CONTRIBUTED ARTICLES

A New Golden Age for Computer Architecture

By John L. Hennessy, David A. Patterson

Communications of the ACM, February 2019, Vol. 62 No. 2, Pages 48-60
10.1145/3282307
Comments

Credit: Peter Crowther Associates

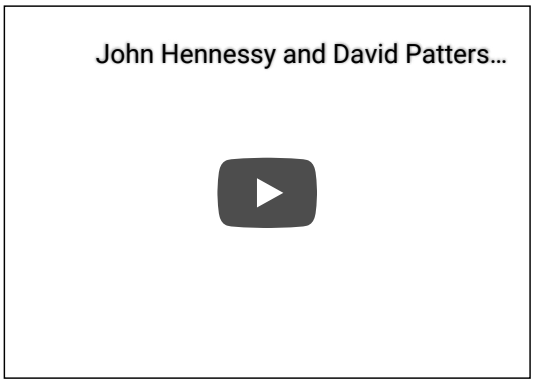
We began our Turing Lecture June 4, 2018[11] with a review of computer architecture since the 1960s. In addition to that review, here, we highlight current challenges and identify future opportunities, projecting another golden age for the field of computer architecture in the next decade, much like the 1980s when we did the research that led to our award, delivering gains in cost, energy, and security, as well as performance.

"Those who cannot remember the past are condemned to repeat it."—George Santayana, 1905

Back to Top

Key Insights

- Software advances can inspire architecture innovation.
- Elevating the hardware/software interface creates opportunities for architecture innovation.
- The marketplace ultimately settles architecture debates.



Software talks to hardware through a vocabulary called an instruction set architecture (ISA). By the early 1960s, IBM had four incompatible lines of computers, each with its own ISA, software stack, I/O system, and market niche—targeting small business, large business, scientific, and real time, respectively. IBM engineers, including ACM A.M. Turing Award laureate Fred Brooks, Jr., thought they could create a single ISA that would efficiently unify all four of these ISA bases.

They needed a technical solution for how computers as inexpensive as those with 8-bit data paths and as fast as those with 64-bit data paths could share a single ISA. The data paths are the "brawn" of the processor in that they perform the arithmetic but are relatively easy to "widen" or "narrow." The greatest challenge for computer designers then and now is the "brains" of the processor—the control hardware. Inspired by software programming, computing pioneer and Turing laureate Maurice Wilkes proposed how to simplify control. Control was specified as a two-dimensional array he called a "control store." Each column of the array corresponded to one control line, each row was a microinstruction, and writing microinstructions was called microprogramming.[39] A control store contains an ISA interpreter written using microinstructions, so execution of a conventional instruction takes several microinstructions. The control store was implemented through memory, which was much less costly than logic gates.

The table here lists four models of the new System/360 ISA IBM announced April 7, 1964. The data paths vary by a factor of 8, memory capacity by a factor of 16, clock rate by nearly 4, performance by 50, and cost by nearly 6. The most expensive computers had the widest control stores because more complicated data paths used more control lines. The least-costly computers had narrower control stores due to simpler hardware but needed more microinstructions since they took more clock cycles to execute a System/360 instruction.

Facilitated by microprogramming, IBM bet the future of the company that the new ISA would revolutionize the computing industry and won the bet. IBM dominated its markets, and IBM mainframe descendants of the

computer family announced 55 years ago still bring in $10 billion in revenue per year.

As seen repeatedly, although the marketplace is an imperfect judge of technological issues, given the close ties between architecture and commercial computers, it eventually determines the success of architecture innovations that often require significant engineering investment.

Integrated circuits, CISC, 432, 8086, IBM PC. When computers began using integrated circuits, Moore's Law meant control stores could become much larger. Larger memories in turn allowed much more complicated ISAs. Consider that the control store of the VAX-11/780 from Digital Equipment Corp. in 1977 was 5,120 words x 96 bits, while its predecessor used only 256 words x 56 bits.

Some manufacturers chose to make microprogramming available by letting select customers add custom features they called "writable control store" (WCS). The most famous WCS computer was the Alto[36] Turing laureates Chuck Thacker and Butler Lampson, together with their colleagues, created for the Xerox Palo Alto Research Center in 1973. It was indeed the first personal computer, sporting the first bit-mapped display and first Ethernet local-area network. The device controllers for the novel display and network were microprograms stored in a 4,096-word x 32-bit WCS.

Microprocessors were still in the 8-bit era in the 1970s (such as the Intel 8080) and programmed primarily in assembly language. Rival designers would add novel instructions to outdo one another, showing their advantages through assembly language examples.

Gordon Moore believed Intel's next ISA would last the lifetime of Intel, so he hired many clever computer science Ph.D.'s and sent them to a new facility in Portland to invent the next great ISA. The 8800, as Intel originally named it, was an ambitious computer architecture project for any era, certainly the most aggressive of the 1980s. It had 32-bit capability-based addressing, object-oriented architecture, variable-bit-length instructions, and its own operating system written in the then-new programming language Ada.

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Control store size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock rate (ROM cycle time)	1.3 MHz (750 ns)	1.6 MHz (625 ns)	2 MHz (500 ns)	5 MHz (200 ns)
Memory capacity	8–64 KiB	16–256 KiB	64–512 KiB	128–1,024 KiB
Performance (commercial)	29,000 IPS	75,000 IPS	169,000 IPS	567,000 IPS
Performance (scientific)	10,200 IPS	40,000 IPS	133,000 IPS	563,000 IPS
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000

Figure. Features of four models of the IBM System/360 family; IPS is instructions per second.

This ambitious project was alas several years late, forcing Intel to start an emergency replacement effort in Santa Clara to deliver a 16-bit microprocessor in 1979. Intel gave the new team 52 weeks to develop the new "8086" ISA and design and build the chip. Given the tight schedule, designing the ISA took only 10 person-weeks over three regular calendar weeks, essentially by extending the 8-bit registers and instruction set of the 8080 to 16 bits. The team completed the 8086 on schedule but to little fanfare when announced.

To Intel's great fortune, IBM was developing a personal computer to compete with the Apple II and needed a 16-bit microprocessor. IBM was interested in the Motorola 68000, which had an ISA similar to the IBM 360, but it was behind IBM's aggressive schedule. IBM switched instead to an 8-bit bus version of the 8086. When IBM announced the PC on August 12, 1981, the hope was to sell 250,000 PCs by 1986. The company instead sold 100 million worldwide, bestowing a very bright future on the emergency replacement Intel ISA.

Intel's original 8800 project was renamed iAPX-432 and finally announced in 1981, but it required several chips and had severe performance problems. It was discontinued in 1986, the year after Intel extended the 16-bit 8086 ISA in the 80386 by expanding its registers from 16 bits to 32 bits. Moore's prediction was thus correct that the next ISA would last as long as Intel did, but the marketplace chose the emergency replacement 8086 rather than the anointed 432. As the architects of the Motorola 68000 and iAPX-432 both learned, the marketplace is rarely patient.

