TPU: In-Datacenter Performance Analysis of a Tensor Processing Unit

---

Norman P. Jouppi, et al.

44th International Symposium on Computer Architecture (ISCA), 2017

Presenter: Constant (Sang-Soo Park)

sonicstage12@naver.com

April 05, 2020

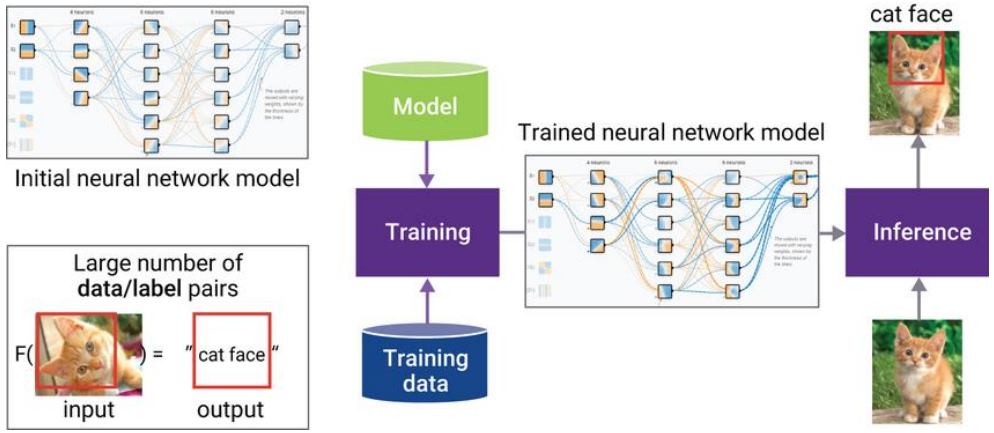
Neural Acceleration Study Season #1

Contents of presentation

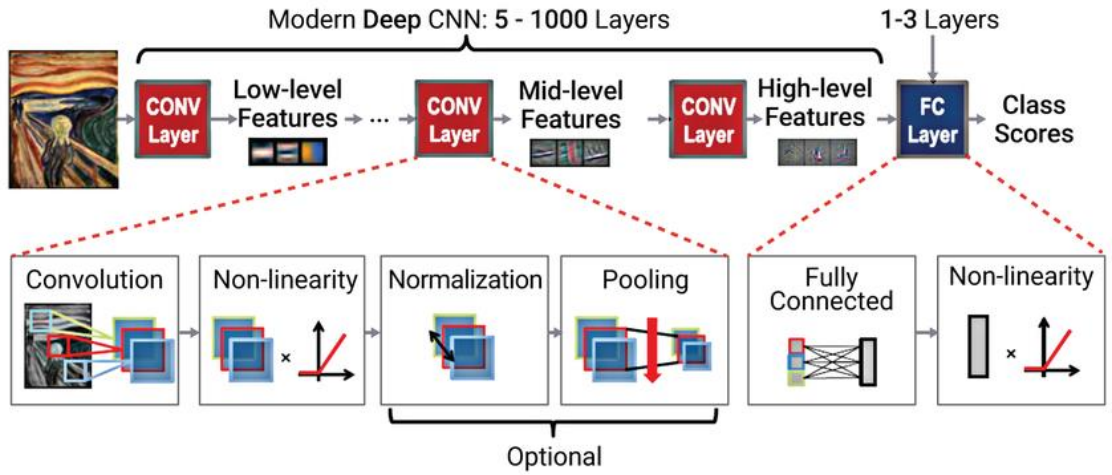
- **Introduction and Background**
  - Fast acceleration
  - Various types accelerators (CPU/GPU, FPGA, and ASIC)
- **Tensor processing unit (TPU) architecture**
  - Partitioned Boolean quadratic assignment problem (PBQP) based selection
- **Performance**
- **Conclusion and Discussion**

Introduction: Fast acceleration

- **Fast inference or training**
  - Inference: Image classification, Acoustic speech recognition
  - Training: ImageNet, TIMIT learning, Etc.
  - The goal of fast acceleration is to classify or learn data quickly
  - Target: Convolution or Fully-connected layer



Neural network models in deep learning framework[1]



Modern convolutional neural networks [1]

[1] https://www.synopsys.com/designware-ip/technical-bulletin/building-efficient-deep-learning-dwtb_q318.html

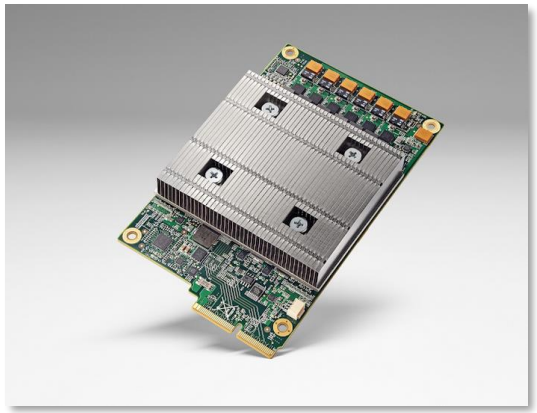
Introduction: Various types accelerators

