Motivation for and Evaluation of the First Tensor Processing Unit

Norman P. Jouppi, Cliff Young, Nishant Patil, and David Patterson
Google

The first-generation tensor processing unit (TPU) runs deep neural network (DNN) inference 15-30 times faster with 30-80 times better energy efficiency than contemporary CPUs and GPUs in similar semiconductor technologies. This domain-specific architecture (DSA) is a custom chip that has been deployed in Google datacenters since 2015, where it serves billions of people.

DNNs have led to breakthroughs such as cutting the error rate in an image recognition competition from 26 to 3.5 percent since 2011 and beating a human champion at Go. Ironically, the success of DNNs led to a potential crisis in 2013. If users searched by voice for three minutes a day using speech recognition DNNs, computation demands on Google's datacenters would double. This increase would be unaffordable using conventional CPUs. Indeed, matrix multiply, which is at the heart of DNN computations, rose significantly in popularity in our datacenter workload.

In response, Google started a high-priority project to produce a custom chip for DNNs. The goal was to improve cost-performance by 10X over CPUs in just 15 months, including design, fabrication, testing, and deployment. This article gives an overview of the motivation, architecture, power, and performance of the resulting TPU. Our original article goes into greater depth.[1]

DNN

The two phases of DNN are called training (or learning) and inference (or prediction), and they refer to development versus production. The goal for the first-generation TPU was to accelerate inference by 10X. (The second-generation TPU tackles training.[2])

Training computes the weights (parameters) of a DNN, adjusting them repeatedly until the DNN produces the desired results. Virtually all training is in floating point. Quantization transforms floating-point numbers into narrow integers—often only 8 bits—which are usually good enough for inference. Eight-bit integer multiplies can be 6X less energy and 6X less area than IEEE 754 16-bit floating-point multiplies, and the edge for integer addition is 13X and 38X, respectively.

MOORE'S LAW AND CPU VERSUS DSA DESIGN

CPU architects historically used the O(n^2) more transistors per chip provided by a Moore's-law scaling by a factor of *n* to build single-core computers with increasingly aggressive implementations and bigger cache memories, up to the limits of single-core design. After this limit was reached (circa 2003 with Intel Core Duo), they used transistors to increase throughput, so the number of cores essentially grows with O(n^2).

Alas, the peak power per mm^2 of chip was increasing (due to the end of Dennard scaling), but the power budget per chip was limited (due to electromigration, mechanical, and thermal limits), and we already tried switching to multiple cores (limited by Amdahl's law). Given these constraints, the architects needed to take a radically different approach to deliver the next 10X.

They decided to build a DSA, which allowed either omitting or reallocating conventional CPU resources that DNNs do not need. Unlike some DSA targets, DNNs cover numerous problems, allowing reuse of a DNN-specific ASIC for speech, vision, language, translation, and search ranking. Ideas in Mark Horowitz's keynote speech inspired the design.[3] Figure 1 shows the energy overhead to perform a single addition on the traditional CPU: An 8-bit add that uses only 0.03 picoJoules (pJ) has an overhead of 70 pJ. If a single instruction did hundreds or thousands of 8-bit additions or multiplies, it could perform 10X more operations per second and live within the power constraints of the chip.

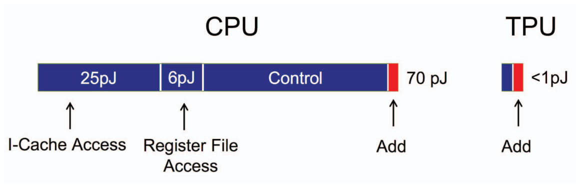

Figure 1. Energy breakdown (not to scale) of instruction fetch and execute of a traditional CPU on the left versus a TPU on the right.

The good news is that DNNs are dominated by tensors, so the architects created instructions that operate on tensors of data rather than one data element per instruction. As vectors are tensors of Order 1 (O(n)) and 2D matrices are tensors of Order 2 (O(n^2)), they could use the O(n^2) transistors from Moore's law to support multiplication of 2D matrices natively. While a straightforward matrix multiply requires O(n^3) operations, a chip could compute it in O(n) time given a large 2D array in the matrix multiply unit.

