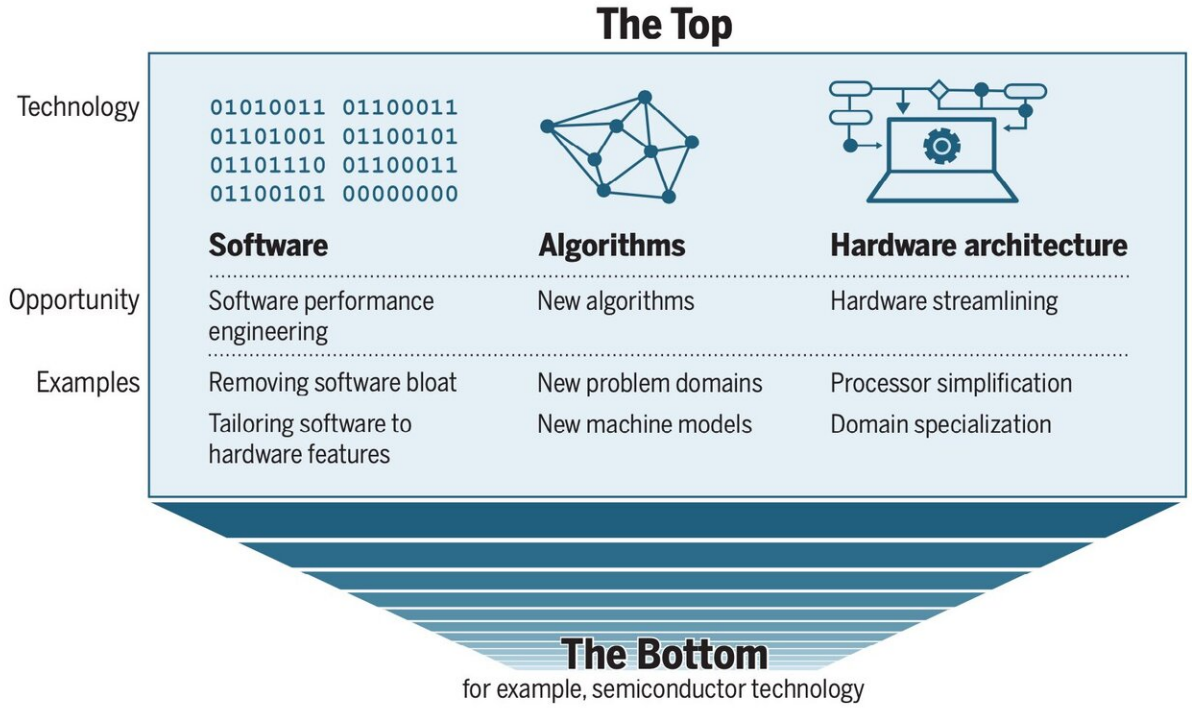
If transistors can't get smaller, then software developers have to get smarter

June 5 2020



Performance gains after Moore's law ends. In the post-Moore era, improvements in computing power will increasingly come from technologies at the Top of the computing stack, not from those at the Bottom, reversing the historical trend.

Credit: Massachusetts Institute of Technology

In 1965, Intel co-founder Gordon Moore predicted that the number of transistors that could fit on a computer chip would grow exponentially

—- and they did, doubling about every two years. For half a century Moore's Law has endured: computers have gotten smaller, faster, cheaper and more efficient, enabling the rapid worldwide adoption of PCs, smartphones, high-speed Internet and more.

This miniaturization trend has led to silicon chips today that have almost unimaginably small circuitry. Transistors, the tiny switches that implement computer microprocessors, are so small that 1000 of them laid end-to-end are no wider than a human hair. For a long time, the smaller the transistors were, the faster they could switch.

But today, we're approaching the limit of how small transistors can get. As a result, over the last decade researchers have been scratching their heads to find other ways to improve performance so that the computer industry can continue to innovate.