- **From processor to dedicated accelerator**
  - Processor: CPU (x86, ARM, RISC-V), GPU (NVIDIA, AMD), DSP (CEVA, Qualcomm)
  - Dedicated accelerator: ASIC/FPGA based accelerator (RTL, HLS)

CPU: Ryzen Threadripper

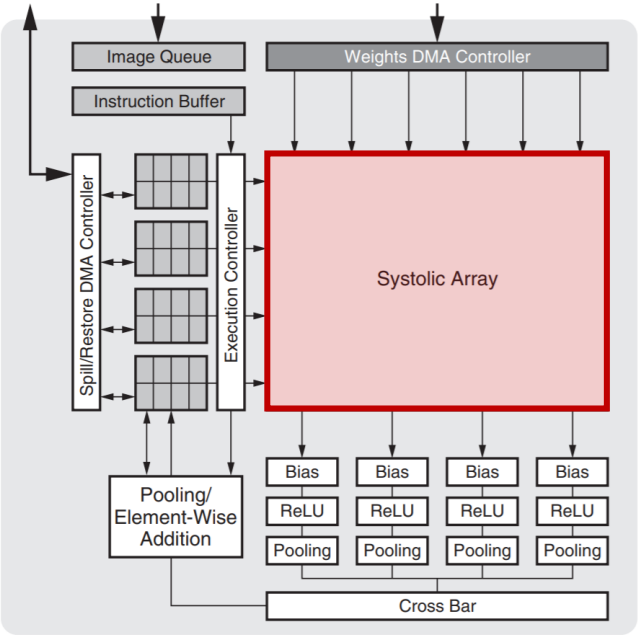
GPU: VOTLA, TURING, PASCAL

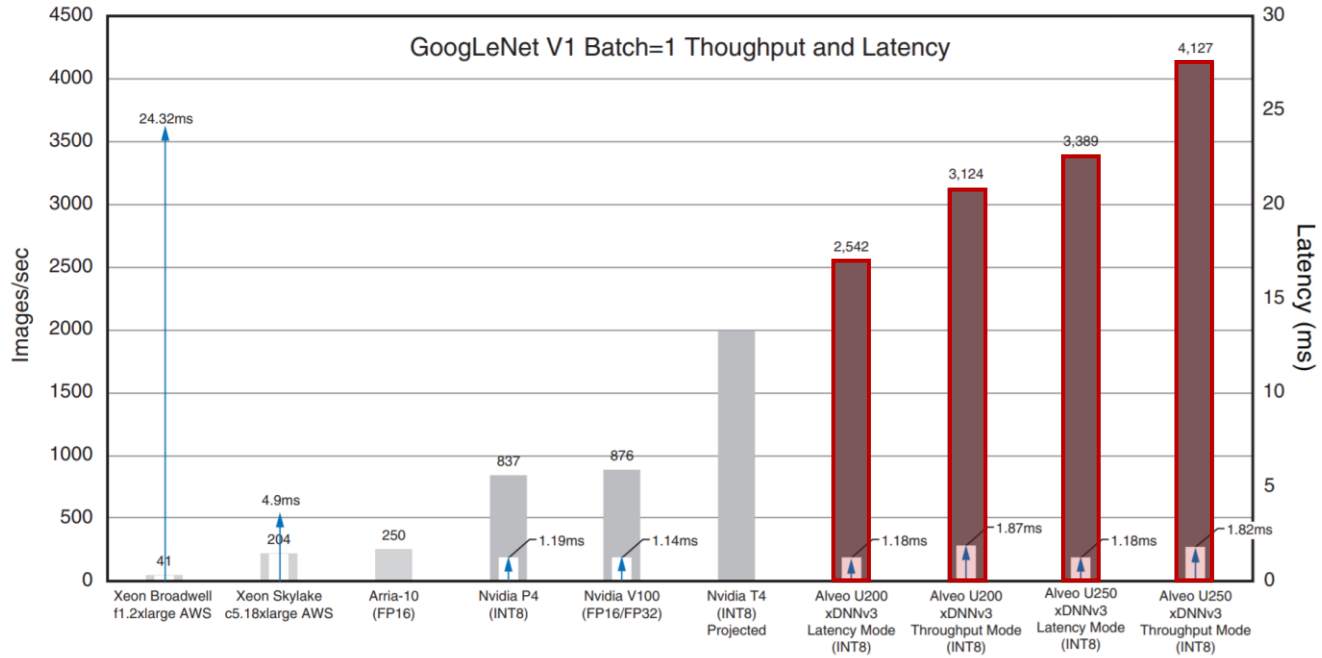
FPGA: ALVEO


ASIC: TPU

Introduction: Various types accelerators

- **Xilinx's ALVEO platform (xDNN hardware architecture)**
  - Systolic array-based FPGA accelerator running on ALVEO platform[2]
  - Latency is slow, but throughput is higher than that of GPU

