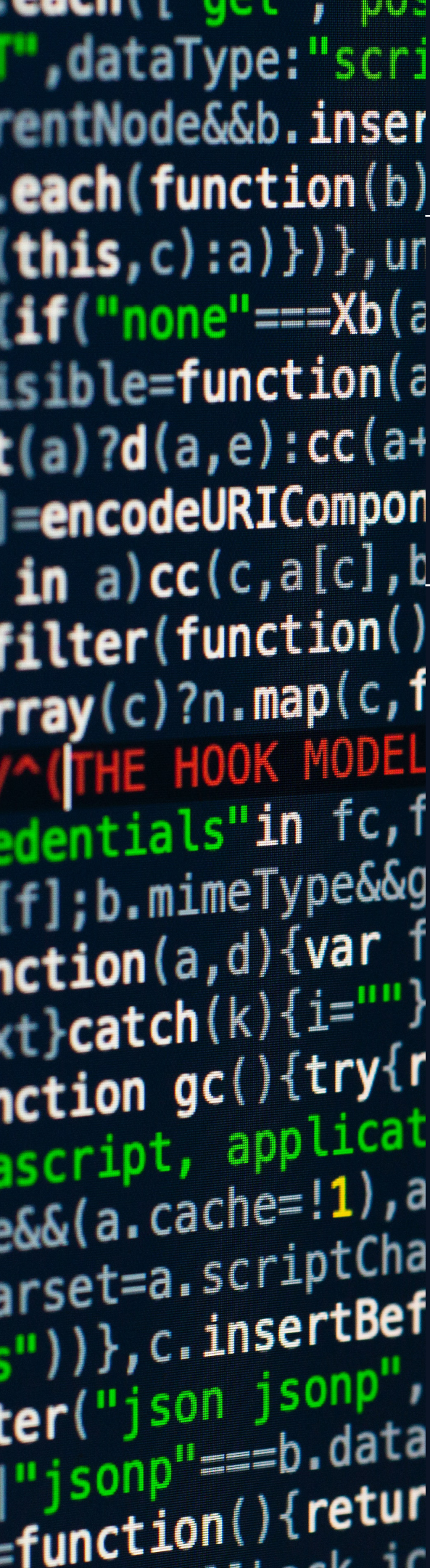

ON THE BRINK OF *DISCONNECTION*

BY CANDY XU

On April 19, 1965, Gordon Moore presented an article that revolutionized the entire computer industry. He pointed out that due to the falling cost of circuit components, we would be able to squeeze more and more of them onto silicon chips over the next several decades.[1] One effect of this increase in components is an increase in speed, a core characteristic that determines the functionality of a computer. Computing speed is highly related to the arrangement of components and communication between signals and code. These design mechanisms all belong to the field of computer architecture. Vital elements of computer architecture include the central processing unit (CPU), random access memory (RAM), read-only memory (ROM), Input and Output (I/O), and system bus.[2] Together, these components construct a path for software and hardware to communicate with each other. If a computer is like a human body, then computer architecture would be the ways in which the brain and the rest of the body, namely the mental and physical components, interact. While existing architecture has already matured greatly since its genesis, the next decade will likely bring about efforts to revise the current architectural design further so as to continue increasing computing speed.[3]