From a historical perspective, 1D vectors have been in Cray supercomputers since the Cray-1 in 1976 and in desktop computers since the x86 MMX extension in 1997. The TPU generalizes such hardware structures to 2D, which greatly expands the operations per instruction. The matrix operations use a 256x256 array for 8-bit data. This approach reduced the energy overhead per computation by more than 10X by increasing the reuse of fetched memory and register values by more than 100X. Figure 1 illustrates the slim energy overhead per operation for the TPU.

TPU HARDWARE

Figure 2 shows the block diagram of the TPU. The internal blocks are connected by 256-byte-wide paths. The matrix multiply unit is the heart of the TPU. It contains 256x256 multiply-accumulators (MACs) that can perform 8-bit multiplies on integers. The 16-bit products are summed in the 4 mebibytes (MiB) of 32-bit accumulators below the matrix unit. It holds 4,096 256-element, 32-bit accumulators. The matrix unit produces one 256-element partial sum per cycle.

The weights that the DNN needs for the matrix unit are staged through an on-chip weight FIFO that reads from an off-chip 8- gibibyte (GiB) DRAM called weight memory. The intermediate results are held in the 24-MiB on-chip unified buffer, which can serve as input to the matrix unit. A programmable DMA controller transfers data to or from CPU host memory and the unified buffer. To deploy dependably at Google scale, internal and external memory have error detection and correction hardware.

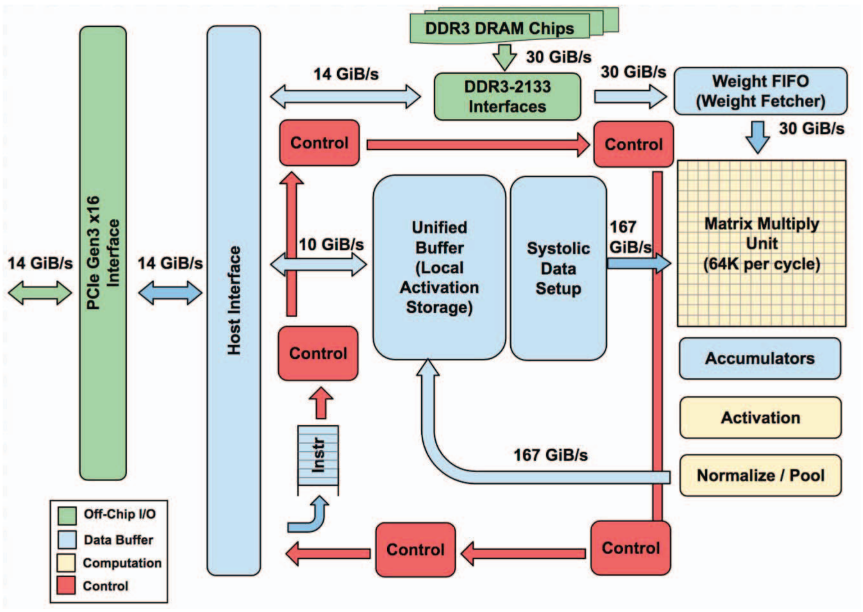

Figure 2. TPU block diagram. The main computation is the yellow matrix multiply unit. Its inputs are the blue weight FIFO and the blue unified buffer, and its output is the blue accumulators. The yellow activation unit operates on the accumulators, whose results go to the unified buffer.

The philosophy of the TPU microarchitecture is to keep the large matrix unit busy. Toward that end, the instruction that reads the weights follows the decoupled-access/execute philosophy,[4] in that it can complete after sending its address, but before the weight is fetched from weight memory. The matrix unit will stall if the input activation or weight data is not ready.