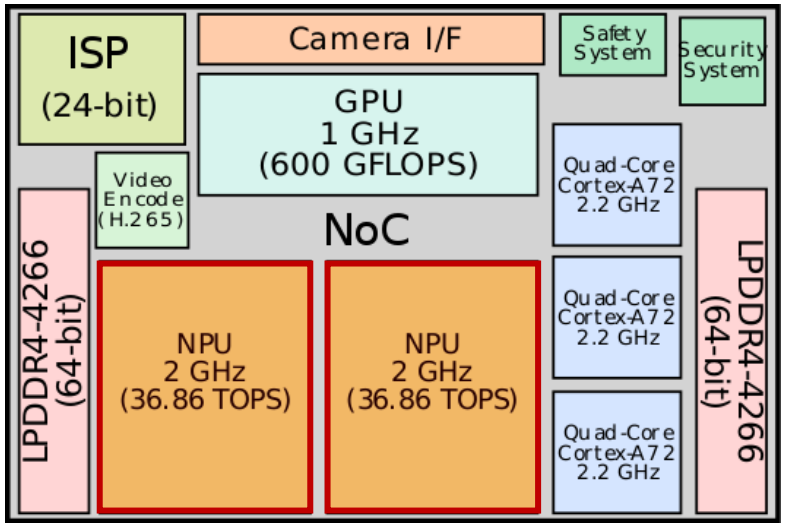


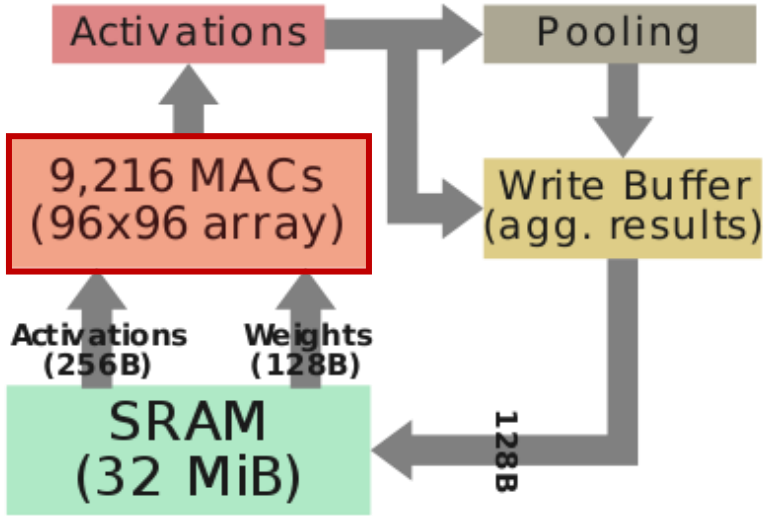


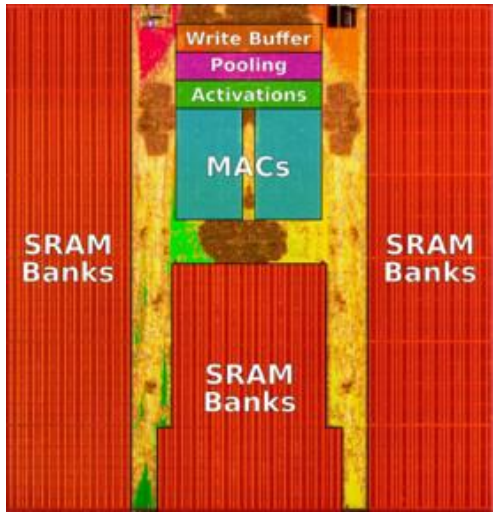
[2] https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Introduction: Various types accelerators

- **Tesla's FSD chip (Full self-driving SoC)**
  - Maybe systolic array-based ASIC accelerator[3]
  - NPU: 2GHz 9,216 MAC operations (8-bit by 8-bit integer multiply and 32-bit integer addition)







[3] https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip

TPU architecture: HW performance

- **TPU v1.0 specification**
  - Build on 28nm technology
  - Thermal design power (75W), Running (40W)
  - Connected to host via PCIe Gen3 x16 bus
  - Up to 4 cards/server

Model	Die										Benchmarked Servers				
	mm^2	nm	MHz	TDP	Measured		TOPS/s		GB/s	On-Chip Memory	Dies	DRAM Size	TDP	Measured	
					Idle	Busy	8b	FP						Idle	Busy
Haswell E5-2699 v3	662	22	2300	145W	41W	145W	2.6	1.3	51	51 MiB	2	256 GiB	504W	159W	455W
NVIDIA K80 (2 dies/card)	561	28	560	150W	25W	98W	--	2.8	160	8 MiB	8	256 GiB (host) + 12 GiB x 8	1838W	357W	991W
TPU	NA*	28	700	75W	28W	40W	92	--	34	28 MiB	4	256 GiB (host) + 8 GiB x 4	861W	290W	384W

Table 2. Benchmarked servers use Haswell CPUs, K80 GPUs, and TPUs. Haswell has 18 cores, and the K80 has 13 SMX processors. Figure 10 has measured power. The low-power TPU allows for better rack-level density than the high-power GPU. The 8 GiB DRAM per TPU is Weight Memory. GPU Boost mode is not used (Sec. 8). SECDEC and no Boost mode reduce K80 bandwidth from 240 to 160. No Boost mode and single die vs. dual die performance reduces K80 peak TOPS from 8.7 to 2.8. (*The TPU die is ≤ half the Haswell die size.)

