoir simulation, uncertainty quantification and other compute-intensive workflows. He also led a collaboration between ExxonMobil and IBM exploring the use of quantum computing for energy applications.

Andreas Gerstlauer received the PhD degree in Information and Computer Science (ICS) from the University of California at Irvine (UCI), Irvine, CA, USA, in 2004. He is a Cullen Trust for Higher Education Professor in the Chandra Family Department of Electrical and Computer Engineering at The University of Texas at Austin (UT Austin), Austin, TX, USA. Prior to joining UT Austin in 2008, he was an Assistant Researcher with the Center for Embedded Computer Systems (CECS) at UCI. His research interests include systems-level design automation, system modeling, design languages and methodologies, and embedded hardware and software synthesis. Prof. Gerstlauer's work was recognized with several best paper awards and nominations from major conferences, such as DAC, DATE, and HOST, as one of the most influential contributions in ten years at DATE in 2008, and as recipient of a 2016–2017 Humboldt Research Fellowship. He serves or has served as an Editor for ACM TECS and TODAES journals, as well as the General or Program Chair for major international conferences such as ESWEEK.

Lizy Kurian John holds the Truchard Foundation Chair in Engineering in the Department of Electrical and Computer Engineering at The University of Texas at Austin. Her research is in the areas of computer architecture, multicore processors, memory systems, performance evaluation and benchmarking, workload characterization, and reconfigurable computing. She has published four books, 300+ refereed journal and conference publications and holds 20 U. S. patents. Prof. John was the Editor-in-Chief of IEEE Micro from 2019-2023. She is an IEEE Fellow, ACM Fellow, AAAS Fellow, and Fellow of the National Academy of Inventors.

A list of acronyms

ACAP	Adaptive Compute Acceleration Platform
AI	Artificial Intelligence
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
APB	Advanced Peripheral Bus
ASB	Advanced System Bus
ASIC	Application-Specific Integrated Circuit
CAE	Computer-Aided Engineering
CAGR	Compound Annual Growth Rate
CC	CUDA Cores

CDI	Composable/Disaggregated Infrastructure
CFD	Computational Fluid Dynamics
CMOS	Complementary MOSFET
CMR	Conventional Magnetic Recording
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CUDA	Compute Unified Device Architecture
CXL	Compute Express Link
DDR	Double-Data Rate
dG	discontinuous Galerkin
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DOE	Department of Energy
DPU	Data Processing Unit
DRAM	Dynamic Random Access Memory
EDO	Extended Data Out
EDR	Extended Data Rate
FDR	Fourteen Data Rate
FEC	Forward Error Correction
FET	Field-Effect Transistors
FIFO	First-In-First-Out
FLOPS	Floating-Point Operations Per Second
FPGA	Field-Programmable Gate Array
FPM	Fast Page Mode
GDDR	Graphics Double-Data Rate
GEMM	General Matrix-Matrix multiplication
GPU	Graphics Processing Unit
HBM	High-Bandwidth Memory
HCA	Host Channel Adapter
HDD	Hard Disk Drive
HDL	Hardware Description Language
HDR	High Data Rate
HMC	Hybric Memory Cube
HPC	High Performance Computing
HTT	HyperTransport Technology
IBTA	InfiniBand Trade Association
IBM	International Business Machines corporation
IFIS	Infinity Fabric Inter-Socket
ILP	Integer-Linear-Programming
IO	Input/Output
IP	Internet Protocol
IPU	Intelligence Processing Unit
JEDEC	Joint Electron Device Engineering Council
LFU	Least Frequently Used
LRU	Least Recently Used
LPDDR	Low-Power Double-Data Rate
MCDRAM	Multi-Channel Dynamic Random Access Memory
MD	Molecular Dynamics
ML	Machine Learning
MOSFET	Metal Oxide Semiconductor Field-Effect Transistors
MPI	Message Passing Interface
MUX	Multiplexer
NDR	Next Data Rate
NGIO	Next Generation I/O
NIC	Network Interface Card
NMP	Near-Memory Processing
NoC	Network on Chip
NVM	Non-Volatile Memory
NRC	Nuclear Regulatory Commission
NRZ	Non-Return to Zero
NSF	National Science Foundation
OCP	Open Compute Project
ODE	Ordinary Differential Equation
OOO	Out-Of-Order
PAM	Pulse-Amplitude Modulation
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
PCM	Phase-Change Memory
PDE	Partial Differential Equation
PIM	Processing-In-Memory
PUE	Power Usage Effectiveness
QDR	Quad-Data Rate
RDMA	Remote Direct Memory Access
RoCE	RDMA over Converged Ethernet
ROCm	Radeon Open Compute platform
RR	Random Replacement
RRAM	Resistive Random Access Memory
RTL	Register Transfer Level
SAM	Serial Access Memory
SDR	Single-Data Rate
SerDes	Serializer-Deserializer
SIMD	Single-Instruction Multiple-Data
SM	Streaming Multiprocessor
SMR	Shingled Magnetic Recording
SMR	Small Modular Reactor
SP	Streaming Processor
SoC	System on Chip
SPUE	Server Power Usage Effectiveness
SRAM	Static Random-Access Memory
SSD	Solid-State Drive
TACC	Texas Advanced Computing Center
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TPU	Tensor Processing Unit
TSMC	Taiwan Semiconductor Manufacturing Company
TSV	Through-Silicon Vias
UART	Universal Asynchronous Receiver/ Transmitter
UCIe	Universal Chiplet Interconnect express
USB	Universal Serial Bus
VCT	Vertical Channel Transistor
VLIW	Very Long Instruction Words
VRAM	Video Random-Access Memory
WSE	Wafer Scale Engine
XDR	Extreme Data Rate.