While we wait for the maturation of new computing technologies like quantum, carbon nanotubes, or photonics (which may take a while), other approaches will be needed to get performance as Moore's Law comes to an end. In a recent journal article published in *Science*, a CSAIL team identifies three key areas to prioritize to continue to deliver computing speed-ups: better software, new algorithms and more streamlined hardware.

Senior author Charles E. Leiserson says that the performance benefits from miniaturization have been so great that, for decades, programmers have been able to prioritize making the writing of code easier rather than making the code itself run faster. The inefficiency that this tendency introduces has been acceptable, because faster computer chips have always been able to pick up the slack.

"But nowadays, being able to make further advances in fields like machine learning, robotics and virtual reality will require huge amounts

of computational power that miniaturization can no longer provide," says Leiserson, the Edwin Sibley Webster Professor in MIT's Department of Electrical Engineering and Computer Science (EECS). "If we want to harness the full potential of these technologies, we must change our approach to computing."

Leiserson co-wrote the paper with research scientist Neil Thompson, professor Daniel Sanchez, adjunct professor Butler Lampson and research scientists Joel Emer, Bradley Kuszmaul and Tao Schardl. It will be published in the next issue of *Science*, out this week.

No more Moore

The authors make recommendations about three areas of computing: software, algorithms and hardware architecture.

With software, they say that programmers' previous prioritization of productivity over performance has led to problematic strategies like "reduction": taking code that worked on problem A, and using it to solve problem B. For example, if someone has to create a system to recognize yes-or-no voice commands, but doesn't want to code a whole new custom program, they could take an existing program that recognizes a wide range of words, and tweak it to respond only to yes-or-no answers.

While this approach reduces coding time, the inefficiencies it creates quickly compound: if a single reduction is 80 percent as efficient as a custom solution, and you then add twenty layers of reduction, the code will ultimately be 100 times less efficient than it could be.