From complex to reduced instruction set computers. The early 1980s saw several investigations into complex instruction set computers (CISC) enabled by the big microprograms in the larger control stores. With Unix demonstrating that even operating systems could use high-level languages, the critical question became: "What instructions would compilers generate?" instead of "What assembly language would programmers use?" Significantly raising the hardware/software interface created an opportunity for architecture innovation.

Turing laureate John Cocke and his colleagues developed simpler ISAs and compilers for minicomputers. As an experiment, they retargeted their research compilers to use only the simple register-register operations and load-store data transfers of the IBM 360 ISA, avoiding the more complicated instructions. They found that programs ran up to three times faster using the simple subset. Emer and Clark[6] found 20% of the VAX instructions needed 60% of the microcode and represented only 0.2% of the execution time. One author (Patterson) spent a sabbatical at DEC to help reduce bugs in VAX microcode. If microprocessor manufacturers were going to follow the CISC ISA designs of the larger computers, he thought they would need a way to repair the microcode bugs. He wrote such a paper,[31] but the journal *Computer* rejected it. Reviewers opined that it was a terrible idea to build microprocessors with ISAs so complicated that they needed to be repaired in the field. That rejection called into question the value of CISC ISAs for microprocessors. Ironically, modern CISC microprocessors do indeed include microcode repair mechanisms, but the main result of his

paper rejection was to inspire him to work on less-complex ISAs for microprocessors—reduced instruction set computers (RISC).

These observations and the shift to high-level languages led to the opportunity to switch from CISC to RISC. First, the RISC instructions were simplified so there was no need for a microcoded interpreter. The RISC instructions were typically as simple as microinstructions and could be executed directly by the hardware. Second, the fast memory, formerly used for the microcode interpreter of a CISC ISA, was repurposed to be a cache of RISC instructions. (A cache is a small, fast memory that buffers recently executed instructions, as such instructions are likely to be reused soon.) Third, register allocators based on Gregory Chaitin's graph-coloring scheme made it much easier for compilers to efficiently use registers, which benefited these register-register ISAs.[3] Finally, Moore's Law meant there were enough transistors in the 1980s to include a full 32-bit datapath, along with instruction and data caches, in a single chip.

In today's post-PC era, x86 shipments have fallen almost 10% per year since the peak in 2011, while chips with RISC processors have skyrocketed to 20 billion.

For example, Figure 1 shows the RISC-I[8] and MIPS[12] microprocessors developed at the University of California, Berkeley, and Stanford University in 1982 and 1983, respectively, that demonstrated the benefits of RISC. These chips were eventually presented at the leading circuit conference, the IEEE International Solid-State Circuits Conference, in 1984.[33,35] It was a remarkable moment when a few graduate students at Berkeley and Stanford could build microprocessors that were arguably superior to what industry could build.

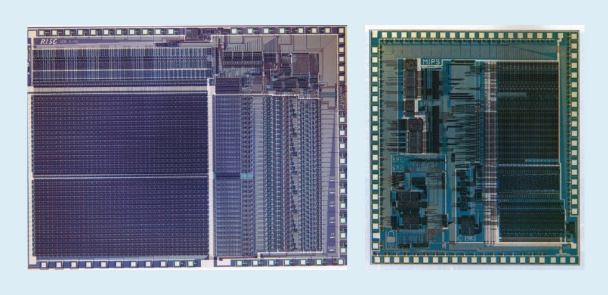


Figure 1. University of California, Berkeley, RISC-I and Stanford University MIPS microprocessors.

These academic chips inspired many companies to build RISC microprocessors, which were the fastest for the next 15 years. The explanation is due to the following formula for processor performance:

Time/Program = Instructions / Program x (Clock cycles) / Instruction x Time / (Clock cycle)

DEC engineers later showed[2] that the more complicated CISC ISA executed about 75% of the number instructions per program as RISC (the first term), but in a similar technology CISC executed about five to six more clock cycles per instruction (the second term), making RISC microprocessors approximately 4x faster.

Such formulas were not part of computer architecture books in the 1980s, leading us to write *Computer Architecture: A Quantitative Approach*[13] in 1989. The subtitle suggested the theme of the book: Use measurements and benchmarks to evaluate trade-offs quantitatively instead of relying more on the architect's intuition and experience, as in the past. The quantitative approach we used was also inspired by what Turing laureate Donald Knuth's book had done for algorithms.[20]

VLIW, EPIC, Itanium. The next ISA innovation was supposed to succeed both RISC and CISC. Very long instruction word (VLIW)[7] and its cousin, the explicitly parallel instruction computer (EPIC), the name Intel and Hewlett Packard gave to the approach, used wide instructions with multiple independent operations bundled together in each instruction. VLIW and EPIC advocates at the time believed if a single instruction could specify, say, six independent operations—two data transfers, two integer operations, and two floating point operations—and compiler technology could efficiently assign operations into the six instruction slots, the hardware could be made simpler. Like the RISC approach, VLIW and EPIC shifted work from the hardware to the compiler.

Working together, Intel and Hewlett Packard designed a 64-bit processor based on EPIC ideas to replace the 32-bit x86. High expectations were set for the first EPIC processor, called Itanium by Intel and Hewlett Packard, but the reality did not match its developers' early claims. Although the EPIC approach worked well for highly structured floating-point programs, it struggled to achieve high performance for integer programs that had less predictable cache misses or less-predictable branches. As Donald Knuth later noted:[21] "The Itanium approach ... was supposed to be so terrific—until it turned out that the wished-for compilers were basically impossible to write." Pundits noted delays and underperformance of Itanium and re-christened it

"Itanic" after the ill-fated Titantic passenger ship. The marketplace again eventually ran out of patience, leading to a 64-bit version of the x86 as the successor to the 32-bit x86, and not Itanium.

The good news is VLIW still matches narrower applications with small programs and simpler branches and omit caches, including digital-signal processing.

Back to Top

RISC vs. CISC in the PC and Post-PC Eras

AMD and Intel used 500-person design teams and superior semiconductor technology to close the performance gap between x86 and RISC. Again inspired by the performance advantages of pipelining simple vs. complex instructions, the instruction decoder translated the complex x86 instructions into internal RISC-like microinstructions on the fly. AMD and Intel then pipelined the execution of the RISC microinstructions. Any ideas RISC designers were using for performance—separate instruction and data caches, second-level caches on chip, deep pipelines, and fetching and executing several instructions simultaneously—could then be incorporated into the x86. AMD and Intel shipped roughly 350 million x86 microprocessors annually at the peak of the PC era in 2011. The high volumes and low margins of the PC industry also meant lower prices than RISC computers.