As reading a large SRAM uses much more power than arithmetic, the matrix unit uses systolic execution to save energy by reducing reads and writes of the unified buffer.[5] It relies on data from different directions arriving at cells in an array regularly where they are combined. A given 256-element matrix-vector multiplication moves through the matrix as a diagonal wavefront. The weights are preloaded and take effect with the advancing wave alongside the first data of a new block. Control and data are pipelined to give software the illusion that the matrix unit reads 256 inputs simultaneously, and that they instantly update one location of each of the 256 accumulators.

To reduce time of deployment, the TPU was designed to be a coprocessor on the PCI Express (PCIe) I/O bus rather than be tightly integrated with a CPU, allowing it to plug into existing servers just as a GPU does. The goal was to run whole inference models in the TPU to reduce I/O between the TPU and the host CPU.

Moreover, to simplify hardware design and debugging, the host server sends TPU instructions over the PCIe bus for it to execute rather than having the TPU sequence itself. Hence, the TPU is closer in spirit to a floating-point coprocessor than it is to a GPU. As instructions travel over this relatively slow bus, the TPU follows the CISC tradition. It has about a dozen instructions, such as Read_Host_Memory, Read_Weights, Matrix_Multiply, and Write_Host_Memory. The average clock cycles per instruction of these CISC instructions is typically 10 to 20.

Figure 3 shows the floor plan of the TPU die. The 24-MiB unified buffer is almost a third of the die and the matrix multiply unit is a quarter, so the data path is nearly two-thirds of the die. Control is just 2 percent. Control is much larger (and harder to design) in a CPU or GPU. Figure 4 shows the TPU on its printed circuit card, which inserts into existing servers such as a serial advanced technology attachment (SATA) disk.

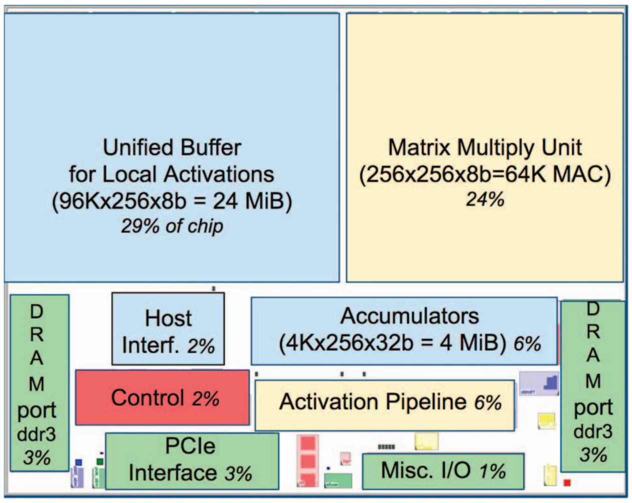

Figure 3. Floorplan of TPU die. The shading follows Figure 2. The blue data path is 67 percent, the green I/O is 10 percent, and the red control is only 2 percent of the die.

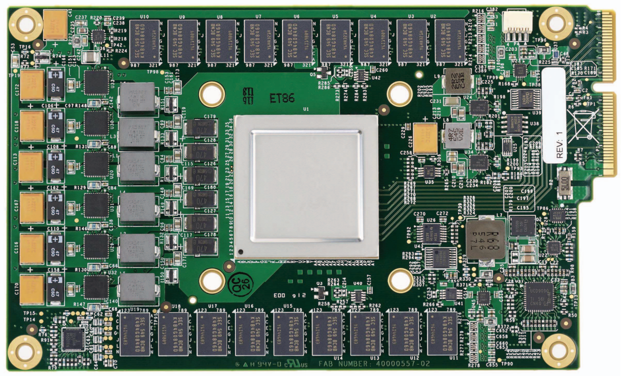

Figure 4. TPU printed circuit board, which can be inserted into the slot for a disk in a server.

TPU BENCHMARKS

Most architecture articles rely on simulations running small, easily portable benchmarks that project potential performance if the hardware is ever implemented. This article is not one of those. It is a retrospective evaluation of real machines running large production workloads in datacenters since 2015, some of which more than 1 billion people use routinely.

