

DOI:10.1145/3418291

As a new era in computing emerges, so too must our fundamental thinking patterns.

BY YUHANG LIU, XIAN-HE SUN, YANG WANG, AND YUNGANG BAO

HCUDA: From Computational Thinking to a Generalized Thinking Paradigm

IN 2006, JEANNETTE M. Wing⁴⁵ proposed the concept of “computational thinking,” which has produced significant worldwide impacts on the education, research, and development of computer science. After more than a decade, we reexamine computational thinking, and find that a more general-thinking paradigm is urgently needed to address new challenges.

A couple of recent commentaries^{12,41} regarding computational thinking attracted our interests and inspired us to reflect further. More than that, we want to summarize and generalize the rationale of our solutions,

for instance, the Labeled von Neumann Architecture (LvNA)^{1,28} and the Layered Performance Matching (LPM) methodology.^{26,27}

Nurtured by Moore’s Law, the number of transistors available on a single chip increases exponentially. Meanwhile, due to architectural innovations, transistors are organized more effectively and utilized more vigorously. As a result of the combined efforts, computers have witnessed a significant performance advancement during their 70-year history. However, the new age, characterized by the slowdown of Moore’s Law and Dennard scaling,⁴⁴ and by the rise of big data applications, brings serious challenges that computer scientists must face.

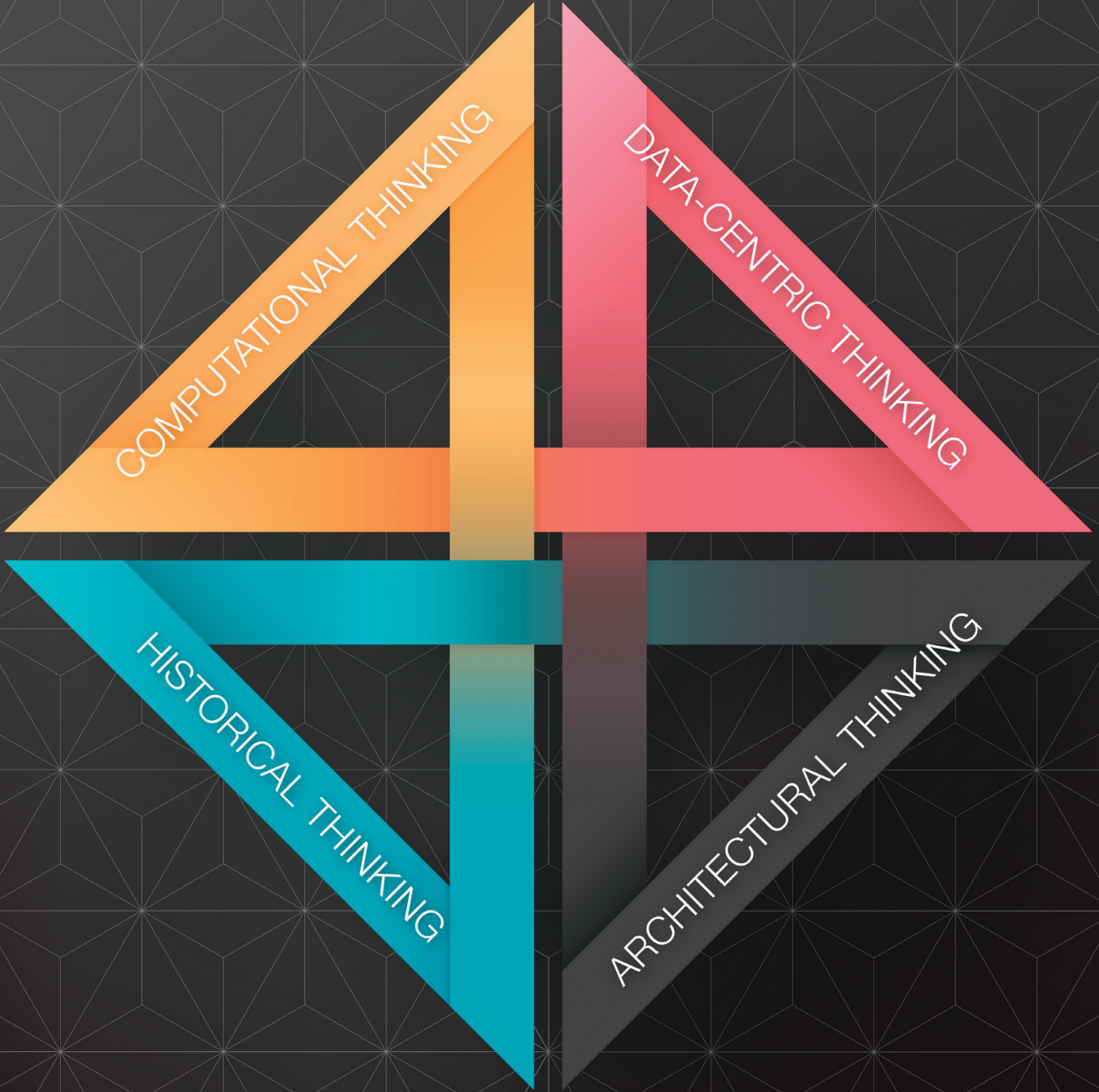
The scaling of on-chip transistors impacts microprocessor performance significantly. However, further improvements to transistor density and power become more difficult due to the limits of semiconductor physics.⁴⁴ As a result, architectural innovations become increasingly crucial for performance breakthroughs, and the epoch we are entering is “a new golden age for computer architecture.”¹⁴

The rise of big data has caused an unprecedented shift, where the memory system, rather than the computational core, plays a more vital role. Accordingly, the memory access limitation de-

» key insights

- **Architecturally organizing the computing system and exploiting the available transistors more effectively can reduce the number of transistors required, and the labeling and matching are crucial in architectural design.**
- **Computing history is a growing warehouse that has not been well utilized, and the evolution of computing technology is not always linear but has lots of “reuses” or “rising from the ashes,” and hence historical thinking is needed.**
- **Memory access can become a killer performance bottleneck for high performance computing, and simply adding more on chip memories is not a feasible solution to the memory-wall problem.**