NA*: die size≤331mm^2

TPU architecture: NN workloads

- **3 kinds of popular neural network**
  - Six NN applications that represent 95% of TPU's workload
  - Convolutional neural network (CNN)
  - Recurrent neural network (RNN): long short-term memory (LSTM)
  - Multi-layer perceptron (MLP)

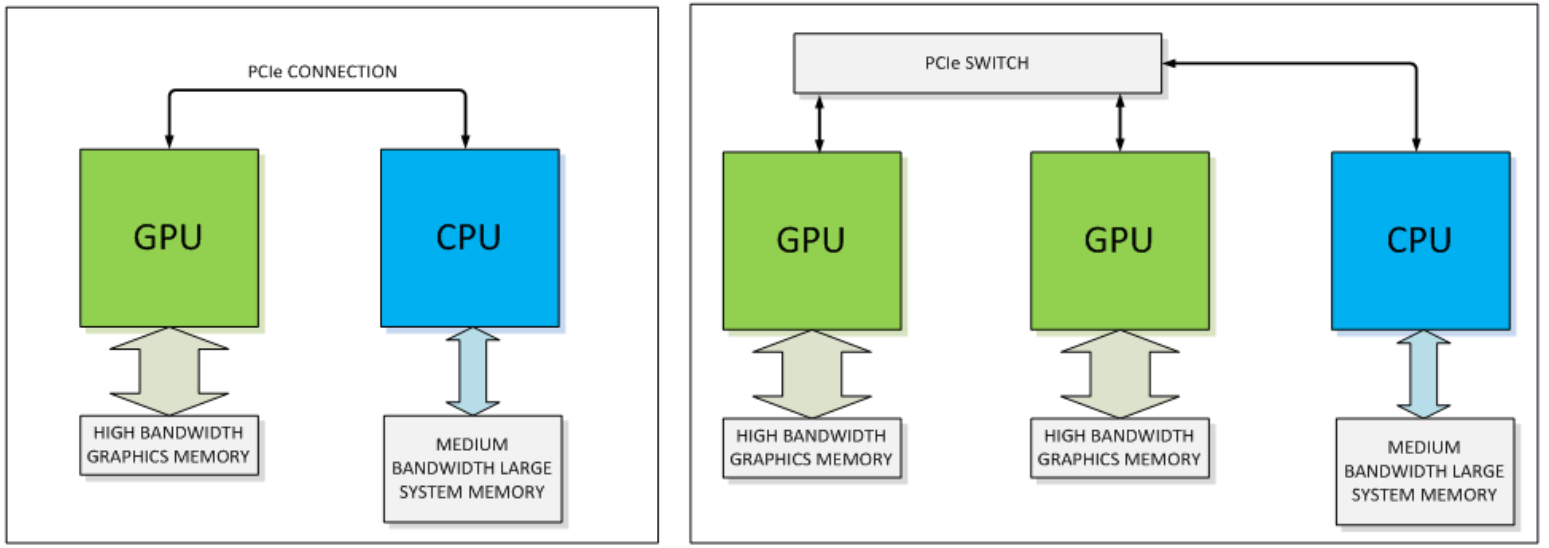
*LOC: # of lines of code

Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

Table 1. Six NN applications (two per NN type) that represent 95% of the TPU's workload. The columns are the NN name; the number of lines of code; the types and number of layers in the NN (FC is fully connected, Conv is convolution, Vector is self-explanatory, Pool is pooling, which does nonlinear downsizing on the TPU; and TPU application popularity in July 2016. One DNN is RankBrain [Cla15]; one LSTM is a subset of GNM Translate [Wu16]; one CNN is Inception; and the other CNN is DeepMind AlphaGo [Sil16][Jou15].