Table 1 shows the six production DNN applications that we used to evaluate the TPU, which represented 95 percent of inference workload in our datacenters in 2016. As most research articles in computer architecture concentrated on convolutional neural networks (CNNs), we were surprised that they were only 5 percent of Google's production inference workload. The portion of the production applications running on the TPU is surprisingly small, only 100 to 1,500 lines of TensorFlow.[6] Our benchmarks are small pieces of larger applications that run on the host server, which can be thousands to millions of lines of C++ code. The applications are often user-facing, which leads to rigid response-time limits.

Each model needs between 5 and 100 million weights (see Table 1), which can take a lot of time and energy to access. To amortize the access costs from the off-chip DRAM, the same weights are reused across a batch of independent examples, which improves performance. Table 2 compares the TPU to two large chips contemporary to the TPU: an 18-core Intel Haswell CPU and the Nvidia Kepler K80 GPU. The TPU is about half the area of Haswell and Kepler, and half the power, yet packs 25 times the MACs (65,536 8-bit versus 2,496 32-bit) and 3.5 times the on-chip memory (28 MiB versus 8 MiB) as the GPU.

Table 1. Six DNN applications (two per DNN type). MLP stands for multi-layer perceptron, and LSTM stands for long short-term memory.

Name	Lines of Code	Weights	TPU Ops / Weight Byte	Percent of Deployed TPUs in July 2016
MLP0	100	20 M	200	61%
MLP1	1,000	5 M	168	
LSTM0	1,000	52 M	64	29%
LSTM1	1,500	34 M	96	
CNN0	1,000	8 M	2,888	5%
CNN1	1,000	100 M	1,750	

Table 2. Benchmarked servers use Haswell CPUs, K80 GPUs, and TPUs. The GPU and TPU use the Haswell server as host. The TPU die is less than half the Haswell die size.

Model	mm^2	nm	MHz	Thermal Design Power (TDP) per Chip	Cores	TeraOps/s		Memory Gbyte/s	Chips/ Server	TDP per Server
						8 bit	FP32			
Haswell	662	22	2,300	145 W	18	2.6	1.3	51	2	504 W
Nvidia K80	561	28	560	150 W	13	--	2.8	160	8	1,838 W
TPU	< 331	28	700	75 W	1	92	--	34	4	861 W

To illustrate the performance of the six applications, we adapt the Roofline Performance model from high-performance computing.[7] This simple visual model offers insights into the causes of performance bottlenecks. The assumption behind the model is that applications don't fit in on-chip memory, so they are either computation-limited or memory bandwidth-limited. The *y*-axis is performance in operations per second, thus the peak computation rate forms the "flat" part of the roofline. The *x*-axis is operational intensity, measured as operations per DRAM byte accessed. Memory bandwidth is bytes per second, which turns into the "slanted" part of the roofline because (flops/s) / (flops/byte) = bytes/s. Without sufficient operational intensity, a program is memory bandwidth-bound and lives under the slanted part of the roofline. Figure 5 shows the rooflines and the performance of the six programs on our platforms. The TPU's roofline is most slanted because it has very high peak performance comparatively, but relatively slow memory bandwidth.

The six DNN applications are close to the TPU ceiling but generally further below their ceilings for Haswell and K80. Response time is the surprising reason. For example, the application developer of MLP0 required a 99th-percentile response-time limit of 7 ms. The reason for tight response-time limit is that inference applications like MPL0 are part of end-user-facing services, which have notoriously rigid limits to maintain user satisfaction.

At that limit, Table 3 shows that Haswell and the K80 run at just 42 and 37 percent, respectively, of the highest throughput achievable for MLP0. If the response-time limit was relaxed, these applications would run much closer to the CPU and GPU rooflines. These bounds affect the TPU, as well. But at 80 percent, it is operating much closer to its highest MLP0 throughput, so the applications are closer to the TPU roofline. Compared to CPUs and GPUs, the single-threaded TPU has no sophisticated microarchitectural features that consume transistors and energy to improve the average case but not the 99th-percentile case: no caches, branch prediction, out-of-order execution, multiprocessing, speculative prefetching, address coalescing, multi-threading, context switching, and so on. Minimalism is a virtue of domain-specific processors. Table 4 shows that the TPU is, on average, 29 and 15 times faster than the CPU and GPU.