Given the hundreds of millions of PCs sold worldwide each year, PC software became a giant market. Whereas software providers for the Unix marketplace would offer different software versions for the different commercial RISC ISAs—Alpha, HP-PA, MIPS, Power, and SPARC—the PC market enjoyed a single ISA, so software developers shipped "shrink wrap" software that was binary compatible with only the x86 ISA. A much larger software base, similar performance, and lower prices led the x86 to dominate both desktop computers and small-server markets by 2000.

Apple helped launch the post-PC era with the iPhone in 2007. Instead of buying microprocessors, smartphone companies built their own systems on a chip (SoC) using designs from other companies, including RISC processors from ARM. Mobile-device designers valued die area and energy efficiency as much as performance, disadvantaging CISC ISAs. Moreover, arrival of the Internet of Things vastly increased both the number of processors and the required trade-offs in die size, power, cost, and performance. This trend increased the importance of design time and cost, further disadvantaging CISC processors. In today's post-PC era, x86 shipments have fallen almost 10% per year since the peak in 2011, while chips with RISC processors have skyrocketed to 20 billion. Today, 99% of 32-bit and 64-bit processors are RISC.

Concluding this historical review, we can say the marketplace settled the RISC-CISC debate; CISC won the later stages of the PC era, but RISC is winning the post-PC era. There have been no new CISC ISAs in decades. To our surprise, the consensus on the best ISA principles for general-purpose processors today is still RISC, 35 years after their introduction.

Back to Top

Current Challenges for Processor Architecture

"If a problem has no solution, it may not be a problem, but a fact—not to be solved, but to be coped with over time."—Shimon Peres

While the previous section focused on the design of the instruction set architecture (ISA), most computer architects do not design new ISAs but implement existing ISAs in the prevailing implementation technology. Since the late 1970s, the technology of choice has been metal oxide semiconductor (MOS)-based integrated circuits, first n-type metal-oxide semiconductor (nMOS) and then complementary metal-oxide semiconductor (CMOS). The stunning rate of improvement in MOS technology—captured in Gordon Moore's predictions—has been the driving factor enabling architects to design more-aggressive methods for achieving performance for a given ISA. Moore's original prediction in 1965[26] called for a doubling in transistor density yearly; in 1975, he revised it, projecting a doubling every two years.[28] It eventually became called Moore's Law. Because transistor density grows quadratically while speed grows linearly, architects used more transistors to improve performance.

Back to Top

End of Moore's Law and Dennard Scaling

Although Moore's Law held for many decades (see Figure 2), it began to slow sometime around 2000 and by 2018 showed a roughly 15-fold gap between Moore's prediction and current capability, an observation Moore made in 2003 that was inevitable.[27] The current expectation is that the gap will continue to grow as CMOS technology approaches fundamental limits.

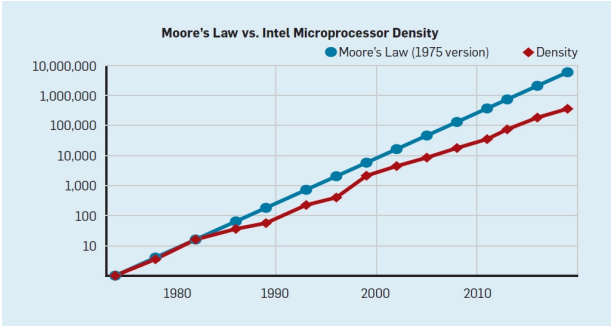


Figure 2. Transistors per chip of Intel microprocessors vs. Moore's Law.

Accompanying Moore's Law was a projection made by Robert Dennard called "Dennard scaling,"[5] stating that as transistor density increased, power consumption per transistor would drop, so the power per mm^2 of silicon would be near constant. Since the computational capability of a mm^2 of silicon was increasing with each new generation of technology, computers would become more energy efficient. Dennard scaling began to slow significantly in 2007 and faded to almost nothing by 2012 (see Figure 3).

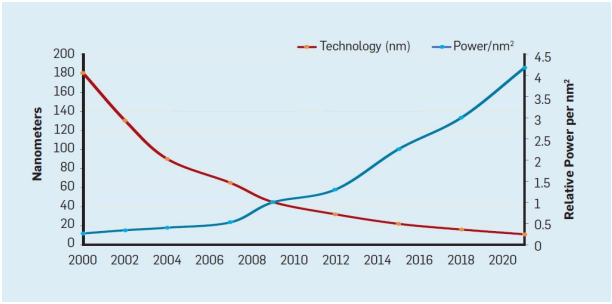


Figure 3. Transistors per chip and power per mm^2.

Between 1986 and about 2002, the exploitation of instruction level parallelism (ILP) was the primary architectural method for gaining performance and, along with improvements in speed of transistors, led to an annual performance increase of approximately 50%. The end of Dennard scaling meant architects had to find more efficient ways to exploit parallelism.

To understand why increasing ILP caused greater inefficiency, consider a modern processor core like those from ARM, Intel, and AMD. Assume it has a 15-stage pipeline and can issue four instructions every clock cycle. It thus has up to 60 instructions in the pipeline at any moment in time, including approximately 15 branches, as they represent approximately 25% of executed instructions. To keep the pipeline full, branches are predicted and code is speculatively placed into the pipeline for execution. The use of speculation is both the source of ILP performance and of inefficiency. When branch prediction is perfect, speculation improves performance yet involves little added energy cost—it can even save energy—but when it "mispredicts" branches, the processor must throw away the incorrectly speculated instructions, and their computational work and energy are wasted. The internal state of the processor must also be restored to the state that existed before the mispredicted branch, expending additional time and energy.

To see how challenging such a design is, consider the difficulty of correctly predicting the outcome of 15 branches. If a processor architect wants to limit wasted work to only 10% of the time, the processor must predict each branch correctly 99.3% of the time. Few general-purpose programs have branches that can be predicted so accurately.

To appreciate how this wasted work adds up, consider the data in Figure 4, showing the fraction of instructions that are effectively executed but turn out to be wasted because the processor speculated incorrectly. On average, 19% of the instructions are wasted for these benchmarks on an Intel Core i7. The amount of wasted energy is greater, however, since the processor must use additional energy to restore the state when it speculates incorrectly. Measurements like these led many to conclude architects needed a different approach to achieve performance improvements. The multicore era was thus born.

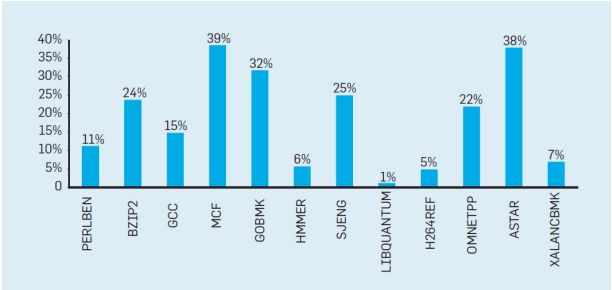


Figure 4. Wasted instructions as a percentage of all instructions completed on an Intel Core i7 for a variety of SPEC integer benchmarks.