IMAGE BY ANDREW KRASOVITCKII



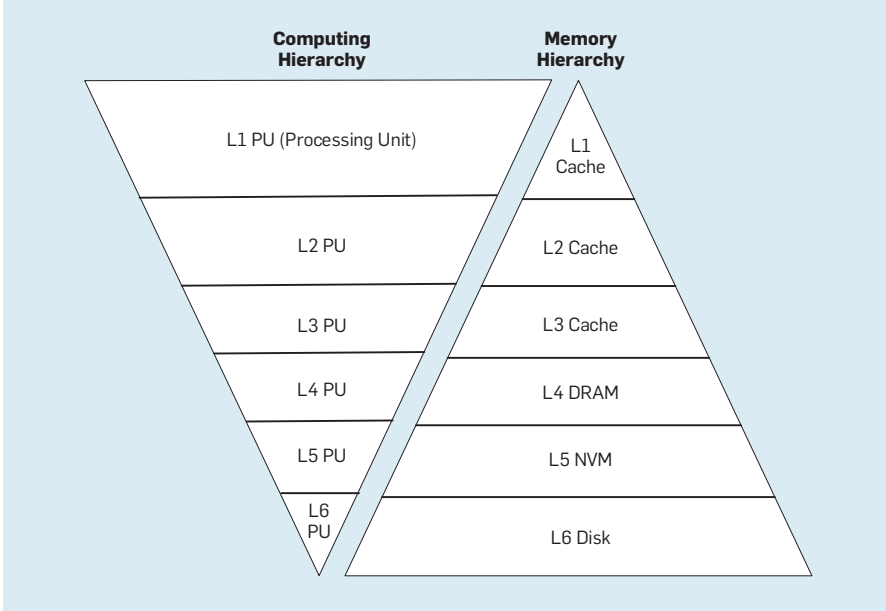
COMPUTATIONAL THINKING

DATA-CENTRIC THINKING

HISTORICAL THINKING

ARCHITECTURAL THINKING

Figure 1. The symmetry between memory and computation from a hierarchy perspective.



scribed by Sun-Ni’s Law³⁸ is becoming a performance killer for many applications. Thus, data-centric innovations of computer system design are urgently needed to address the issues of data storage and access. Both the emergence of big data and the slowdown of Moore’s Law have changed the landscape of computer systems and require us to examine past solutions to pave a new path for future innovations and for addressing new challenges.

The first step to learn from extant innovations is to find commonalities and patterns within them. In the seminal literature “The Art of Scientific Research,”³ Beveridge summarized the research methods adopted by classical natural scientists and found commonalities, routines and regularities in the innovation process, which was previously thought to be accidental. By following Beveridge’s analysis methodology, from a thinking pattern perspective, we try to summarize and therefore provide some useful hints for the art of computer science research.

One interesting observation that motivates us in this study is the symmetry between computation (data processing) and memory (data access). The name “computer” is somewhat misleading, leaving an impression that the only focus of “computer” is computation. In contrast to the traditional computation-centric design, the newly emerged memory-centric computer system design views memory as the in-

trinsic bottleneck of the von Neumann architecture.

Before turning to non von Neumann architectures (for example, quantum computing²⁰ and DNA computing⁶), we should determine whether it is possible to design new von Neumann architectures that are still transistor-based but can overcome the memory wall. After examining the original representation of von Neumann,^{42,43} we found that von Neumann preserves neutrality between computation and memory. That is, data processing and data access are equally important according to the von Neumann architecture. Computation and data are two sides of the same coin. They are the premises of each other. Therefore, equilibrium (a.k.a. balanced) design is the must without any room for maneuver.

The traditional computation-centric architectural design is due to historical reasons. In the beginning, CPUs were very slow and expensive compared to memory. As a result, computer systems are computation-centric, focusing on the speed of computing components. Later, due to the mismatch in advancement of processor and memory technologies, computer systems became increasingly memory-centric where data was treated as a first-class citizen. From a historical perspective, the later memory-centric focus counteracts the former computation-centric focus.

More specifically, as shown in Fig-

ure 1, the memory hierarchy is the counterpart of the computational hierarchy, memory-level-parallelism (MLP) is the counterpart of instruction-level-parallelism (ILP), and processor in memory (PIM) is the counterpart of memory in processor (MIP).^a Therefore, in this study, we propose a framework that allows computational thinking and data-centric thinking to naturally coexist.

The thought process of a computer system designer is dynamic, but it has patterns. Wing introduced the important concept of “computational thinking” in 2006.⁴⁵ In this study, we build on her ideas to extend the concept of computational thinking and present a paradigm of thinking patterns for computer researchers and practitioners in the new era of computing.

More generally, we identify four foundational thinking patterns that will foster computer system innovations: *historical thinking, computational thinking, data-centric thinking, and architectural thinking* (HDCA). The combination of these four patterns forms a regular tetrahedron thinking paradigm, where history is the source of references and lessons, computation is the functionality provided by the computing systems, data is the processing objects of computing systems, and architecture is the hardware mechanism to physically execute operations. The four thinking patterns are four dimensions that are logically connected, providing a unified framework for computer design innovations. This thinking paradigm will provide a reference model for researchers and practitioners to guide and boost the next wave of innovations in computer systems.

Historical Thinking

Historical thinking is the first point of the regular tetrahedron. Present innovations are based on past innovations. Revolutionary innovations are based on the accumulation of knowledge. Historical thinking explores solutions and methodologies from past results, lessons and experiences.

^a State-of-the-art processors usually have multiple levels of cache on a single chip. For instance, Fujitsu’s 48-core A64FX processor used in Fugaku supercomputer has 32MB L2 cache.⁴⁰

As Grier noted,¹³ many researchers seldom read literature published over five years ago. They only pay attention to the immediate concerns but miss the wealth of important ideas and methodologies accumulated in the field of computing for many years. That limits their vision and often leads to researchers “re-inventing the wheel.” One instance is “The Computer and the Brain,” an unfinished book written by John von Neumann more than 60 years ago, which includes many valuable ideas that today’s researchers could benefit from. Unfortunately, many have not read it.

Historical thinking urges us to reflect on the successful innovations from the past by reading more historical literature in the field, and to learn more from others’ experiences to facilitate our research. This kind of reading and “casual” study will extend our horizons and stimulate our imaginations, further motivating us to think outside the box and start on solid ground.

Historical thinking is a necessity rather than a luxury, and it plays several vital roles in scientific innovation. Armed with a rich historical perspective, computer scientists could frame their work within the broader advancement of human civilization and give them insight into the innovation process. The contributions of computer science research toward the well-being of mankind should be fully recognized and emphasized historically.

Through historical thinking, we can discover new opportunities on the ground of past experiences. The creativity of individual scientists undoubtedly plays an important role in their success. However, the impact and necessity of joint efforts that usually last for a long time and cross geographic lines should not be ignored. History shows that a scientific discovery is generally the result of many scientists’ continuous efforts. Scientists rarely work from scratch; instead, they usually extend and follow up others’ work.