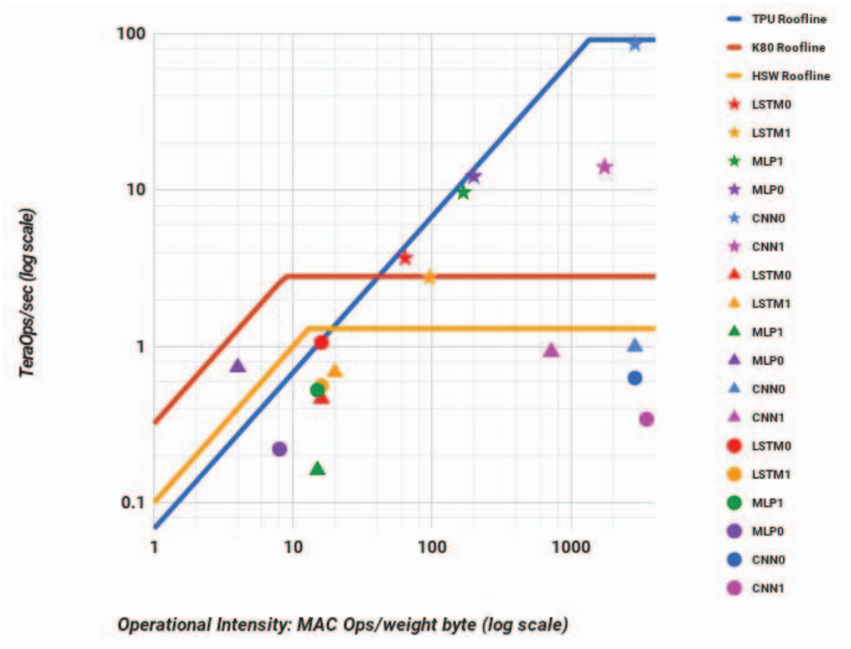

Figure 5. Roofline performance model of the TPU (stars), K80 (triangles), and Haswell (circles). All TPU stars are at or above the other two rooflines.

Table 3. 99th-percentile response time versus throughput (inferences per second) for MLP0.

Type	Batch Size	99th% Response	Inferences/s (IPS)	% Max IPS
CPU	16	7.2 ms	5,482	42%
	64	21.3 ms	13,194	100%
GPU	16	6.7 ms	13,461	37%
	64	8.3 ms	36,465	100%
TPU	200	7.0 ms	225,000	80%
	250	10.0 ms	280,000	100%

In datacenters, cost-performance trumps performance. We can't disclose our costs, but power is correlated with total cost of ownership, and we can report performance/Watt. Figure 6 shows the relative performance/Watt of GPU versus CPU (blue bar), TPU versus CPU (red), and TPU versus GPU (orange). The relative performance/Watt (ignoring host power) is 83 for TPU versus CPU and 29 for TPU versus GPU. We also evaluated a hypothetical TPU (TPU') using the same much-faster GDDR5 memory as used in the GPU. As two-thirds of the benchmarks are memory-bound, Figure 6 shows that the relative performance/Watt (ignoring host power) of TPU' leapt to an amazing 196X over Haswell (green bar) and 68X over the K80 (purple).

Table 4. K80 GPU die and TPU die performance relative to CPU for the DNN workload. The mean uses the actual mix of the six apps in Table 1.