TPU architecture: Implementation

- **Architecture implementation**
  - Coprocessor on PCIe I/O bus like GPU (Reducing the chance of delaying deployment)
  - Host sends TPU instruction for it to execute rather than fetching them itself
  - Goal: to reduce interactions with host CPU and be flexible enough to match NN needs of 2015



CPU-GPU PCIe interconnection[4]

[4] https://devblogs.nvidia.com/nvlink-pascal-stacked-memory-feeding-appetite-big-data/

TPU architecture: Implementation

- **TPU components**
  - Matrix multiply unit: 65,536 (256*256) 8-bit MAC units, 32-bit accumualtors
  - Peak performance: 92TOPS (65,536*2*700M)
  - Weight stationary: weight FIFO (30GiB/s), activation FIFO (167 GiB/s)
  - Sparisty will have high priority in future designs

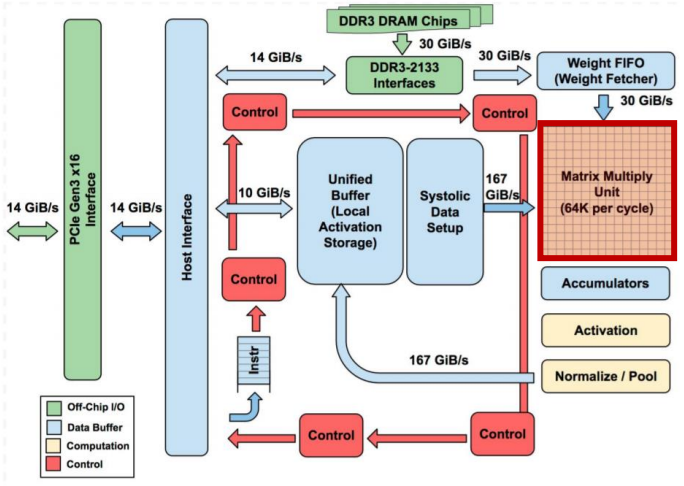


Figure 1. TPU Block Diagram. The main computation part is the yellow Matrix Multiply unit in the upper right hand corner. Its inputs are the blue Weight FIFO and the blue Unified Buffer (UB) and its output is the blue Accumulators (Acc). The yellow Activation Unit performs the nonlinear functions on the Acc, which go to the UB.

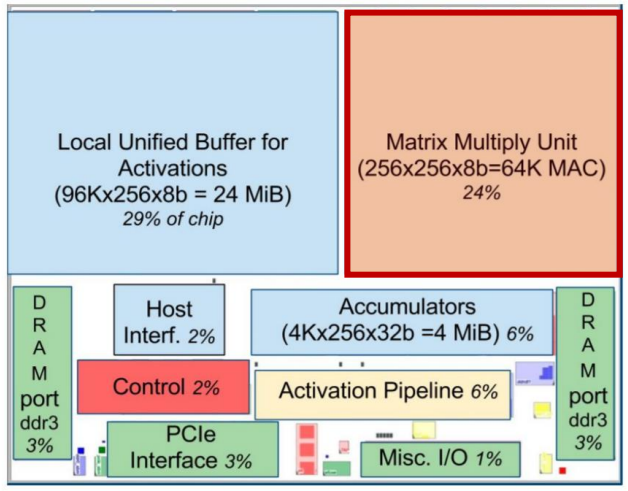


Figure 2. Floor Plan of TPU die. The shading follows Figure 1. The light (blue) data buffers are 37% of the die, the light (yellow) compute is 30%, the medium (green) I/O is 10%, and the dark (red) control is just 2%. Control is much larger (and much more difficult to design) in a CPU or GPU

TPU architecture: Implementation

- **CISC based TPU**
  - As instructions are send over relatively slow PCIe bus, TPU instruction follow CISC tradition.
  - Complex instruction set computer (CISC): complex and variable length instruction
  - Reduced instruction set computer (RISC): simple and fixed length instruction
  - CISC focuses on running more complex task (matrix multiplication) with its instruction

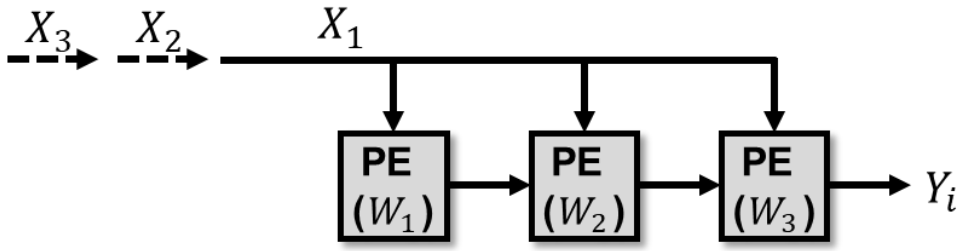
1. `Read_Host_Memory` reads data from the CPU host memory into the Unified Buffer (UB).
2. `Read_Weights` reads weights from Weight Memory into the Weight FIFO as input to the Matrix Unit.
3. `MatrixMultiply/Convolve` causes the Matrix Unit to perform a matrix multiply or a convolution from the Unified Buffer into the Accumulators. A matrix operation takes a variable-sized B*256 input, multiplies it by a 256x256 constant weight input, and produces a B*256 output, taking B pipelined cycles to complete.
4. `Activate` performs the nonlinear function of the artificial neuron, with options for ReLU, Sigmoid, and so on. Its inputs are the Accumulators, and its output is the Unified Buffer. It can also perform the pooling operations needed for convolutions using the dedicated hardware on the die, as it is connected to nonlinear function logic.
5. `Write_Host_Memory` writes data from the Unified Buffer into the CPU host memory.