Taking one example of our work, by taking inspiration from Amdahl’s Law in 1967 and Gustafson’s Law in 1988, Sun and Ni developed the memory-bounded speedup model (often referred to as Sun-Ni’s Law) in 1990, which not only extends and unifies Amdahl’s Law and Gustafson’s Law, but also sheds light on the trade-off be-

tween computation and memory for scalable computing.³⁸ These three laws have been incorporated into many textbooks and taken as classic principles in the field of supercomputing. As another example, by extending the well-known memory performance model, AMAT (Average Memory Access Time), we proposed C-AMAT (Concurrent AMAT) in 2014³⁹ to characterize the memory access delay in modern computing systems where data access concurrency is pervasive. Furthermore, by combining Sun-Ni’s law and C-AMAT, a new performance model, C²-bound (the square bound of memory capacity and concurrency), was proposed to show the combined effects of memory capacity and concurrency in 2015.²⁵

Figure 2 depicts a historical progression of memory modeling with respect to changing architectures, from AMAT to C-AMAT, from Amdahl’s Law to Sun-Ni’s Law, and from Moore’s Law to Hill-Marty’s model.¹⁵ These new models are technology driven. For instance, the introduction of CAMAT is due to the prevalence of concurrent data access in modern computing systems. The Hill-Marty’s model is an extension of Amdahl’s Law to multi-core design (please note, though not listed in Figure 2, paired with Hill and Marty, Sun and Chen also extended the scalable computing concept to multicore design³⁷). With the advancement of technologies, each of the models has its own historical significance.

The evolution of technology is not always linear but has lots of “reuses” or “rising from the ashes.” For example, the contemporary Google TPU is the result of a team of history-aware architects.¹⁸ Three important architectural

features could date back to the early 1980s: systolic arrays^{23,24} decoupled-access/execute,³⁵ and Complex Instruction Set Computer (CISC) instructions.³² These features are historical “re-runs.” Some of them did not become the mainstream when they were first introduced, but they are effective and valuable in the TPU’s design.

Based on historical thinking, we can focus on computation and data separately, to develop computational thinking and data-centric thinking, respectively.

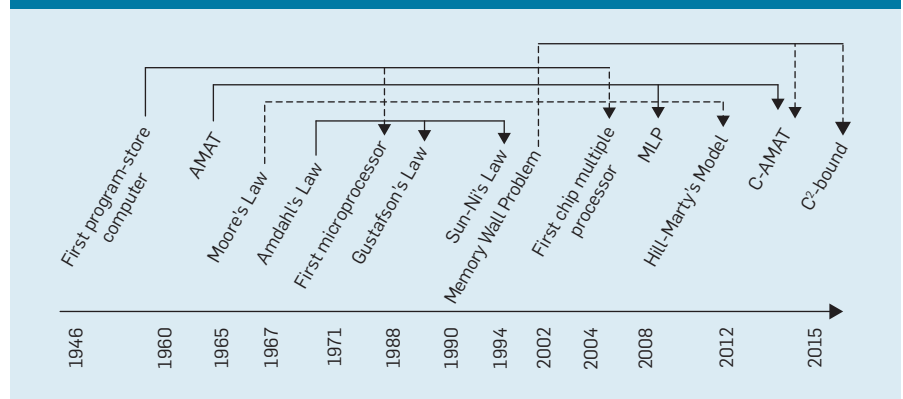
Computational Thinking

Computational thinking (aka computation-centric thinking) is the second point of the regular tetrahedron. Computational thinking has two different meanings from the perspectives of mechanism and behaviorism, respectively. Mechanism focuses on “how” and “performance”, while behaviorism is concerned with “what” and “functionality.” Therefore, computational thinking focuses on making the machine more functional (for example, via AI) and more quickly (for example, via parallel computing).

From a mechanism perspective, computational thinking implies computation-centric design, making the speed of computing components as fast as possible. Pursuing high peak performance of a computer is such an example. On the other hand, from a behaviorism perspective, computational thinking implies solving problems via computation.

In the sense of mechanism, an essential feature of computational thinking is the creation of microarchitectures and the design of computational structures to assist data processing with re-

Figure 2. An instance of historical thinking that shows the machines, laws, and models related to the memory wall problem.



gards to the performance of a problem solver. Time and patience are scarce resources for mankind, so we want to make computers faster and faster, and make the turn-around time of problem-solving as small as possible.

In the sense of behaviorism, the goal of computational thinking is to make the computer more versatile by increasing its utility for an ever-expanding set of applications. Computers are intelligent and powerful machines that can execute different programs for different purposes. For instance, the 2013 Nobel Prize in chemistry was awarded to three scientists due to their contributions to the computational model.³³ As the Nobel committee pointed out, “computer models mirroring real life have become crucial for most advances made in chemistry today, and computers unveil chemical processes, such as a catalyst’s purification of exhaust fumes or the photosynthesis in green leaves.”³⁰ It is simply more difficult and time-consuming to obtain the same results that one would with a simulation via traditional experimental methods, that is, in a wet laboratory.

Computational thinking enables us to understand computation across multiple phases of human history. Computational thinking existed far before the birth of modern computers. For example, humans had a tool to conduct decimal multiplication as early as 305 BC.³² The 2,300-year-old matrix hidden in Chinese bamboo strips was found to be the world’s oldest decimal multiplication table. Before modern computers were invented, a variety of computational tools had already been developed, including ropes, rods and abacuses. Computation is already a crucial part of our daily life, with a profound impact on production and living quality. With computation as a tool, mankind can pursue activities that would be impossible without it. Moreover, with high performance computing, mankind can solve problems that would be prohibitive with traditional computational methods.

Computational thinking also requires us to understand computation in a multidisciplinary manner. The usefulness and broad application of computers benefit from the involvement of domain experts from other fields (for



Both the emergence of big data and the slowdown of Moore’s Law have changed the landscape of computer systems and require us to examine past solutions to pave a new path for future innovations and for addressing new challenges.



example, biology, chemistry and physics), rather than just computer scientists.

Jeannette M. Wing described computational thinking as using abstraction and decomposition when solving real problems and designing large complex systems.⁴⁵ Thinking like a computer scientist means more than being able to program a computer. Instead, it requires thinking at multiple levels of abstraction and examining what computers and automated processes can do. In this way, scientists in diverse fields can extend the functionality of computers to solve more problems.

Computational thinking can greatly improve the level of machine automation and relieve humans of performing tedious computational processes. To illustrate, recall that computers based on the von Neumann architecture take an instruction sequence (program) as a generalized stream of data items, then executes the stream sequentially, guided by the program counter. Computers are powerful and accurate but can only run executable programs. The ability to transform scientific problems into computable mathematical models is essential to computational thinking and must be trained and learned.