Multicore shifted responsibility for identifying parallelism and deciding how to exploit it to the programmer and to the language system. Multicore does not resolve the challenge of energy-efficient computation that was exacerbated by the end of Dennard scaling. Each active core burns power whether or not it contributes effectively to the computation. A primary hurdle is an old observation, called Amdahl's Law, stating that the speedup from a parallel computer is limited by the portion of a computation that is sequential. To appreciate the importance of this observation, consider Figure 5, showing how much faster an application runs with up to 64 cores compared to a single core, assuming different portions of serial execution, where only one processor is active. For example, when only 1% of the time is serial, the speedup for a 64-processor configuration is about 35. Unfortunately, the power needed is proportional to 64 processors, so approximately 45% of the energy is wasted.

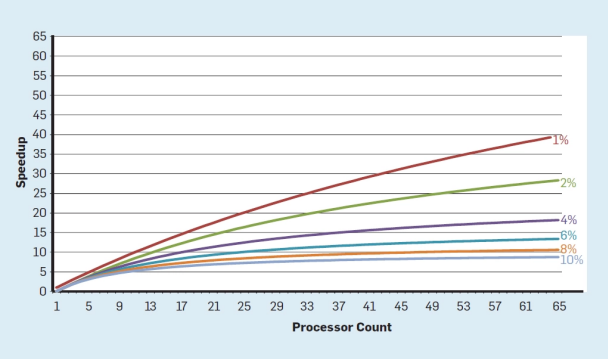


Figure 5. Effect of Amdahl's Law on speedup as a fraction of clock cycle time in serial mode.

Real programs have more complex structures of course, with portions that allow varying numbers of processors to be used at any given moment in time. Nonetheless, the need to communicate and synchronize periodically means most applications have some portions that can effectively use only a fraction of the processors. Although Amdahl's Law is more than 50 years old, it remains a difficult hurdle.

With the end of Dennard scaling, increasing the number of cores on a chip meant power is also increasing at nearly the same rate. Unfortunately, the power that goes into a processor must also be removed as heat. Multicore processors are thus limited by the thermal dissipation power (TDP), or average amount of power the package and cooling system can remove. Although some high-end data centers may use more advanced packages and cooling technology, no computer users would want to put a small heat exchanger on their desks or wear a radiator on their backs to cool their cellphones. The limit of TDP led directly to the era of "dark silicon," whereby processors would slow on the clock rate and turn off idle cores to prevent overheating. Another way to view this approach is that some chips can reallocate their precious power from the idle cores to the active ones.

An era without Dennard scaling, along with reduced Moore's Law and Amdahl's Law in full effect means inefficiency limits improvement in performance to only a few percent per year (see Figure 6). Achieving higher rates of performance improvement—as was seen in the 1980s and 1990s—will require new architectural approaches that use the integrated-circuit capability much more efficiently. We will return to what approaches might work after discussing another major shortcoming of modern computers—their support, or lack thereof, for computer security.

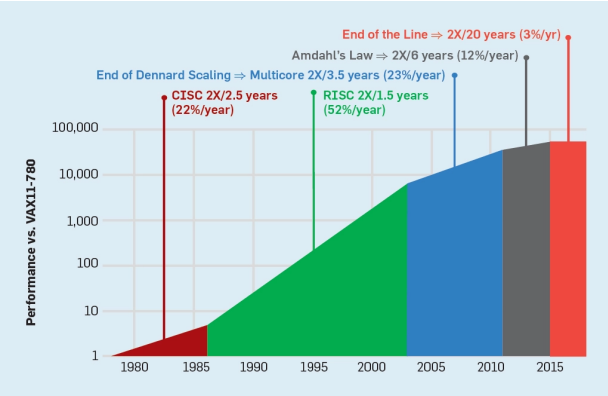


Figure 6. Growth of computer performance using integer programs (SPECintCPU).

Back to Top

Overlooked Security

In the 1970s, processor architects focused significant attention on enhancing computer security with concepts ranging from protection rings to capabilities. It was well understood by these architects that most bugs would be in software, but they believed architectural support could help. These features were largely unused by operating systems that were deliberately focused on supposedly benign environments (such as personal computers), and the features involved significant overhead then, so were eliminated. In the software community, many thought formal verification and techniques like microkernels would provide effective mechanisms for building highly secure software. Unfortunately, the scale of our collective software systems

and the drive for performance meant such techniques could not keep up with processor performance. The result is large software systems continue to have many security flaws, with the effect amplified due to the vast and increasing amount of personal information online and the use of cloud-based computing, which shares physical hardware among potential adversaries.

The end of Dennard scaling meant architects had to find more efficient ways to exploit parallelism.

Although computer architects and others were perhaps slow to realize the growing importance of security, they began to include hardware support for virtual machines and encryption. Unfortunately, speculation introduced an unknown but significant security flaw into many processors. In particular, the Meltdown and Spectre security flaws led to new vulnerabilities that exploit vulnerabilities in the microarchitecture, allowing leakage of protected information at a high rate.[14] Both Meltdown and Spectre use so-called side-channel attacks whereby information is leaked by observing the time taken for a task and converting information invisible at the ISA level into a timing visible attribute. In 2018, researchers showed how to exploit one of the Spectre variants to leak information over a network without the attacker loading code onto the target processor.[34] Although this attack, called NetSpectre, leaks information slowly, the fact that it allows any machine on the same local-area network (or within the same cluster in a cloud) to be attacked creates many new vulnerabilities. Two more vulnerabilities in the virtual-machine architecture were subsequently reported.[37,38] One of them, called Foreshadow, allows penetration of the Intel SGX security mechanisms designed to protect the highest risk data (such as encryption keys). New vulnerabilities are being discovered monthly.

Side-channel attacks are not new, but in most earlier cases, a software flaw allowed the attack to succeed. In the Meltdown, Spectre, and other attacks, it is a flaw in the hardware implementation that exposes protected information. There is a fundamental difficulty in how processor architects define what is a correct implementation of an ISA because the standard definition says nothing about the performance effects of executing an instruction sequence, only about the ISA-visible architectural state of the execution. Architects need to rethink their definition of a correct implementation of an ISA to prevent such security flaws. At the same time, they should be rethinking the attention they pay computer security and how architects can work with software designers to implement more-secure systems. Architects (and everyone else) depend too much on more information systems to willingly allow security to be treated as anything less than a first-class design concern.

Back to Top

Future Opportunities in Computer Architecture

"What we have before us are some breathtaking opportunities disguised as insoluble problems." —John Gardner, 1965

Inherent inefficiencies in general-purpose processors, whether from ILP techniques or multicore, combined with the end of Dennard scaling and Moore's Law, make it highly unlikely, in our view, that processor architects and designers can sustain significant rates of performance improvements in general-purpose processors. Given the importance of improving performance to enable new software capabilities, we must ask: What other approaches might be promising?