TPU architecture: Implementation

- **TPU instruction (12B)**
  - Unfired buffer address (3B), Accumulator address (2B), Length (4B), Opcode and Flag (3B)
  - Average clock cycles per instruction > 10
  - Philosophy of TPU architecture is to keep matrix unit busy (4-stage CISC instruction)
    (Execute other instructions while matrix multiplier busy)

TPU architecture: Implementation

- **Systolic array (WS)**
  - PE holds weights, multiplies it with its input and add partial sums
  - Weight stationary as weight stay stationary and the input are streamed in



$t = 0$	$W_1 * X_1$	$W_2 * X_1$	$W_3 * X_1$	$W_3 * X_1$
$t = 1$	$W_1 * X_2$	$W_2 * X_2$ $+ W_1 * X_1$	$W_3 * X_2$ $+ W_2 * X_1$	$W_3 * X_2$ $+W_2 * X_1$
$t = 2$	$W_1 * X_3$	$W_2 * X_3$ $+ W_1 * X_2$	$W_3 * X_2$ $+ W_2 * X_1$ $+ W_1 * X_1$	$W_3 * X_2$ $+W_2 * X_1$ $+ W_1 * X_1$

TPU architecture: Evaluation

- **Roofline model comparison (TPU)**
  - Y-axis (computing performance), X-axis (arithmetic intensity, operations per DRAM byte accessed)
  - Five of the six applications are happily bumping their heads against the ceiling
  - MLPs and LSTMs are memory bound, and CNNs are computation bound
  - CNN1 (very high operational intensity, 14.1 TOPS), CNN0 (86TOPS)

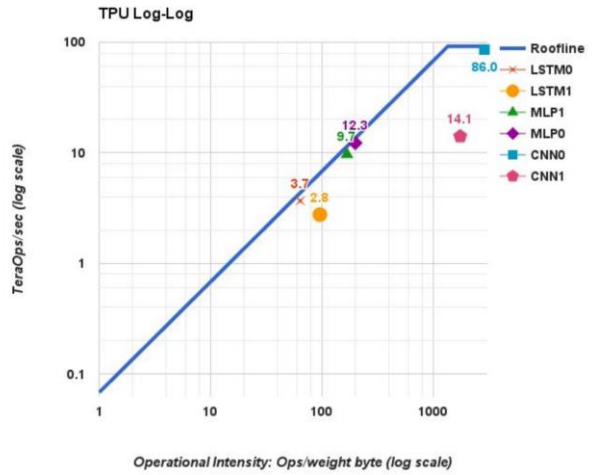


Figure 5. TPU (die) roofline. Its ridge point is far to the right at 1350 operations per byte of weight memory fetched.

Application	*MLP0*	*MLP1*	*LSTM0*	*LSTM1*	*CNN0*	*CNN1*	*Mean*	*Row*
Array active cycles	12.7%	10.6%	8.2%	10.5%	78.2%	46.2%	28%	1
Useful MACs in 64K matrix (% peak)	12.5%	9.4%	8.2%	6.3%	78.2%	22.5%	23%	2
Unused MACs	0.3%	1.2%	0.0%	4.2%	0.0%	23.7%	5%	3
Weight stall cycles	53.9%	44.2%	58.1%	62.1%	0.0%	28.1%	43%	4
Weight shift cycles	15.9%	13.4%	15.8%	17.1%	0.0%	7.0%	12%	5
Non-matrix cycles	17.5%	31.9%	17.9%	10.3%	21.8%	18.7%	20%	6
RAW stalls	3.3%	8.4%	14.6%	10.6%	3.5%	22.8%	11%	7
Input data stalls	6.1%	8.8%	5.1%	2.4%	3.4%	0.6%	4%	8
TeraOps/sec (92 Peak)	12.3	9.7	3.7	2.8	86.0	14.1	21.4	9

Table 3. Factors limiting TPU performance of the NN workload based on hardware performance counters. Rows 1, 4, 5, and 6 total 100% and are based on measurements of activity of the matrix unit. Rows 2 and 3 further break down the fraction of 64K weights in the matrix unit that hold useful weights on active cycles. Our counters cannot exactly explain the time when the matrix unit is idle in row 6; rows 7 and 8 show counters for two possible reasons, including RAW pipeline hazards and PCIe input stalls. Row 9 (TOPS) is based on measurements of production code while the other rows are based on performance-counter measurements, so they are not perfectly consistent. Host server overhead is excluded here. The MLPs and LSTMs are memory-bandwidth limited but CNNs are not. CNN1 results are explained in the text.