Data-Centric Thinking

Data-centric thinking (aka data thinking) is the third point of the regular tetrahedron. Like computational thinking, data-centric thinking also has behavioral and mechanistic meanings in the view of external and internal characteristics of data, respectively. From a behaviorism perspective, data-centric thinking implies solving problems via data. From a mechanism perspective, data-centric thinking implies memory-centric design, making the impact of data access delay as small as possible.

As massive data becomes available, the interests and feasibility of knowledge discovery are also increased. For instance, Google found that if the number of Internet searchers for “flu” suddenly peaked in a region, that region might be experiencing a flu epidemic. Internet search activity can be considered as one form of knowledge discovery, while discovery of the link between search activity and the flu epidemic can be treated as another form.

Data-centric thinking solves prob-

lems by means of collecting and utilizing data. While computational thinking focuses on formulating a problem to make it computationally solvable, data-centric thinking is for gathering and exploiting data to provide insights.^{2,29,24} For instance, suppose we want to provide a set of travel plans for users. Using computational thinking, we might identify the shortest path along a graph (representing cities and routes) with Dijkstra’s algorithm or the Bellman-Ford algorithm. Here, the graph is the abstract mathematical model of the travel planning. Creating this graph mathematical model is a process of computational thinking, as well as the shortest-path algorithms. On the other hand, with data-centric thinking, we no longer need to focus on computation. Instead, we focus on collecting historical route data, analyzing, and exploring the data, and finally making a recommendation to users based on the data. If the recommendation is generated based on deep learning or other AI methods, we often do not know exactly the reasoning behind the recommendations. In other words, data thinking solutions are weak in “causality and interpretability” than mathematical model-based solutions, as stated in Liu et al.,²⁹ while they may be more effective or even the only way to find a solution.

As shown in Figure 3, data-centric thinking is needed to address the challenges of knowledge discovery (aka data mining), which is a daunting task similar to ore smelting, because it is difficult to decide what kinds of data should be collected and how to exploit knowledge from a large amount of data.

The 4V (that is, Volume, Variety, Velocity, and Veracity)^{4,17} characteristics of big data applications have a significant impact on memory localities and have changed the landscape of computing. Big data applications usually have a large quantity of data with poor locality. They are nightmares for the modern memory hierarchy, leading to data starvation that causes processor pipeline stall, known as the memory-wall problem. Local disks and interconnection networks for remote access also can be considered a layer of a more generalized memory hierarchy

The Fugaku supercomputer, currently the world’s fastest computer, only had a peak efficiency of 80.86% when it was first released.⁵ Notably,

for a supercomputer, that is a remarkable achievement, and was the result of a long and labor-intensive performance tuning effort for a particular application (that is, LINPACK). In practice, a high system utilization is very difficult to reach, and the achieved sustained efficiency is often in the single-digit range.

Data access can become a killer performance bottleneck for high performance computing. Simply adding more on-chip memories is not a feasible solution to the memory-wall problem. Deep multi-level cache hierarchy involves more than 80% of the microprocessor on-chip transistors, while studies show that on average these cache blocks are never used during more than 80% of their lifetime.¹⁹ These unutilized cache blocks not only waste power consumption and die spaces, but also increase data searching time. We need to have data-centric thinking toward both the hardware and system designs, to increase system efficiency, and to reduce memory stall time at the same time.

Data-centric thinking for system design involves the whole life cycle of data processing, from data collection, data storage, to data movement and operation. It places the memory system at the highest priority, at least at the same level as the CPU. As early as 1990, we revealed that memory is a major constraint of scalable computing and introduced the memory-bounded speedup model.³⁸ More recently, we established the C-AMAT model to promote and utilize concurrent data access.³⁹ These models are embodiments of data-centric thinking for computer system designs. Big data infrastructures and supercomputers have different design considerations and are with indi-

vidual ecosystems. Since data access is as important as data processing (that is, computing), big data infrastructures and supercomputers will coexist for a long time, and need to support and complement one another, resulting in a converged unified eco-system.

Architectural Thinking

Architectural thinking is the fourth point of the regular tetrahedron. Architectural thinking refers to utilizing existing transistors to build optimized computer systems through hardware and firmware designs. In general, the term “computer system” could mean a hardware system, a software system or a combination of the two. Architectural thinking focuses on hardware systems. The emphasis on utilizing existing transistors implies that we no longer merely call for more transistors but focus squarely on their optimal organization and configuration.

Architectural thinking can be conducted at different levels, the highest of which is the system level, which includes processor, memory, network, and input and output (I/O). For instance, the choice of adopting MCDRAM³⁶ and NVRAM¹¹ has revolutionized the organization of memory systems. This organization and configuration can be extended to the chip level and component level as well. For example, the number and connection of cores, the capacity of on-chip caches, the cache management (insertion, promotion and replacement), and the design of solid-state drives (SSDs) all involve architectural thinking.

Computer architecture is facing many new challenges. First, conventionally, the “best” system is measured in CPU performance; now, as discussed, to optimize CPU performance

Figure 3. Analogy between data mining and ore smelting.

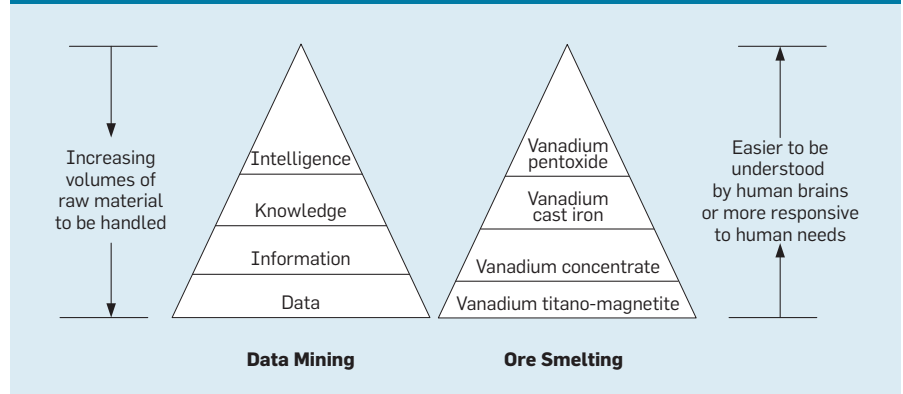
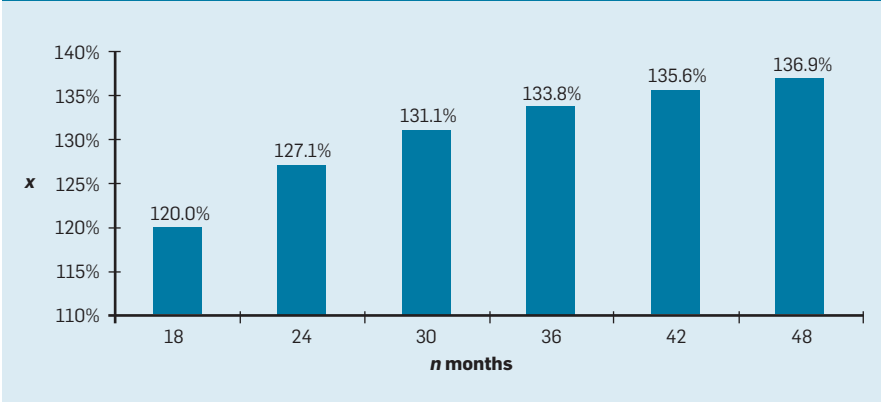