There are two clear opportunities, as well as a third created by combining the two. First, existing techniques for building software make extensive use of high-level languages with dynamic typing and storage management. Unfortunately, such languages are typically interpreted and execute very inefficiently. Leiserson et al.[24] used a small example—performing matrix multiply—to illustrate this inefficiency. As in Figure 7, simply rewriting the code in C from Python—a typical high-level, dynamically typed language—increases performance 47-fold. Using parallel loops running on many cores yields a factor of approximately 7. Optimizing the memory layout to exploit caches yields a factor of 20, and a final factor of 9 comes from using the hardware extensions for doing single instruction multiple data (SIMD) parallelism operations that are able to perform 16 32-bit operations per instruction. All told, the final, highly optimized version runs more than 62,000x faster on a multicore Intel processor compared to the original Python version. This is of course a small example, one might expect programmers to use an optimized library for. Although it exaggerates the usual performance gap, there are likely many programs for which factors of 100 to 1,000 could be achieved.

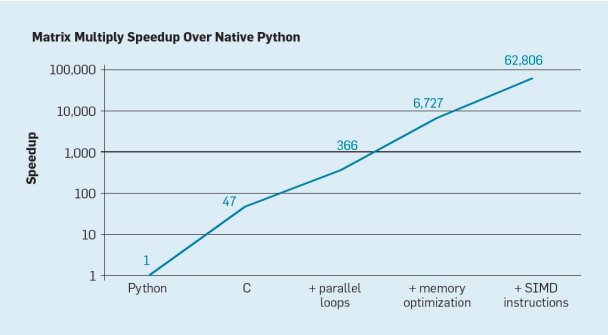


Figure 7. Potential speedup of matrix multiply in Python for four optimizations.

An interesting research direction concerns whether some of the performance gap can be closed with new compiler technology, possibly assisted by architectural enhancements. Although the challenges in efficiently translating and implementing high-level scripting languages like Python are difficult, the potential gain is enormous. Achieving even 25% of the potential gain could result in Python programs running tens to hundreds of times faster. This simple example illustrates how great the gap is between modern languages emphasizing programmer productivity and traditional approaches emphasizing performance.

Domain-specific architectures. A more hardware-centric approach is to design architectures tailored to a specific problem domain and offer significant performance (and efficiency) gains for that domain, hence, the name "domain-specific architectures" (DSAs), a class of processors tailored for a specific domain—programmable and often Turing-complete but tailored to a specific class of applications. In this sense, they differ from application-specific integrated circuits (ASICs) that are often used for a single function with code that rarely changes. DSAs are often called accelerators, since they accelerate some of an application when compared to executing the entire application on a general-purpose CPU. Moreover, DSAs can achieve better performance because they are more closely tailored to the needs of the application; examples of DSAs include graphics processing units (GPUs), neural network processors used for deep learning, and processors for software-defined networks (SDNs). DSAs can achieve higher performance and greater energy efficiency for four main reasons:

First and most important, DSAs exploit a more efficient form of parallelism for the specific domain. For example, single-instruction multiple data parallelism (SIMD), is more efficient than multiple instruction multiple data (MIMD) because it needs to fetch only one instruction stream and processing units operate in lockstep.[9] Although SIMD is less flexible than MIMD, it is a good match for many DSAs. DSAs may also use VLIW approaches to ILP rather than speculative out-of-order mechanisms. As mentioned earlier, VLIW processors are a poor match for general-purpose code[15] but for limited domains can be much more efficient, since the control mechanisms are simpler. In particular, most high-end general-purpose processors are out-of-order superscalars that require complex control logic for both instruction initiation and instruction completion. In contrast, VLIWs perform the necessary analysis and scheduling at compile-time, which can work well for an explicitly parallel program.

Second, DSAs can make more effective use of the memory hierarchy. Memory accesses have become much more costly than arithmetic computations, as noted by Horowitz.[16] For example, accessing a block in a 32-kilobyte cache involves an energy cost approximately 200x higher than a 32-bit integer add. This enormous differential makes optimizing memory accesses critical to achieving high-energy efficiency. General-purpose processors run code in which memory accesses typically exhibit spatial and temporal locality but are otherwise not very predictable at compile time. CPUs thus use multilevel caches to increase bandwidth and hide the latency in relatively slow, off-chip DRAMs. These multilevel caches often consume approximately half the energy of the processor but avoid almost all accesses to the off-chip DRAMs that require approximately 10x the energy of a last-level cache access.

Caches have two notable disadvantages:

When datasets are very large. Caches simply do not work well when datasets are very large and also have low temporal or spatial locality; and

When caches work well. When caches work well, the locality is very high, meaning, by definition, most of the cache is idle most of the time.

In applications where the memory-access patterns are well defined and discoverable at compile time, which is true of typical DSLs, programmers and compilers can optimize the use of the memory better than can dynamically allocated caches. DSAs thus usually use a hierarchy of memories with movement controlled explicitly by the software, similar to how vector processors operate. For suitable applications, user-controlled memories can use much less energy than caches.

Third, DSAs can use less precision when it is adequate. General-purpose CPUs usually support 32- and 64-bit integer and floating-point (FP) data. For many applications in machine learning and graphics, this is more accuracy than is needed. For example, in deep neural networks (DNNs), inference regularly uses 4-, 8-, or 16-bit integers, improving both data and computational throughput. Likewise, for DNN training applications, FP is useful, but 32 bits is enough and 16 bits often works.

Finally, DSAs benefit from targeting programs written in domain-specific languages (DSLs) that expose more parallelism, improve the structure and representation of memory access, and make it easier to map the

application efficiently to a domain-specific processor.

Back to Top

Domain-Specific Languages

DSAs require targeting of high-level operations to the architecture, but trying to extract such structure and information from a general-purpose language like Python, Java, C, or Fortran is simply too difficult. Domain specific languages (DSLs) enable this process and make it possible to program DSAs efficiently. For example, DSLs can make vector, dense matrix, and sparse matrix operations explicit, enabling the DSL compiler to map the operations to the processor efficiently. Examples of DSLs include Matlab, a language for operating on matrices, TensorFlow, a dataflow language used for programming DNNs, P4, a language for programming SDNs, and Halide, a language for image processing specifying high-level transformations.

The challenge when using DSLs is how to retain enough architecture independence that software written in a DSL can be ported to different architectures while also achieving high efficiency in mapping the software to the underlying DSA. For example, the XLA system translates Tensorflow to heterogeneous processors that use Nvidia GPUs or Tensor Processor Units (TPUs).[40] Balancing portability among DSAs along with efficiency is an interesting research challenge for language designers, compiler creators, and DSA architects.