TPU architecture: Evaluation

- **Roofline model comparison (CPU and GPU)**
  - Six NN applications are generally below their ceilings than that of TPU

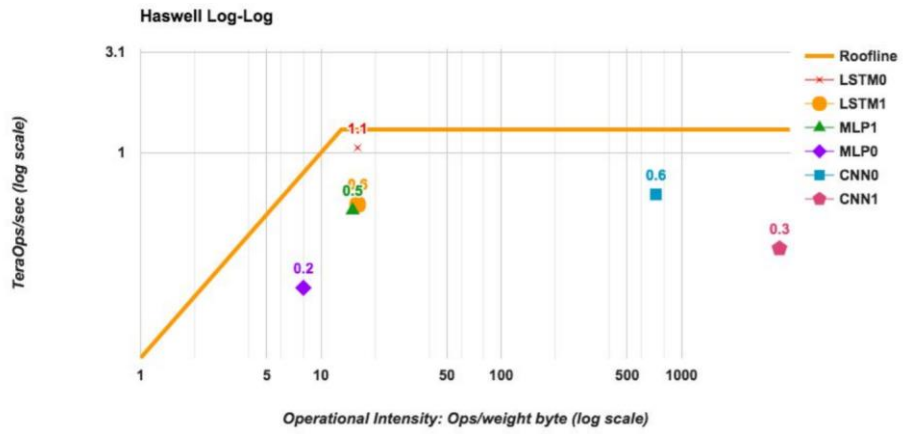


Figure 6. Intel Haswell CPU (die) roofline with its ridge point at 13 operations/byte, which is much further left than in Figure. 5. LSTM0 and MLP1 are faster on Haswell than on the K80, but it is vice versa for the other DNNs.

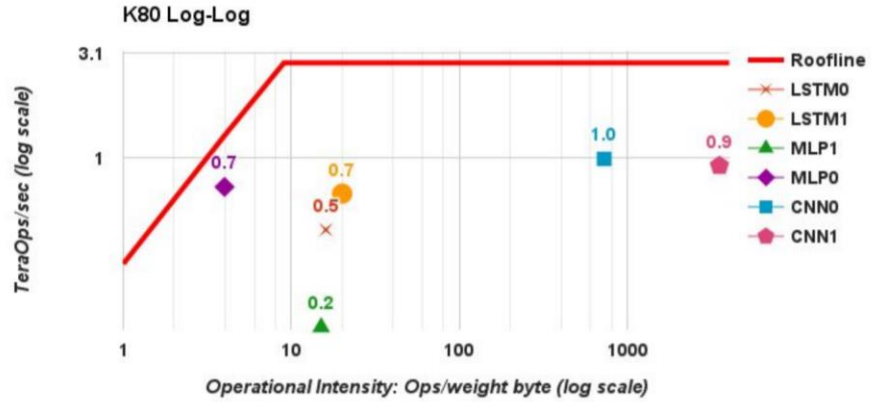


Figure 7. NVIDIA K80 GPU die Roofline. The much higher memory bandwidth moves the ridge point to 9 operations per weight byte, which is even further left than in Figure 6. The DNNs are much lower than their Roofline because of response time limits (see Table 4).

TPU architecture: Evaluation

- **Execution performance (CPU,GPU, and TPU)**
  - Q: Why below their ceilings?
    A: Response time (small increase in response time cause customer to use service less)
  - Inference prefers latency over throughput

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

Table 4. 99-th% response time and per die throughput (IPS) for MLP0 as batch size varies for MLP0. The longest allowable latency is 7 ms. For the GPU and TPU, the maximum MLP0 throughput is limited by the host server overhead. Larger batch sizes increase throughput, but as the text explains, their longer response times exceed the limit, so CPUs and GPUs must use less-efficient, smaller batch sizes (16 vs. 200).

TPU architecture: Evaluation

- **Computing per watt (Cost)**
  - TPU' (improved TPU, GDDR5 like in K80, BW:34GB/s->180GB/s)
  - Total perf./watt includes CPU and GPU power consumption for server computing
  - Incremental perf./watt doesn't include CPU and GPU power consumption for server computing
  - In incremental, TPU' perf./watt by GM is 17, 34 times better than CPU and is 14,16 times better than GPU

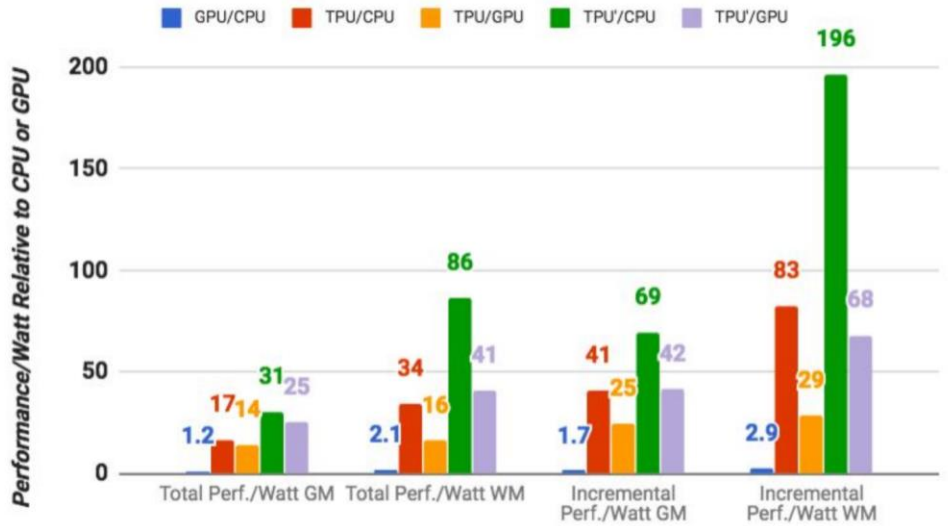


Figure 9. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU (Sec. 7). The green bar shows its ratio to the CPU server and the lavender bar shows its relation to the GPU server. Total includes host server power, but incremental doesn't. GM and WM are the geometric and weighted means.