Figure 4. Architectural innovation x needs to be increased with the slowdown of Moore's Law (Assume the number of transistors in a unit area doubles every n month, and the performance improvement that must be contributed by architectural innovation is x).



we first need to consider memory performance in addition to memory power consumption. Second, even for conventional CPU-centric architectural design, we cannot merely rely on Moore's Law to improve CPU performance.

The International Technology Roadmap for Semiconductors (ITRS) has called off the pursuit of Moore's Law and stopped the half-century practice.⁴⁴ Note that the fastest supercomputer, Fugaku, reached 415.5 petaflops of performance in June 2020.⁴⁰ Assume the number of compute nodes in a supercomputer is fixed.^b If we want to achieve exascale (1,024 petaflops) compute speed before June 2021, we have only one year and require a 2.46-fold (1024/415.5) speedup. In other words, on average about 146% performance improvement per year is required.

Realizing 146% per year performance improvement is a daunting challenge. According to Moore's Law, the density of transistors is doubled every 18 months, which amounts to a 58% density increase per year. According to Pollack's Rule,⁵ the performance is proportional to the square root of the number of transistors, and therefore 58% density increase can be restated as 26% performance improvement. As a result, even with the contribution of Moore's Law, we still must have an ad-

ditional 120% performance improvement to fulfill the 146% per year requirement. However, the speed of density increase has slowed down, and the density of transistors has been doubled every three years since 2010, which is only 50% of what Moore's Law predicted.

Assuming the number of transistors in a unit area doubles every n months, and the annual rate of growth is a , we then have

$$(1 + a)^{\frac{12}{n}} = 2$$

Therefore, according to Pollack's rule, the increase in the number of transistors can be converted into performance improvement by b -fold,

$$b = (1 + a)^{\frac{1}{2}} - 1$$

By combining Eq. (1) and Eq. (2), we have

$$b = 2^{\frac{6}{n}} - 1$$

By Eq. (3), as n increases, the performance improvement due to non-architectural innovation, b , is decreasing. Assume the performance improvement that needs to be contributed by architectural innovation is x , where the sum of x and b is expected to be 146% (that is, the total performance improvement is 146%), then by Eq. (3) we can obtain

$$x = 2.46 - 2^{6/n}$$

By Eq. (4), x increases with n , and the relation between x and n is shown in Figure 4. As Moore's Law begins to

slowdown, the requirement for architectural improvement will grow from 120% to 136.9% as n increases from 18 to 36 (see Figure 4). This is a new challenge for architectural design.

Architecture is the hardware skeleton for running algorithms, impacting the efficiency of computer systems and the quality of user experience. For example, an 8-core processor developed by our colleagues has attained a 35% improvement in efficiency compared with its predecessor.¹⁶ The significant performance improvement is mainly attributed to the innovations of its architecture: the input/output structure and the associated bandwidth and latency are improved by upgrading the interconnect and the memory interface.

From the early mainframe computers to the millions of processor cores in present supercomputers, there is an analogy to the "few but giant" and "small but many." By this analogy, as shown in Figure 5, several dimensions need to be considered for the trade-off between the components in a computer system, including uni-core vs. multi-core, reduced vs. complex, shared vs. private, distributed vs. centralized, latency vs. bandwidth, locality vs. concurrency, homogeneous vs. heterogeneous, synchronous vs. asynchronous, general purpose vs. special purpose, and so on. Many works published on top conferences and journals on computer architecture can be mapped onto different positions in the huge design space shown in Figure 5. For instance, domain-specific hardware accelerators⁸ realized by ASICs, FPGAs, or GPUs are for special-purpose computing, conventional CPUs are for general purpose computing, and architecture design choices are laying between the two extremes.

Architectural thinking has several dimensions, each having many choices. We assume an architecture A including n different components ranging from computation, to memory access and communication. For any component p_i in a computer system, there exist multiple (N_i) design choices. According to the combinatorics, all the combinations of choices constitute a huge design space, which is of size $\prod_{i=1}^n N_i$. Even if the design space only has 10 dimensions and each dimension only has 10 different options, there would be 10 billion (10^{10}) possible configurations.

^b This assumption is intended to save resources and maintain system size, but in practice people tend to increase the overall performance of the supercomputer by increasing system size. For instance, as the fastest supercomputer in the top500 list, Fugaku's performance is 2.8 times that of the runner-up (Summit), but the former has 3.02 times more processor cores than the latter.⁴⁰

What makes the issue even more challenging is the effectiveness of architectural design is application dependent. The huge space of applications/workloads must be considered in an architectural design. In addition, there are a series of constraints in terms of energy consumption, temperature, area, as well as cost and performance. An architectural innovation must satisfy these constraints under current technologies, while optimized to serve a variety of applications. However, it is difficult for a single architecture to fit all applications to deliver the highest performance. A feasible way to achieve this goal is to discover an elastic and flexible structure, which can dynamically map application workloads onto the architectural design space. Given the size of the design space, an exhaustive search of architectural and application pairings is extremely difficult, if not impossible.

Discussion on Solutions

Thinking paradigm can guide us to address research issues more effectively. In the following, we give some examples on how our thinking paradigm helped us in research:

We start with historical thinking. First, after investigating and examining a large number of solutions and literature, we find two keywords that can summarize the commonalities of the solutions of memory system optimization. They are “labeling”^{1,21,28} and “matching.”^{26,27}

Then we apply computational thinking and data-centric thinking to reason about the necessity of “labeling” and “matching.”