Example DSA TPU v1. As an example DSA, consider the Google TPU v1, which was designed to accelerate neural net inference.[17,18] The TPU has been in production since 2015 and powers applications ranging from search queries to language translation to image recognition to AlphaGo and AlphaZero, the DeepMind programs for playing Go and Chess. The goal was to improve the performance and energy efficiency of deep neural net inference by a factor of 10.

As shown in Figure 8, the TPU organization is radically different from a general-purpose processor. The main computational unit is a matrix unit, a systolic array[22] structure that provides 256 x 256 multiply-accumulates every clock cycle. The combination of 8-bit precision, highly efficient systolic structure, SIMD control, and dedication of significant chip area to this function means the number of multiply-accumulates per clock cycle is approximately 100x what a general-purpose single-core CPU can sustain. Rather than caches, the TPU uses a local memory of 24 megabytes, approximately double a 2015 general-purpose CPU with the same power dissipation. Finally, both the activation memory and the weight memory (including a FIFO structure that holds weights) are linked through user-controlled high-bandwidth memory channels. Using a weighted arithmetic mean based on six common inference problems in Google data centers, the TPU is 29x faster than a general-purpose CPU. Since the TPU requires less than half the power, it has an energy efficiency for this workload that is more than 80x better than a general-purpose CPU.

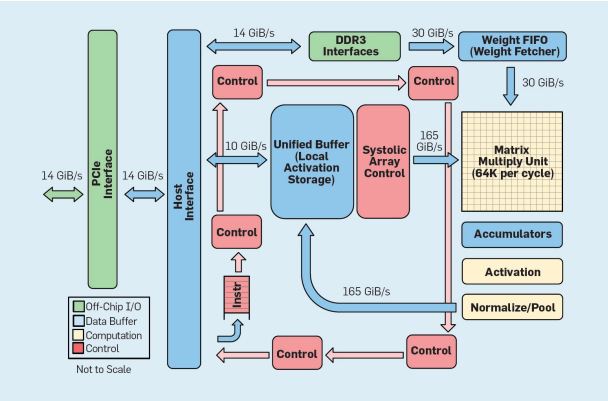


Figure 8. Functional organization of Google Tensor Processing Unit (TPU v1).

Back to Top

Summary

We have considered two different approaches to improve program performance by improving efficiency in the use of hardware technology: First, by improving the performance of modern high-level languages that are typically interpreted; and second, by building domain-specific architectures that greatly improve performance and efficiency compared to general-purpose CPUs. DSLs are another example of how to improve the hardware/software interface that enables architecture innovations like DSAs. Achieving significant gains through such approaches will require a vertically integrated design team that understands applications, domain-specific languages and related compiler technology, computer architecture and organization, and the underlying implementation technology. The need to vertically integrate and make design decisions across levels of abstraction was characteristic of much of the early work in computing before the industry became horizontally structured. In this new era, vertical integration has become more important, and teams that can examine and make complex trade-offs and optimizations will be advantaged.

This opportunity has already led to a surge of architecture innovation, attracting many competing architectural philosophies:

GPUs. Nvidia GPUs use many cores, each with large register files, many hardware threads, and caches;[4]

TPUs. Google TPUs rely on large two-dimensional systolic multipliers and software-controlled on-chip memories;[17]

FPGAs. Microsoft deploys field programmable gate arrays (FPGAs) in its data centers it tailors to neural network applications;[10] and

CPUs. Intel offers CPUs with many cores enhanced by large multi-level caches and one-dimensional SIMD instructions, the kind of FPGAs used by Microsoft, and a new neural network processor that is closer to a TPU than to a CPU.[19]

In addition to these large players, dozens of startups are pursuing their own proposals.[25] To meet growing demand, architects are interconnecting hundreds to thousands of such chips to form neural-network supercomputers.

This avalanche of DNN architectures makes for interesting times in computer architecture. It is difficult to predict in 2019 which (or even if any) of these many directions will win, but the marketplace will surely settle the competition just as it settled the architectural debates of the past.

Back to Top

Open Architectures

Inspired by the success of open source software, the second opportunity in computer architecture is open ISAs. To create a "Linux for processors" the field needs industry-standard open ISAs so the community can create open source cores, in addition to individual companies owning proprietary ones. If many organizations design processors using the same ISA, the greater competition may drive even quicker innovation. The goal is to provide processors for chips that cost from a few cents to $100.

The first example is RISC-V (called "RISC Five"), the fifth RISC architecture developed at the University of California, Berkeley.[32] RISC-V's has a community that maintains the architecture under the stewardship of the RISC-V Foundation (http://riscv.org/). Being open allows the ISA evolution to occur in public, with hardware and software experts collaborating before decisions are finalized. An added benefit of an open foundation is the ISA is unlikely to expand primarily for marketing reasons, sometimes the only explanation for extensions of proprietary instruction sets.

RISC-V is a modular instruction set. A small base of instructions run the full open source software stack, followed by optional standard extensions designers can include or omit depending on their needs. This base includes 32-bit address and 64-bit address versions. RISC-V can grow only through optional extensions; the software stack still runs fine even if architects do not embrace new extensions. Proprietary architectures generally require upward binary compatibility, meaning when a processor company adds new feature, all future processors must also include it. Not so for RISC-V, whereby all enhancements are optional and can be deleted if not needed by an application. Here are the standard extensions so far, using initials that stand for their full names:

M. Integer multiply/divide;

A. Atomic memory operations;

F/D. Single/double-precision floating-point; and

C. Compressed instructions.

A third distinguishing feature of RISC-V is the simplicity of the ISA. While not readily quantifiable, here are two comparisons to the ARMv8 architecture, as developed by the ARM company contemporaneously:

Fewer instructions. RISC-V has many fewer instructions. There are 50 in the base that are surprisingly similar in number and nature to the original RISC-I.[30] The remaining standard extensions—M, A, F, and D—add 53 instructions, plus C added another 34, totaling 137. ARMv8 has more than 500; and

Fewer instruction formats. RISC-V has many fewer instruction formats, six, while ARMv8 has at least 14.

Simplicity reduces the effort to both design processors and verify hardware correctness. As the RISC-V targets range from data-center chips to IoT devices, design verification can be a significant part of the cost of development.

Fourth, RISC-V is a clean-slate design, starting 25 years later, letting its architects learn from mistakes of its predecessors. Unlike first-generation RISC architectures, it avoids microarchitecture or technology-dependent features (such as delayed branches and delayed loads) or innovations (such as register windows) that were superseded by advances in compiler technology.

Finally, RISC-V supports DSAs by reserving a vast opcode space for custom accelerators.

Security experts do not believe in security through obscurity, so open implementations are attractive, and open implementations require an open architecture.

Beyond RISC-V, Nvidia also announced (in 2017) a free and open architecture[29] it calls Nvidia Deep Learning Accelerator (NVDLA), a scalable, configurable DSA for machine-learning inference. Configuration options include data type (int8, int16, or fp16) and the size of the two-dimensional multiply matrix. Die size scales from 0.5 mm^2 to 3 mm^2 and power from 20 milliWatts to 300 milliWatts. The ISA, software stack, and implementation are all open.