Type	DNN		LSTM		CNN		Weighted Mean
	0	1	0	1	0	1	
GPU	2.5	0.3	0.4	1.2	1.6	2.7	1.9
TPU	41.0	18.5	3.5	1.2	40.3	71.0	29.2

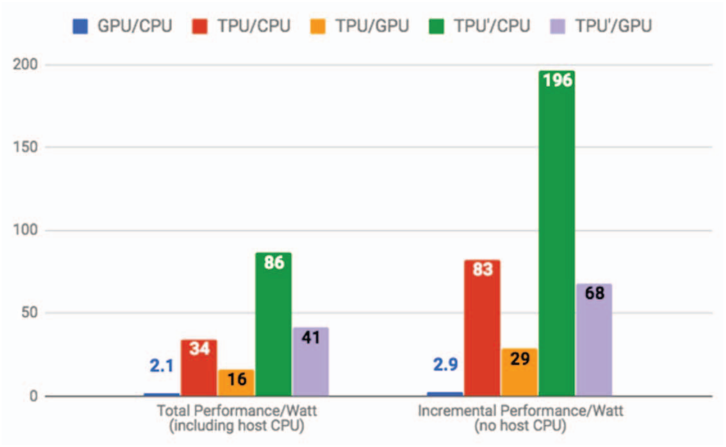

Figure 6. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU using the K80's GDDR5 memory. The green bar shows its ratio to the CPU server, and the lavender bar shows its relation to the GPU server. Total includes host server power, but incremental doesn't.

DISCUSSION

This section presents fallacies and a pitfall in the rebuttal style popular in architecture textbooks.

Fallacy: DNN Inference Applications in Datacenters Value Throughput as much as Response Time.

We were surprised that our developers had strong response-time demands, as some suggested in 2014 that batch sizes would be large enough for the TPU to reach peak performance or that latency requirements wouldn't be as tight. One driving application was off-line image processing, and the intuition was that if interactive services also wanted TPUs, most would just accumulate larger batches. Even the developers of one application in 2014 that cared about response time (LSTM1) said the limit was 10 ms in 2014, but shrank it to 7 ms when they ported it to the TPU. The unexpected desire for TPUs by many such services combined with the impact on and preference for low response time changed the equation, with application writers often opting for reduced latency over waiting for bigger batches to accumulate. Fortunately, the TPU has a simple and repeatable execution model to help meet the response-time targets of interactive services and such high peak throughput that even relatively small batch sizes result in higher performance than contemporary CPUs and GPUs.

Fallacy: The K80 GPU Architecture Is a Good Match to DNN Inference.

GPUs have traditionally been seen as high-throughput architectures that rely on high-bandwidth DRAM and thousands of threads to achieve their goals. This perspective helps explain why the

K80 is only a little faster at latency-bound inference than Haswell and much slower than the TPU. (The K80 was designed long before the recent popularity of DNNs, so it provides no specific support for them.) Successors to the K80 will surely include optimizations to improve peak inference performance, but given their throughput-oriented architectural approach, it might be challenging to do so while meeting strict latency limits. Figure 6 illustrates that there is also plenty of headroom to improve the TPU, so it's not an easy target.

Pitfall: Being Ignorant of Architecture History when Designing a DSA.

Ideas that didn't fly for general-purpose computing might be ideal for DSAs. For the TPU, three important architectural ideas date from the early 1980s: systolic arrays,[5] decoupled-access/execute,[4] and CISC instructions. The first reduced the area and power of the large matrix multiply unit, the second fetches weights concurrently during operation of the matrix multiply unit, and the third better utilizes the limited bandwidth of the PCIe bus for delivering instructions. History-aware architects could have a competitive edge in this DSA era.

RELATED WORK

While hardware for neural networks goes back at least 20 years,[8] there has been recent enthusiasm both in academia[9] and industry.[10] Indeed, 15 percent of articles at a 2016 computer architecture conference were on DNNs, and it was recently reported that there are at least 45 hardware startups working on chips for DNNs.[11] (Space and reference limits constrain this related-work section; see our original article for more.[1])

CONCLUSION

Despite having designed and deployed the TPU in only 15 months and having relatively low memory bandwidth—four of the six applications are memory-bound—a small fraction of a big number can nonetheless be relatively large. This result suggests a "cornucopia corollary" to Amdahl's law: Low utilization of a huge, cheap resource can still deliver a high-performing, cost-effective, and power-efficient solution.