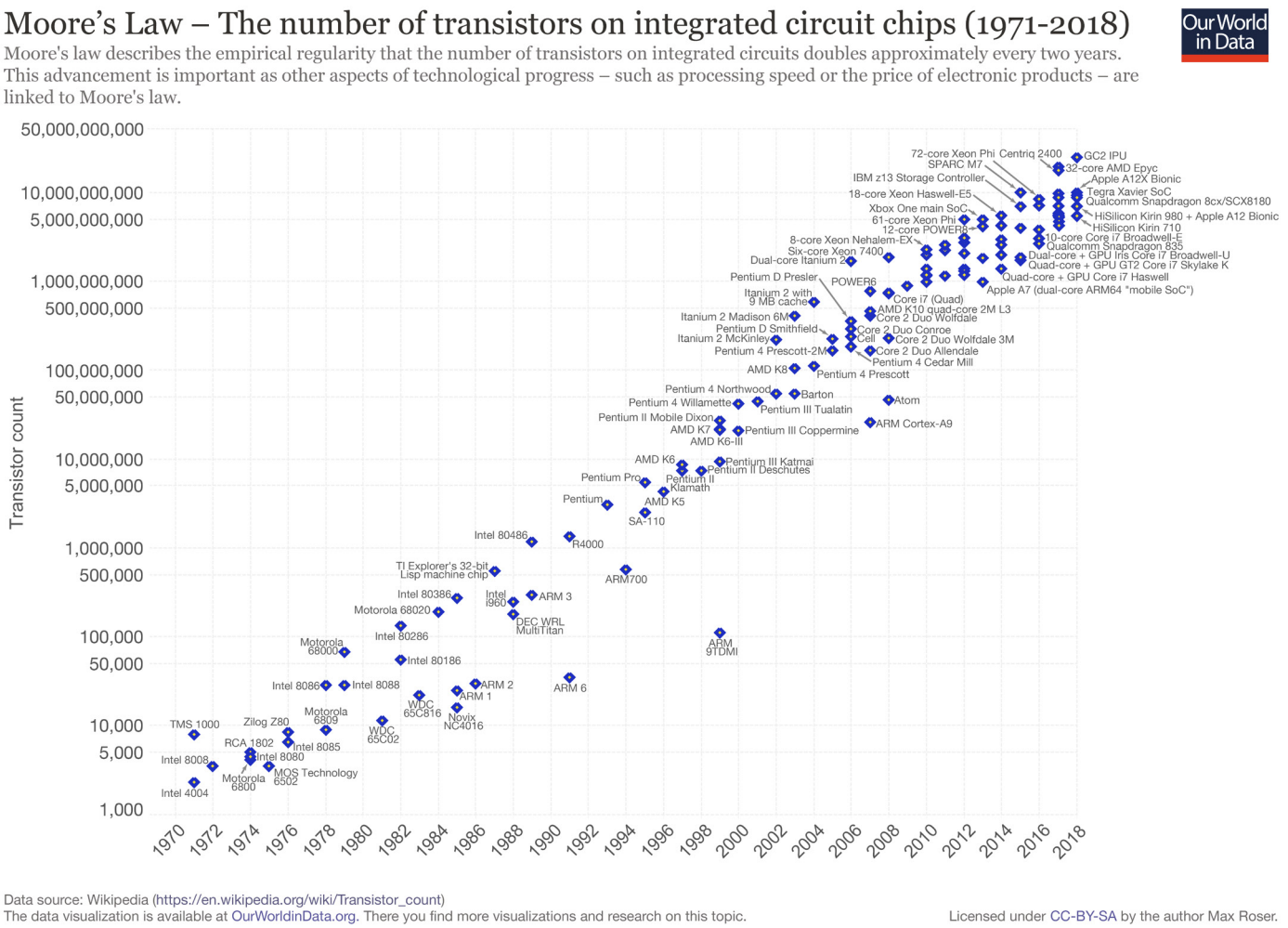

Figure 1: Transistor Count.[11] *The increase of transistors on chips started to slow down in the 21st century.*

MOORE'S LAW

Computing speed is important for computer architects to consider, and improving chips is one way to achieve speedup. Most chips depend on complementary metal-oxide-semiconductor (CMOS) transistors, a type of semiconductor used in integrated circuits and digital logic design. For more than 30 years, we enjoyed "free" computer speedup by brutally adding more and more transistors on chips.[4] This is the idea that Moore presented in his 1965 article. He predicted that the number of transistors would double every year, later revising the statement to every two years.[4]

With this wonderful physical capability, the era of the "lazy software engineer" had begun. The speed of computers did not need to rely on code efficiency or runtime. All that programmers had to do was to wait for microchips to gain more transistors in order for their computers to run faster. This effortless speedup continued until we finally approached the physical limits of the technology. The time frame in which transistors double has increased tremendously because the transistors simply cannot get any smaller (Fig. 1). Currently, the smallest dimension of CMOS on electronic device is close to 10-20 atomic diameters.[5] Any smaller than that and CMOS transistors would physically stop working.[5] As a result, computer scientists began to explore other venues to accelerate computer performance.

LANGUAGE DECODING

With a stagnation in speed, we would lose the ability to implement more powerful programs or enhance the performance of existing ones. The slowing down of Moore's Law thus poses a pressing issue to the whole

> "The slowing down of Moore's Law thus poses a pressing issue to the whole industry: how can we get computers to operate on a faster timescale?"

industry: how can we get computers to operate on a faster timescale? Now that we cannot rely solely on hardware improvements, the solution probably lies in software or the intersection of software and hardware.

Computer scientists have already discovered an area with high potential for increasing computing speed—decoding from high-level programming languages to low-level ones.[3] Languages that are more abstract can be thought of as higher level since they are more readable to humans and easier to use when programming. However, they are thus more structurally complicated and have longer runtimes. Languages that are less abstract can be thought of as lower level, as they usually refer to assembly code or machine code, which can communicate directly with hardware without the need for a complier. Python is one of the most popular examples of a high-level programming language, while the language C is a great example of a language that is less abstract and lower level than Python.