Open simple architectures are synergistic with security. First, security experts do not believe in security through obscurity, so open implementations are attractive, and open implementations require an open architecture. Equally important is increasing the number of people and organizations who can innovate around secure architectures. Proprietary architectures limit participation to employees, but open architectures allow all the best minds in academia and industry to help with security. Finally, the simplicity of RISC-V makes its implementations easier to check. Moreover, the open architectures, implementations, and software stacks, plus the plasticity of FPGAs, mean architects can deploy and evaluate novel solutions online and iterate them weekly instead of annually. While FPGAs are 10x slower than custom chips, such performance is still fast enough to support online users and thus subject security innovations to real attackers. We expect open architectures to become the exemplar for hardware/software co-design by architects and security experts.

Back to Top

Agile Hardware Development

The Manifesto for Agile Software Development (2001) by Beck et al.[1] revolutionized software development, overcoming the frequent failure of the traditional elaborate planning and documentation in waterfall development. Small programming teams quickly developed working-but-incomplete prototypes and got customer feedback before starting the next iteration. The scrum version of agile development assembles teams of five to 10 programmers doing sprints of two to four weeks per iteration.

Once again inspired by a software success, the third opportunity is agile hardware development. The good news for architects is that modern electronic computer aided design (ECAD) tools raise the level of abstraction, enabling agile development, and this higher level of abstraction increases reuse across designs.

It seems implausible to claim sprints of four weeks to apply to hardware, given the months between when a design is "taped out" and a chip is returned. Figure 9 outlines how an agile development method can work by changing the prototype at the appropriate level.[23] The innermost level is a software simulator, the easiest and quickest place to make changes if a simulator could satisfy an iteration. The next level is FPGAs that can run hundreds of times faster than a detailed software simulator. FPGAs can run operating systems and full benchmarks like those from the Standard Performance Evaluation Corporation (SPEC), allowing much more precise evaluation of prototypes. Amazon Web Services offers FPGAs in the cloud, so architects can use FPGAs without needing to first buy hardware and set up a lab. To have documented numbers for die area and power, the next outer level uses the ECAD tools to generate a chip's layout. Even after the tools are run, some manual steps are required to refine the results before a new processor is ready to be manufactured. Processor designers call this next level a "tape in." These first four levels all support four-week sprints.

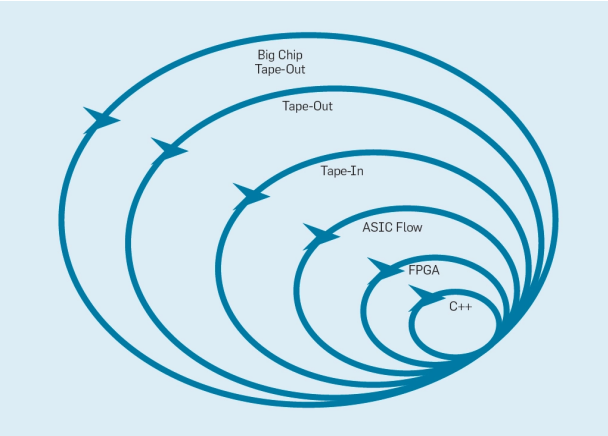


Figure 9. Agile hardware development methodology.

For research purposes, we could stop at tape in, as area, energy, and performance estimates are highly accurate. However, it would be like running a long race and stopping 100 yards before the finish line because the runner can accurately predict the final time. Despite all the hard work in race preparation, the runner would miss the thrill and satisfaction of actually crossing the finish line. One advantage hardware engineers have over software engineers is they build physical things. Getting chips back to measure, run real programs, and show to their friends and family is a great joy of hardware design.

Many researchers assume they must stop short because fabricating chips is unaffordable. When designs are small, they are surprisingly inexpensive. Architects can order 100 1-mm^2 chips for only $14,000. In 28 nm, 1 mm^2 holds millions of transistors, enough area for both a RISC-V processor and an NVLDA accelerator. The outermost level is expensive if the designer aims to build a large chip, but an architect can demonstrate many novel ideas with small chips.

Back to Top

Conclusion

"The darkest hour is just before the dawn." —Thomas Fuller, 1650

To benefit from the lessons of history, architects must appreciate that software innovations can also inspire architects, that raising the abstraction level of the hardware/software interface yields opportunities for innovation, and that the marketplace ultimately settles computer architecture debates. The iAPX-432 and Itanium illustrate how architecture investment can exceed returns, while the S/360, 8086, and ARM deliver high annual returns lasting decades with no end in sight.

The end of Dennard scaling and Moore's Law and the deceleration of performance gains for standard microprocessors are not problems that must be solved but facts that, recognized, offer breathtaking opportunities. High-level, domain-specific languages and architectures, freeing architects from the chains of proprietary instruction sets, along with demand from the public for improved security, will usher in a new golden age for computer architects. Aided by open source ecosystems, agilely developed chips will convincingly demonstrate advances and thereby accelerate commercial adoption. The ISA philosophy of the general-purpose processors in these chips will likely be RISC, which has stood the test of time. Expect the same rapid improvement as in the last golden age, but this time in terms of cost, energy, and security, as well as in performance.

The next decade will see a Cambrian explosion of novel computer architectures, meaning exciting times for computer architects in academia and in industry.

Figure. To watch Hennessy and Patterson's full Turing Lecture, see https://www.acm.org/hennessy-patterson-turing-lecture

Back to Top

References

1. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. ... and Kern, J. *Manifesto for Agile Software Development*, 2001; https://agilemanifesto.org/

2. Bhandarkar, D. and Clark, D.W. Performance from architecture: Comparing a RISC and a CISC with similar hardware organization. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Santa Clara, CA, Apr. 8–11). ACM Press, New York, 1991, 310–319.

3. Chaitin, G. et al. Register allocation via coloring. *Computer Languages 6*, 1 (Jan. 1981), 47–57.

4. Dally, W. et al. Hardware-enabled artificial intelligence. In *Proceedings of the Symposia on VLSI Technology and Circuits* (Honolulu, HI, June 18–22). IEEE Press, 2018, 3–6.

5. Dennard, R. et al. Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE Journal of Solid State Circuits 9*, 5 (Oct. 1974), 256–268.

6. Emer, J. and Clark, D. A characterization of processor performance in the VAX-11/780. In *Proceedings of the 11th International Symposium on Computer Architecture* (Ann Arbor, MI, June). ACM Press, New York, 1984, 301–310.

7. Fisher, J. The VLIW machine: A multiprocessor for compiling scientific code. *Computer 17*, 7 (July 1984), 45–53.

8. Fitzpatrick, D.T., Foderaro, J.K., Katevenis, M.G., Landman, H.A., Patterson, D.A., Peek, J.B., Peshkess, Z., Séquin, C.H., Sherburne, R.W., and Van Dyke, K.S. A RISCy approach to VLSI. *ACM SIGARCH Computer Architecture News 10*, 1 (Jan. 1982), 28–32.