The TPU leverages the order-of-magnitude reduction in energy and area of 8-bit integer systolic matrix multipliers over 32-bit floating-point data paths of a K80 GPU to pack 25 times as many MACs (65,536 8-bit versus 2,496 32-bit) and 3.5 times the on-chip memory (28 MiB versus 8 MiB), while using less than half the power of the K80 in a relatively small die. This larger memory helps increase the operational intensity of applications to let them utilize the abundant MACs more fully. The single-threaded TPU is also a better match to the 99th-percentile response-time bounds of user-facing inference applications than the highly threaded CPU and GPU. The TPU leverages its advantages to run 15 times as fast as the K80 GPU, resulting in a performance/Watt advantage of 29 times. Compared to the Haswell CPU, the corresponding ratios are 29 and 83. While future CPUs and GPUs will surely run inference faster, a redesigned TPU using circa-2015 GPU memory would go three times faster and boost the performance/Watt advantage to nearly 70 over the K80 and 200 over Haswell.

Order-of-magnitude differences between commercial products are rare in computer architecture. Seven factors explain this energy-performance gain, listed roughly in order of importance:

1. The TPU has one very large 2D multiply unit, while the CPU and GPU have many smaller, 1D multiply units (one each in the 18 cores or 13 symmetric multiprocessors). The matrix multiplies inherent to DNNs benefit from 2D hardware.
2. The TPU's DNN applications use 8-bit integers rather than 32-bit floating point to improve efficiency of computation, memory bandwidth, and memory capacity.
3. The 2D organization enables systolic arrays,[5] which reduce register accesses and energy.

4. Because the TPU is a DSA, it can drop features required by CPUs and GPUs that DNNs don't use. Such omissions make the TPU cheaper, save energy, and allow transistors to be repurposed for domain-specific optimizations.
5. The TPU has one thread, while the K80 has 13 and the CPU has 18. A single thread makes it easier to stay within a fixed latency limit of our inference applications, as well as save energy.
6. The TPU has enough flexibility to implement the DNNs of 2018, as well as of 2013.
7. The production applications were written using TensorFlow, which made it easier to port them to the TPU at high performance rather than them having to be rewritten to run well on this very different DSA.

We observe that, despite what hardware is available, machine learning (ML) researchers have a "patience threshold" that limits how long they are willing to wait for the results of an experiment.[12] This impatience is especially true for training, although fast inference helps some applications. For example, AlphaGo trains itself through self-play, with inference accounting for 98 percent of training ML cycles. TPUs accelerated both training and inference in AlphaGo.

Faster hardware enables larger, more-ambitious experiments within the confines of this patience threshold; smaller experiments run the risk that they won't explore a large-enough space to improve ML accuracy. Thus, the demand for ML cycles is effectively unbounded. Speeding up model evaluation by any amount is valuable; factors of 10 or 100 would likely increase the rate of recent breakthroughs such as those mentioned in the introduction.

Alas, just when ML offers a burst of seemingly inexhaustible demand for computing cycles, computer architecture is nearing the end of Moore's law and Dennard scaling. Their death raises the stakes even higher for system designers: We need to solve exponentially harder problems without the exponential increase in resources that Moore's law used to deliver. Architects need to innovate to empower their ML colleagues to help them deliver on the promise of ML.[12]

Encouraging results such as the TPU and the growing appetite for faster ML hardware suggest that the next decade promises to be an exciting one for computer architecture. We project another Renaissance in computer architecture, much like the early 1980s, where the chance to have high impact draws many brilliant young people to the field.

ACKNOWLEDGMENTS

It takes a village to design, verify, and implement the hardware and software of a system like a TPU and to manufacture, deploy, and use it at scale, which is why there are many authors on our original article.[1] We thank Thomas Norrie, Andy Phelps, Bjarke Roune, and the anonymous reviewers for their comments on a draft of this article.