Memory accesses are the ties between computing components and data components. Without memory accesses, there would be no computer systems but only separated components. While many computer nodes form a network, the inside of each computer node is also a network in which hardware components communicate via network packets (for example, over the NoC or PCIe).

Finally, inspired by the above observations, with architectural thinking, we proposed the labeled von Neumann architecture.^{1,28,46} The architecture has and only needs to have three features Existence, Simplicity, and Utilization referred to as ESU with regards to labeling:

► **Existence.** Each memory or I/O ac-

cess is attached a high-level semantic label (for example, a virtual machine or thread ID) to explicitly convey high-level information to the underlying hardware.

► **Simplicity.** Labels should be as simple as possible to travel across the data path of a machine in which different hardware components can identify labels and fulfill control policies based on labels.

► **Utilization.** Control logic is introduced on the data paths for leveraging labels to achieve specific goals such as quality-of-service and security to increase the utility of the hardware.

For ESU, the first two features that are about the label representation provide the foundation for the third feature to achieve the matching between computation and data. The ESU principle is to efficiently enable a computing system to have three abilities: Distinguishing, Isolation, and Prioritizing, referred to as DIP.⁴⁶ A computing system with DIP can handle the resource contention among concurrent tasks and can eliminate the uncertainty around data access latency to guarantee user experience.

Also, with computational thinking and data-centric thinking, we notice that the symmetry between computation and data requires the matching be-

tween application data access patterns and memory system architectures. To this end, we proposed an architectural optimization methodology—Layered Performance Matching (LPM)^{26,27} to achieve the desired matching.

With architectural thinking, we designed light-weight hardware structures to quantify and detect the data request-reply matching degree and utilized diverse hardware economically to explore data access concurrency and locality until the mismatch is eliminated. As with Jason Cong⁷ in examining the customized computation, we believe this matching is a representative direction of architectural innovations.

Architectural thinking urges us to consider whether to address issues in a codesigned manner. Notice that besides the hardware approach, the LPM optimization also can be achieved in a software approach. Generally, a matching of *A* and *B* can be achieved by reorganizing *A* to match *B*, or reorganizing *B* to match *A*, or simultaneously reorganizing *A* and *B* to agree with each other. Figure 6 depicts these three methods.

In practice, for layered performance matching, if the architecture configuration is *A*, and the application data access pattern is *B*, there will exist three different optimization approaches. The meth-

Figure 5. Dimensions of Architectural Design Trade-off.

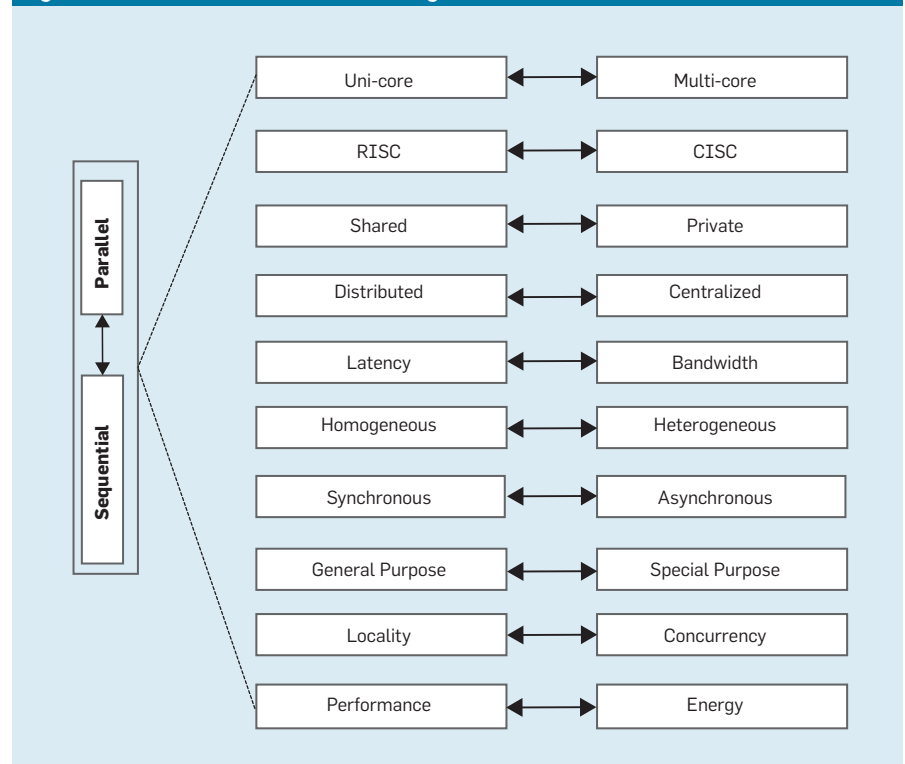


Figure 6. Three different matching methods (A is the “the underlying memory system architecture,” while B denotes “application data access pattern”).

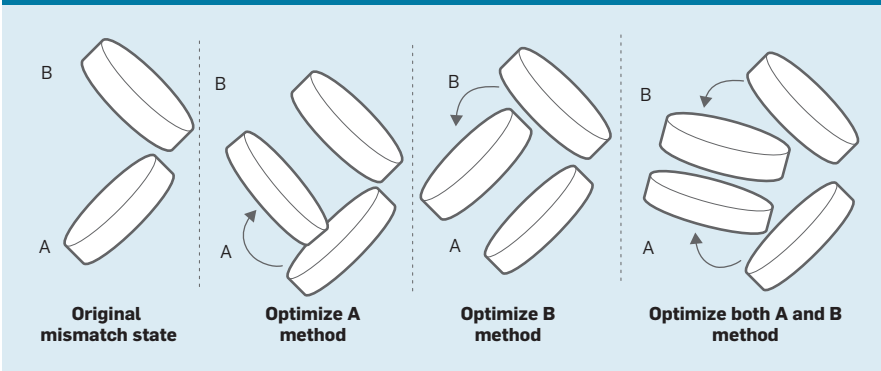
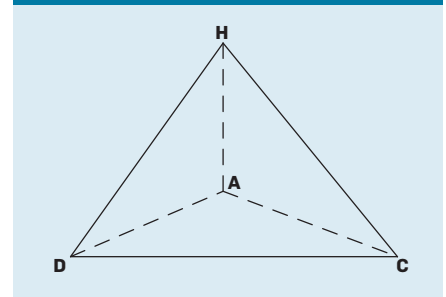


Figure 7. A Framework of the Four Thinking Patterns (“H” represents “Historical thinking,” “A” represents “Architectural thinking,” “D” represents “Data-centric thinking,” and “C” represents “Computational thinking”).



od optimizing *A* is a hardware approach, the method optimizing *B* is a software approach, and the method optimizing both *A* and *B* is a mixed approach. For the hardware approach, the architecture configuration can be adapted online to the data access patterns of applications. On the other hand, for the software approach, the architecture configuration is fixed but with heterogeneity, and through scheduling we also can achieve a better match of the underlying memory systems for a better performance. The scheduling can be implemented in two ways. We can schedule across heterogeneous processors to allocate application data to the underlying hardware according to its data access pattern. Alternatively, we can reorder data accesses at the memory controllers to reshape application data access patterns to adapt to the underlying hardware.