Jun and Ling once performed recursion, a function that calls on itself, to a Fibonacci series calculation using both Python and C to illustrate the significant runtime difference in the two languages: the computation took Python 3.0 about 2.5396 seconds and took C <<0.0001 seconds.[6] This hints at the great potential in speeding up high-level languages like Python by using more efficient techniques to decode them into low-level ones. Currently, JIT is working in that area by directly converting real-time Python models to machine code execution and using cache, a smaller but more accessible storage area for disk data, to temporarily store recently used data, respectively.[6]

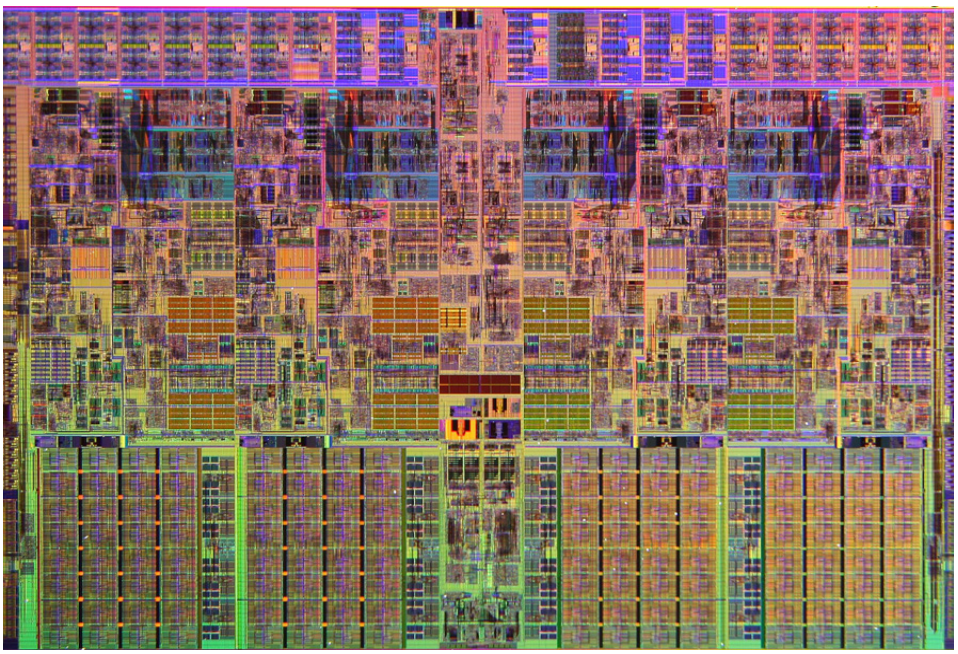

Figure 3: Intel Core i7 Processor.[13] *Intel's Core i7 is an example of a well-architected processor that is widely used in industry right now.*

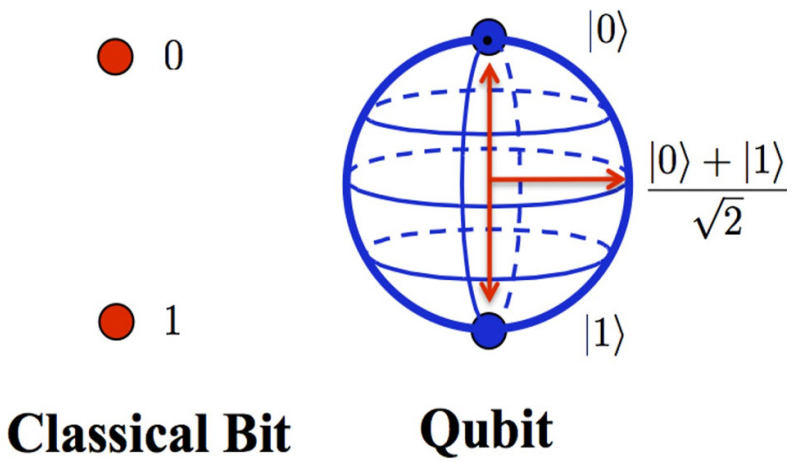

Figure 2: Classical Bit vs. Qubit.[12] *Qubit allows access to many more states than classical bits. The combination of multiple states generate quantum superpositions, which tell us that we can freely add two quantum states together and obtain another valid outcome.*[9] *However, we have to be careful when measuring a superposition state as random results might appear for certain measurements. On the other hand, entangled states cannot be separated. Their ways of combination cannot be recreated, but the two states do have perfect correlation.*[9] *When measuring one state, the other will behave exactly the same.*

DOMAIN-SPECIFIC LANGUAGES

Turing Award winner and Professor at University of California, Berkeley, David Patterson also suggests that another emerging field which may significantly increase computing speed is domain-specific languages.[3] Unlike general languages, such as Java and Python, which can be used in a variety of applications, domain-specific languages are customized for a certain field of interest. For example, Matlab is primarily for numerical computing.