REFERENCES

1. N.J. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," *In Proceedings of the 44th Annual International Symposium on Computer Architecture* (ISCA), 2917, pp. 1–12.
2. J. Dean and U. Hölzle, "Build and train machine learning models on our new Google Cloud TPUs," blog, 2017; www.blog.google/topics/google-cloud/google-cloud-offer-tpus-machine-learning/.
3. M. Horowitz, "Computing's energy problem (and what we can do about it)," *IEEE International Solid-State Circuits Conference Digest of Technical Papers* (ISSCC), 2014, pp. 10–14.
4. J.E. Smith, "Decoupled access/execute computer architectures," *Proceedings of the 9th Annual International Symposium on Computer Architecture* (ISCA), 1982, pp. 112–119.
5. H.T. Kung and C.E. Leiserson, "Algorithms for VLSI processor arrays," *Introduction to VLSI systems*, Addison Wesley, 1980.

6. M. Abadi et al., “TensorFlow: A System for Large-Scale Machine Learning,” *The 12th USENIX Symposium on Operating Systems Design and Implementation* (OSDI), USENIX, 2016, pp. 265–283.
7. S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Communications of the ACM* (CACM), vol. 52, no. 4, 2009, pp. 65–76.
8. K. Asanović, “Programmable Neurocomputing,” *The Handbook of Brain Theory and Neural Networks: Second Edition*, MIT Press, 2002.
9. Y. Chen et al., “DianNao Family: Energy-Efficient Hardware Accelerators for Machine Learning,” *Communications of the ACM*, vol. 59, no. 11, 2016, pp. 105–112.
10. A Putnam et al., “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services,” *Communications of the ACM*, vol. 59, no. 11, 2016, pp. 114–122.
11. C. Metz, “Big Bets on A.I. Open a New Frontier for Chip Start-Ups, Too,” *New York Times*, 2018, p. B1.
12. J. Dean, D. Patterson, and C. Young, “A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution,” *IEEE Micro*, vol. 38, no. 2, 2018.

ABOUT THE AUTHORS

Norman P. Jouppi is a Distinguished Hardware Engineer at Google. He was the tech lead for Google’s first- and second-generation TPUs. He is known for his innovations in computer memory systems, was the principal architect and lead designer of several microprocessors, contributed to the architecture and design of graphics accelerators, and extensively researched telepresence. Jouppi has a PhD from Stanford University and previously studied at Northwestern University. He is a Fellow of the ACM and IEEE, as well as a member of the National Academy of Engineering. Contact him at jouppi@google.com.

Cliff Young is a member of the Google Brain team, where he works on co-design for deep-learning accelerators. He is one of the designers of Google’s TPU, which is used in production applications including Search, Maps, Photos, and Translate. TPUs also powered AlphaGo’s historic 4-1 victory over Go champion Lee Sedol. Before joining Google, Cliff worked at D. E. Shaw Research, building special-purpose supercomputers for molecular dynamics, and at Bell Labs. He has a PhD in computer science from Harvard University. He is a member of the ACM and IEEE. Contact him at cliffy@google.com.

Nishant Patil is a tech lead manager at Google, where he works on system architecture, co-design, and performance optimization for Google TPU and other ML platforms. He has a PhD from Stanford University and previously studied at Carnegie Mellon University. He is a member of the ACM. Contact him at nishantpatil@google.com.

David Patterson is a Distinguished Engineer at Google, a professor emeritus at the University of California, Berkeley, and vice-chair of the RISC-V Foundation board of directors. He is probably best known for the book “Computer Architecture: A Quantitative Approach” and for RISC, RAID, and Network of Workstation (NOW) projects at UC Berkeley. He has a PhD from the University of California, Los Angeles. He is a member of the National Academy of Engineering and the National Academy of Sciences. He recently shared the ACM Turing Award with John Hennessy, his book coauthor. He is a Lifetime Fellow of IEEE, ACM, and both AAAS organizations. Contact him at pattrsn@cs.berkeley.edu.