9. Flynn, M. Some computer organizations and their effectiveness. *IEEE Transactions on Computers 21*, 9 (Sept. 1972), 948–960.

10. Fowers, J. et al. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture* (Los Angeles, CA, June 2–6). IEEE, 2018, 1–14.

11. Hennessy, J. and Patterson, D. A New Golden Age for Computer Architecture. Turing Lecture delivered at the *45th ACM/IEEE Annual International Symposium on Computer Architecture* (Los Angeles, CA, June 4, 2018); http://iscaconf.org/isca2018/turing_lecture.html; https://www.youtube.com/watch?v=3LVeEjsn8Ts

12. Hennessy, J., Jouppi, N., Przybylski, S., Rowen, C., Gross, T., Baskett, F., and Gill, J. MIPS: A microprocessor architecture. *ACM SIGMICRO Newsletter 13*, 4 (Oct. 5, 1982), 17–22.

13. Hennessy, J. and Patterson, D. *Computer Architecture: A Quantitative Approach*. Morgan Kauffman, San Francisco, CA, 1989.

14. Hill, M. A primer on the meltdown and Spectre hardware security design flaws and their important implications, *Computer Architecture Today* blog (Feb. 15, 2018); https://www.sigarch.org/a-primer-on-the-meltdown-spectre-hardware-security-design-flaws-and-their-important-implications/

15. Hopkins, M. A critical look at IA-64: Massive resources, massive ILP, but can it deliver? *Microprocessor Report 14*, 2 (Feb. 7, 2000), 1–5.

16. Horowitz M. Computing's energy problem (and what we can do about it). In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers* (San Francisco, CA, Feb. 9–13). IEEE Press, 2014, 10–14.

17. Jouppi, N., Young, C., Patil, N., and Patterson, D. A domain-specific architecture for deep neural networks. *Commun. ACM 61*, 9 (Sept. 2018), 50–58.

18. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., and Boyle, R. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th ACM/IEEE Annual International Symposium on Computer Architecture* (Toronto, ON, Canada, June 24–28). IEEE Computer Society, 2017, 1–12.

19. Kloss, C. *Nervana Engine Delivers Deep Learning at Ludicrous Speed.* Intel blog, May 18, 2016; https://ai.intel.com/nervana-engine-delivers-deep-learning-at-ludicrous-speed/

20. Knuth, D. *The Art of Computer Programming: Fundamental Algorithms, First Edition.* Addison Wesley, Reading, MA, 1968.

21. Knuth, D. and Binstock, A. Interview with Donald Knuth. InformIT, Hoboken, NJ, 2010; http://www.informit.com/articles/article.aspx

22. Kung, H. and Leiserson, C. Systolic arrays (for VLSI). Chapter in *Sparse Matrix Proceedings Vol. 1.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979, 256–282.

23. Lee, Y., Waterman, A., Cook, H., Zimmer, B., Keller, B., Puggelli, A. ... and Chiu, P. An agile approach to building RISC-V microprocessors. *IEEE Micro 36*, 2 (Feb. 2016), 8–20.

24. Leiserson, C. et al. There's plenty of room at the top. To appear.

25. Metz, C. Big bets on A.I. open a new frontier for chip start-ups, too. *The New York Times* (Jan. 14, 2018).

26. Moore, G. Cramming more components onto integrated circuits. *Electronics 38*, 8 (Apr. 19, 1965), 56–59.

27. Moore, G. No exponential is forever: But 'forever' can be delayed! [semiconductor industry]. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers* (San Francisco, CA, Feb. 13). IEEE, 2003, 20–23.

28. Moore, G. Progress in digital integrated electronics. In *Proceedings of the International Electronic Devices Meeting* (Washington, D.C., Dec.). IEEE, New York, 1975, 11–13.

29. Nvidia. *Nvidia Deep Learning Accelerator (NVDLA)*, 2017; http://nvdla.org/

30. Patterson, D. *How Close is RISC-V to RISC-I?* ASPIRE blog, June 19, 2017; https://aspire.eecs.berkeley.edu/2017/06/how-close-is-risc-v-to-risc-i/

31. Patterson, D. RISCy history. *Computer Architecture Today* blog, May 30, 2018; https://www.sigarch.org/riscy-history/

32. Patterson, D. and Waterman, A. *The RISC-V Reader: An Open Architecture Atlas.* Strawberry Canyon LLC, San Francisco, CA, 2017.

33. Rowen, C., Przbylski, S., Jouppi, N., Gross, T., Shott, J., and Hennessy, J. A pipelined 32b NMOS microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers* (San Francisco, CA, Feb. 22–24) IEEE, 1984, 180–181.

34. Schwarz, M., Schwarzl, M., Lipp, M., and Gruss, D. Netspectre: Read arbitrary memory over network. arXiv preprint, 2018; https://arxiv.org/pdf/1807.10535.pdf

35. Sherburne, R., Katevenis, M., Patterson, D., and Sequin, C. A 32b NMOS microprocessor with a large register file. In *Proceedings of the IEEE International Solid-State Circuits Conference* (San Francisco, CA, Feb. 22–24). IEEE Press, 1984, 168–169.

36. Thacker, C., MacCreight, E., and Lampson, B. *Alto: A Personal Computer.* CSL-79-11, Xerox Palo Alto Research Center, Palo Alto, CA, Aug. 7,1979; http://people.scs.carleton.ca/~soma/distos/fall2008/alto.pdf

37. Turner, P., Parseghian, P., and Linton, M. Protecting against the new 'L1TF' speculative vulnerabilities. Google blog, Aug. 14, 2018; https://cloud.google.com/blog/products/gcp/protectingagainst-the-new-l1tf-speculative-vulnerabilities

38. Van Bulck, J. et al. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium* (Baltimore, MD, Aug. 15–17). USENIX Association, Berkeley, CA, 2018.

39. Wilkes, M. and Stringer, J. Micro-programming and the design of the control circuits in an electronic digital computer. *Mathematical Proceedings of the Cambridge Philosophical Society 49*, 2 (Apr. 1953), 230–238.

40. XLA Team. XLA – TensorFlow. Mar. 6, 2017; https://developers.googleblog.com/2017/03/xlatensorflow-compiled.html

Back to Top

Authors

John L. Hennessy (hennessy@stanford.edu) is Past-President of Stanford University, Stanford, CA, USA, and is Chairman of Alphabet Inc., Mountain View, CA, USA.

David A. Patterson (pattrsn@berkeley.edu) is the Pardee Professor of Computer Science, Emeritus at the University of California, Berkeley, CA, USA, and a Distinguished Engineer at Google, Mountain View, CA, USA.

No entries found