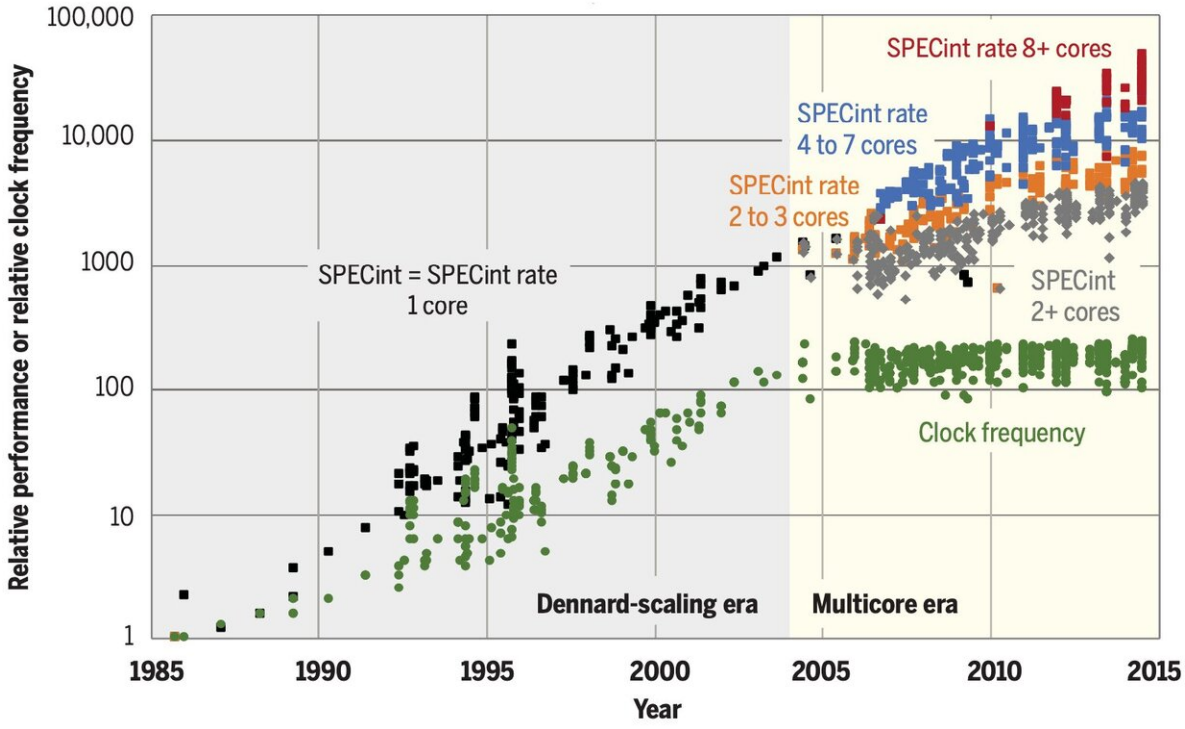


Fig. 2. SPECint (largely serial) performance, SPECint-rate (parallel) performance, and clock-frequency scaling for microprocessors from 1985 to 2015, normalized to the Intel 80386 DX microprocessor in 1985. Credit: Massachusetts Institute of Technology

"These are the kinds of strategies that programmers have to rethink as hardware improvements slow down," says Thompson. "We can't keep doing 'business as usual' if we want to continue to get the speed-ups we've grown accustomed to."

Instead, the researchers recommend techniques like parallelizing code. Much existing software has been designed using ancient assumptions that processors can only do only one operation at a time. But in recent years multicore technology has enabled complex tasks to be completed thousands of times faster and in a much more energy-efficient way.

"Since Moore's Law will not be handing us improved performance on a silver platter, we will have to deliver performance the hard way," says Moshe Vardi, a professor in computational engineering at Rice University who was not part of the project. "This is a great opportunity for computing research, and the [MIT CSAIL] report provides a road map for such research."

For algorithms, the team suggests a three-pronged approach that includes exploring new problem areas, addressing concerns about how algorithms scale, and tailoring them to better take advantage of modern hardware.

In terms of hardware architecture, the team advocates that hardware be streamlined so that problems can be solved with fewer transistors and less silicon. Streamlining includes using simpler processors and creating hardware tailored to specific applications, like the graphics-processing unit (GPU) is tailored for computer graphics.

"Hardware customized for particular domains can be much more efficient and use far fewer transistors, enabling applications to run tens to hundreds of times faster," says Schardl. "More generally, hardware streamlining would further encourage parallel programming, creating additional chip area to be used for more circuitry that can operate in parallel."

While these approaches may be the best path forward, the researchers say that it won't always be an easy one. Organizations that use such techniques may not know the benefits of their efforts until after they've invested a lot of engineering time. Plus, the speed-ups won't be as consistent as they were with Moore's Law: they may be dramatic at first, and then require large amounts of effort for smaller improvements.

Certain companies have already gotten the memo.

"For tech giants like Google and Amazon, the huge scale of their data centers means that even small improvements in software performance can result in large financial returns," says Thompson. "But while these firms may be leading the charge, many others will need to take these issues seriously if they want to stay competitive."

Getting improvements in the areas identified by the team will also require building up the infrastructure and workforce that make them possible.

"Performance growth will require new tools, programming languages and hardware to facilitate more and better performance engineering," says Leiserson. "It also means computer scientists being better educated about how we can make software, algorithms and hardware work together, instead of putting them in different silos."

100x

Code could be 100x less efficient if you reduce it (say) 20 times.

Reduction is when you take code that worked on problem A, and use it to solve problem B. (For example, if someone has to create a system to recognize yes-or-no voice commands, but doesn't want to code a whole new custom program, they could take an existing program that recognizes a wide range of words, and tweak it to respond only to yes-or-no answers.)

While this approach reduces coding time, the inefficiencies it creates quickly compound: if a single reduction is 80 percent as efficient as a custom solution, and you then add 20 layers of reduction, the code will ultimately be 100 times less efficient than it could be.

More information: Charles E. Leiserson et al. There's plenty of room

at the Top: What will drive computer performance after Moore's law?, *Science* (2020). DOI: 10.1126/science.aam9744

Provided by Massachusetts Institute of Technology