We have illustrated how the generalized thinking paradigm helped us in our research, which includes the labeling and matching designs for computing systems. The formal identification and recognition of the thinking paradigm certainly will boost innovation and productivity.

Discussion on Relations

As shown in Figure 7, the four thinking patterns make up a triangular pyramid, which has six undirected edges (12 directed edges) that correspond to six relationships, that is, H-A (History-Architecture), H-D (History-Data), HC (History-Computation), A-C (Architecture-Computation), A-D (Architecture-Data), and C-D (Computation-Data).

In HCDA, the six undirected edges are six dimensions of the paradigm we can follow. In each dimension, there are two directions, thus the 12 directed

edges are 12 directions. For example, the dimension “H-A” can be separated into “H→A” and “A→H”. “H→A” means using historical knowledge and experience to boost architectural innovation. “A→H” denotes examining and viewing current architectural innovation from a historical perspective. It is not uncommon that a long-existing, warehoused technology resurged from the past history to become modern success. A good example is the deep learning technology. It was first proposed in 1986,¹⁰ and only found its stunning success more than 20 years later.²²

For edge H-A, H-D and H-C, historical thinking provides inspiration and foundation for the other three thinking patterns. We can learn from history to explore solutions following the architectural, data-centric and computational thinking patterns. For instance, Danowitz et al developed a CPU database, which allows designers to mine micro-processor trends over the past 40 years.⁹ In general, we can build databases and conduct corresponding data mining from the perspective of the relationship H-A, H-D and H-C, respectively.

For the connections of A-C and A-D, computational thinking and data-centric thinking can be implemented physically through architectural thinking. The dimensions in Figure 5 can be applied in either compute components or memory components. For example, the dimension of sequential vs. parallel that applies to compute components includes techniques from pipelining, superscalar organization, multithreading, and multi-core. Similarly, all levels of the cache hierarchy also support multiple concurrent data accesses by multiple banks, multiple ports and multiple channels.

For the link of C-D, computational thinking and data-centric thinking should coexist and be conducted simultaneously. When we consider how to solve a problem by computation, we also need to consider how to address the problem via data collection, mining and discovery, in addition to the consideration of data access delay and cost. Computational thinking and data-centric thinking have different focuses. This is one of the reasons why high-performance computing (HPC) and big data currently have different ecosystems and form two partially isolated communities.

For A-C, A-D, and C-D, we need to notice the interactions between compute components and the memory system, which have two distinguishing characteristics. First, regardless of whether the memory system has or does not have computing capabilities, the combined computing-memory system can be seen as a filtering process. A stream of access requests, as seen by lower-level cache, has been filtered by the higher-level caches.

Meanwhile, data access can influence computation in a way with negative feedback. When the memory system can quickly deliver data, the computing component is able to issue more data requests. However, when the number of outstanding memory requests increases beyond the memory system’s handling capacity, queue delays and bus contention will become inevitable, and therefore the memory system can no longer feed data on time. As a result, computing components will slow down (partially or completely stall) due to data starvation, and consequently will reduce data requests.

Due to the combined impact of filtering and feedbacking, CPU performance


and memory performance are entangled and mutually influenced by one another. Therefore, they need to be matched and considered in parallel. High-speed computation requires rapid data movement, and data movement is a critical factor for next-generation architectural design, either from computational thinking or from data-centric thinking perspective. As examples of the HCDA thinking paradigm, the labeled von Neumann architecture and the layered performance matching are general and have their significance in system design.

Conclusion

Modern computers have been developed for more than seventy years. Moore's Law, which has guided the chip design for more than five decades, is approaching its end. The 25-year-old memory wall problem is becoming increasingly problematic. The landscape of computers is changing from computation-centric to data-centric. Literally, the term "computer" cannot convey the full meanings we intend it to. Therefore, to adapt to the changing landscape, we need to explore new ways of thinking and new directions for innovation.

While new computing models such as quantum computing and DNA computing under continued development, this article holds that the conventional von Neumann architecture also can be further developed for a great future. We propose a tetrahedron "historical, computational, data, and architectural" thinking paradigm, referred to as HCDA, to extend the computation thinking in the big data era. Enriching computer scientists with historical thinking, computational thinking, data-centric thinking, and architectural thinking, will boost innovation and provide corresponding actions that enhance the impact of technology on our rapidly evolving society. The four patterns of thinking and their corresponding actions⁴¹ constitute an effective paradigm that can facilitate the advance of computer technologies in the new age of computing.

Acknowledgments. This work is supported in part by the National Natural Science Foundation of China (No. 61772497, 61672513), National Key R&D Program of China (No. 2016YFB1000201), Strategic Priority Research Program of Chinese Academy of Sciences (XDC05030100), YIPA-CAS,

BAAI, PCL, and the U.S. National Science Foundation under NSF CCF-1536079 and CNS-1730488. We thank the anonymous reviewers for their feedback and Kyle Hale for their assistance. The first author thanks Mingfa Zhu for his guidance. 