Schaumont and Verbauwhede once ran a 128-bit key Advanced Encryption Standard algorithm using a completely customized program involving application-specific integrated circuit (ASIC) and Java. It turns out the customization increased the performance by a factor of nearly 3 million.[7] Therefore, in order to support domain-specific computing, we would need greater customizable computer architecture. It is not

"These 10 years of disconnection in the speed up of computers would likely rely on re-structuring the way software languages communicate with each other and their hardware."

only easier to run domain-specific languages on such a architecture, but also cheaper. The cost of implementing all applications using ASICs with a 45nm CMOS is already exceeding $50,000,000, implying the need for another architectural method if we want to spread the use of customized computing.[7] Technologies such as CHP, a customizable heterogeneous platform that integrates customizable cores and tunes performance to a specific application's needs, are currently being researched and developed.

QUANTUM COMPUTING

Other newly emerging ideas are also trying to break through the barrier of speed limits. Quantum computing is one that gained a huge amount of attention recently. Instead of calculating information based on binary systems, which consist only of two levels (0 or 1), quantum systems can distinguish between multiple levels and enable data access to many parts of the computer simultaneously. They rely on qubits, superposition, and entanglement, which together allow us to manipulate combinations of individual states (Fig. 2).[8]

Although methods such as harnessing entanglement for computation have boosted quantum computing speed, this technology is so new and powerful that a real-world application has not been found yet. Professor Patterson and others suggest that a tangible application of quantum computing is most likely not going to take effect within the next decade.[9,10]

Thus, these ten years of disconnection in the speed up of computers would likely rely on re-architecturing the way that software languages communicate with each other and their hardware counterparts. If developed correctly, we can achieve as great of an increase in computing speed as we saw in the era of the "lazy software engineer."

Living in this era of great technological advancement, we have the opportunity to join the battle and get involved with the next golden decade of computer re-architecture. In order to reconstruct the current system, we must redefine the means of interaction between software and hardware. Collectively, such efforts could not only power development in newly emerging technologies, but could in turn push past the boundaries that are currently preventing the next generation of advancements in computing.

Acknowledgements: I would like to express my sincere appreciation and acknowledge Professor David Patterson (Professor Emeritus of Computer Science at UC Berkeley and Google Distinguished Engineer) for his support during the writing process.

REFERENCES

1. Moore, G. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE, 86*(1), 82-85. doi:10.1109/jproc.1998.658762
2. Wells, C. J. (2009, January 28). Computer architecture. *Technology UK.* Retrieved from http://www.technologyuk.net/computing/computer-systems/architecture.shtml
3. Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. *Communications of the ACM, 62*(2), 48-60. doi:10.1145/3282307
4. Brenner, A. (1997). Moore's law. *Science, 275*(5306), 1401-1404. Retrieved from http://www.jstor.org/stable/2893682
5. Spencer, M. (2018). The end of Moore's law. *US Black Engineer and Information Technology, 42*(1), 76-76. Retrieved from http://www.jstor.org/stable/26305580
6. Jun, L., & Ling, L. (2010, October). Comparative research on Python speed optimization strategies. In *2010 International Conference on Intelligent Computing and Integrated Systems* (pp. 57-59). IEEE.
7. Cong, J., Sarkar, V., Reinman, G., & Bui, A. (2011). Customizable domain-specific computing. *IEEE Design & Test of Computers, 28*(2), 6-15. doi: 10.1109/MDT.2010.141
8. IBM Research Editorial Staff. (2019, February 08). Quantum computing: you know it's cool, now find out how it works. *IBM*. Retrieved from https://www.ibm.com/blogs/research/2017/09/qc-how-it-works/
9. Patterson, D. (2019, February 25). Talk with Professor David Patterson [Personal interview].
10. National Academies of Sciences, Engineering, and Medicine. (2019). *Quantum Computing: Progress and Prospects.* Washington, DC: The National Academies Press. doi: 10.17226/25196

IMAGE REFERENCES

11. Roser, M. (2019). *Moore's Law—the number of transistors on integrated circuit chips (1971-2016)* [digital image]. Retrieved from https://commons.wikimedia.org/wiki/File:Moore%27s_Law_Transistor_Count_1971-2018.png
12. Zahid, H. (2016). Strengths and Weaknesses of Quantum Computing. *International Journal of Scientific and Engineering Research, 7*(9), 1526-1531.
13. Intel. (2008). *Nehalem die shot 2* [digital image]. Retrieved from https://www.intel.com/pressroom/archive/releases/2008/20081117comp_sm.htm