References

- Bao, Y.-G. and Wang, S. Labeled von Neumann architecture for software-defined cloud. *J. Computer Science and Technology* 32, 2 (2017), 219–223.
- Berman, F. et al. Realizing the potential of data science. *Commun. ACM* 61, 4 (Apr.) 67–72.
- Beveridge, W. *The Art of Scientific Investigation*. Vintage Books, 1954.
- Beyer, M. Gartner says solving big data challenge involves more than just managing volumes of data; <http://www.gartner.com/it/page.jsp?id=1731916>.
- Borkar, S. Thousand core chips: a technology perspective. In *Proceedings of the 54th IEEE Annual Design Automation Conf.* (San Diego, CA), 746–749.
- Carell, T. Molecular computing: DNA as a logic operator. *Nature* 469, 7328, 45–46.
- Cong, J., Wei, P., Yu, C. H., and Zhou, P. Bandwidth optimization through on-chip memory restructuring for HLS. In *Proceedings of the 54th IEEE Annual Design Automation Conf.* (Austin, TX), 1–6.
- Dally, W.J., Turakhia, Y., and Han, S. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (July 2020), 48–57.
- Danowitz, A., Kelley, K., Mao, J., Stevenson, J.P., and Horowitz, M. CPU DB: Recording microprocessor history. *Queue* 10, 4, (2012), 10–27.
- Dechter, R. Learning while searching in constraint-satisfaction problems. In *Proceedings of the 5th National Conf. Artificial Intelligence* (Philadelphia, PA, USA, Aug. 11–15, 1986).
- Fernando, P., Kannan, S., Gavrilovska, A., and Schwan, K. Phoenix: Memory speed HPC I/O with NVM. In *Proceedings IEEE 23rd Intern. Conf. High Performance Computing* (Hyderabad, India, 2017).
- Grier, D.A. How to think from the perspective of computing? *Commun. Chinese Computing Federation* 15, 4, (2019), 36–38.
- Grier, D.A. Old and new: The computer and the brain. *Commun. Chinese Computing Federation* 13, 3 (2017), 67–68.
- Hennessy, J.L. and Patterson, D.A. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2018), 48–60.
- Hill, M.D. and Marty, M.D. Amdahl's law in the multicore era. *Computer* 41, 7 (2008), 33–38.
- Hu, W., Zhang, Y., Yang, L., and Fan, B. Godson-3b1500: A 32nm 1.35ghz 40w 172.8gflops 8-core processor. In *Proceedings of Intern. Solid-State Circuits Conf. IEEE*, 54–55.
- IBM big data hub; <http://www.ibmbigdatahub.com/infographic/fourvs-big-data>.
- Jouppi, N.P. et al. In datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual Intern. Symp. Computer Architecture*, 2017, 1–12.
- Khan, S.M., Tian, Y., and Jimenez, D.A. Sampling dead block prediction for last-level caches. In *Proceedings of the 43rd Annual IEEE/ACM Intern. Symp. on Microarchitecture*. IEEE, 2010, 175–186.
- Knill, E. Physics: Quantum computing. *Nature* 463, 7280 (2010), 441–443.
- Kougkas, A., Devarajan, H., Lofstead, J., and Sun, X.-H. Labios: A distributed label-based I/O system. In *Proceedings in the 28th ACM Intern. Symp. High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA).
- Krizhevsky, A., Sutskever, I., Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25, 2, 1–9.
- Kung, H. Why systolic architectures? *IEEE Computer* 15, 1 (1982), 37–46.
- Kung, H. and Leiserson, C. Algorithms for VLSI Processor Arrays. Introduction to VLSI Systems, 1980.
- Liu, Y. and Sun, X.-H. C2-bound: A Capacity and Concurrency Driven Analytical Model for Many-core Design. In *Proceedings of Intern. Conf. for High Performance Computing, Networking, Storage and Analysis*. IEEE/ACM (Austin, TX, 2015), 1–11.

- Liu, Y. and Sun, X.-H. LPM: A systematic methodology for concurrent data access pattern optimization from a matching perspective. *IEEE Trans. on Parallel and Distributed Systems* 30, 11 (2019), 1–16.
- Liu, Y. and Sun, X.-H. LPM: Concurrency-driven layered performance matching. In *Proceedings of the 44th Intern. Conf. Parallel Processing*. ACM/IEEE (Beijing, China, 2015), 879–888.
- Ma, J. et al. Supporting Differentiated Services in Computers via Programmable Architecture for Resourcing-on-Demand. In *Proceedings of the 20th Intern. Conf. Architectural Support for Programming Languages and Operating Systems* (2015), 131–143.
- Mayer-Schnberger, V. and Cukier, K. *Big Data: A Revolution that Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt Publishing Co., Boston, New York, 2013.
- Norwegian Nobel Committee. Nobel prize in chemistry; <http://www.nobelprize.org/nobel>
- Patterson, D.A. and Ditzel, D.R. The case for the Reduced Instruction Set Compute. *ACM SIGARCH Computer Architecture News* 8, 6, (1980), 25–33.
- Qiu, J. "Ancient Times Table Hidden in Chinese Bamboo Strip. *Nature News*, Jan. 2014.
- Reed, D.A. and Dongarra, J. Exascale computing and big data. *Commun. ACM* 58, 7 (2015), 56–68.
- Reshef, D.N. et al. Detecting novel associations in large data sets. *Science* 334, 6062, (2008), 1518–1524.
- Smith, J. Decoupled access/execute computer architectures. In *Proceedings of the 9th Annual Symp. Computer Architecture* (Austin, TX, 1982), 26–29.
- Smith, S., Park, J., and Karypis, G. Sparse tensor factorization on many-core processors with high-bandwidth memory. In *Proceedings of the 2017 IEEE Intern. of Parallel and Distributed Processing Symp.* IEEE, 2017, 1058–1067.
- Sun, X.-H. and Chen, Y. Reevaluating Amdahl's Law in the multicore era. *J. Parallel and Distributed Computing* 70, 2, (2010), 183–188.
- Sun, X.-H. and Ni, L. Another view on parallel speedup. In *Proceedings of Intern. Conf. for High Performance Computing, Networking, Storage and Analysis*. IEEE/ACM, New York, (1990), 324–333.
- Sun, X.-H. and Wang, D. Concurrent average memory access time. *Computer* 47, 5 (2014), 74–80.
- Supercomputer top 500 list; <https://www.top500.org/lists/2020/06/>
- Tissenbaum, M.J., Sheldon, J., and Abelson, H. From computational thinking to computational action. *Commun. ACM* 62, 3 (Mar. 2019), 34–36.
- Von Neumann, J. *The Computer and the Brain*. Yale University Press, 1958.
- Von Neumann, J. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing* 15, 4, (1993), 27–75.
- Waldrop, M. The chips are down for Moore's Law. *Nature* 530 (Feb. 2016), 144–147.
- Wing, J.M. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.
- Xu, Z. and Li, C. Low-entropy cloud computing systems. *Scientia Sinica Informationis* 47, 9, (2017), 1149–1163.

Yuhang Liu (liuyuhang@ict.ac.cn) is an associate professor at the State Key Laboratory of Computer Architecture, University of Chinese Academy of Sciences, and Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

X.-H. Sun is a distinguished professor with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA.

Yang Wang is a professor at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China.

Yungang Bao is a professor at the State Key Laboratory of Computer Architecture, University of Chinese Academy of Sciences, and Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

© 2021 ACM 0001-0782/21/5



Watch the authors discuss this work in the exclusive Communications video. <https://cacm.acm.org/videos/hcda>