TPU architecture: Evaluation

- **Alternative TPU designs**
  - Memory bandwidth is most important element for performance
  - Most NN benchmark applications cause performance bottlenecks due to memory bounds

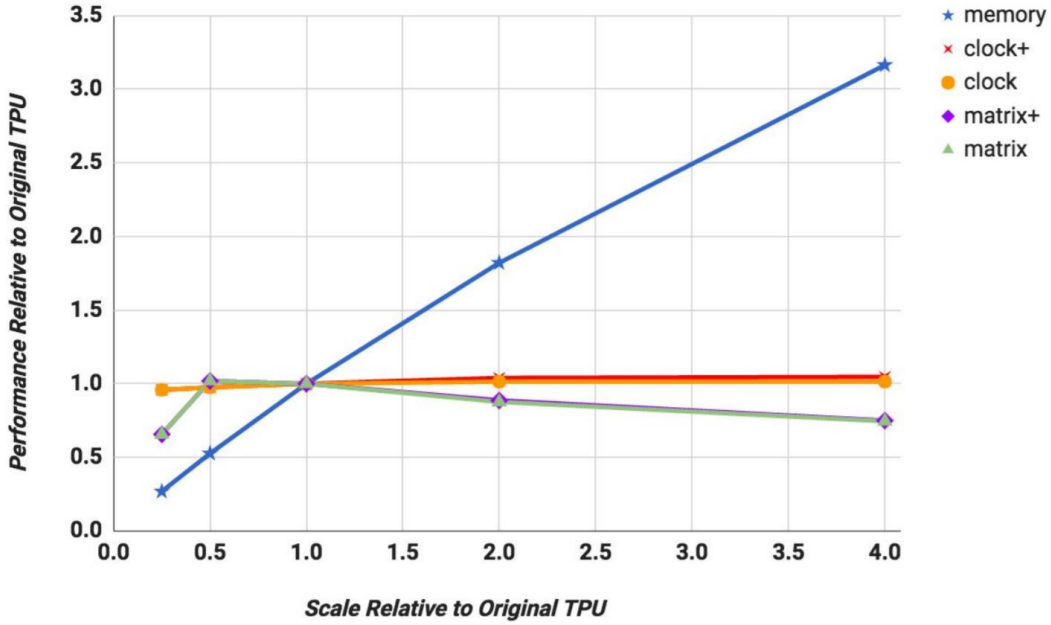


Figure 11. Weighted mean TPU performance as metrics scale from 0.25x to 4x: memory bandwidth, clock rate + accumulators, clock rate, matrix unit dimension + accumulators, and matrix unit dimension. The weighted mean makes it hard see to contributions of individual DNNs, but MLPs and LSTMs improve 3X with 4X memory bandwidth, but get nothing from a higher clock. For CNNs it's vice versa; 2X for 4X clock, but get little benefit from faster memory. A bigger matrix multiply unit doesn't help any DNN.

Discussion: Fallacy

- *Fallacy: NN 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 of them 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 actually 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 small batch sizes result in higher performance than contemporary CPUs and GPUs.

- *Fallacy: After two years of software tuning, the only path left to increase TPU performance is hardware upgrades.*

The performance of CNN1 on the TPU could improve if developers and compiler writers did more work to match CNN1 to the TPU hardware. For example, developers could reorganize the applications to aggregate multiple short batches out of the convolution layers into a single, deeper batch (from 32 to 128) for the four fully connected layers. Such a single layer would improve utilization of the matrix unit (Table 3). As CNN1 currently runs more than 70 times faster on the TPU than the CPU, the CNN1 developers are already very happy, so it's not clear whether or when such optimizations would be performed.

Discussion: Pitfall

- *Pitfall: For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric.*

Our results show that IPS is a poor overall performance summary for NN hardware, as it's simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers). For example, the TPU runs the 4-layer MLP1 at 360,000 IPS but the 89-layer CNN1 at only 4,700 IPS, so TPU IPS vary by 75X! Thus, using IPS as the single-speed summary is *even more misleading* for NN accelerators than MIPS or FLOPS are for regular processors [Hen18], so IPS should be even more disparaged. To compare NN machines better, we need a benchmark suite written at a high-level to port it to the wide variety of NN architectures. Fathom is a promising new attempt at such a benchmark suite [Ado16].

- *Pitfall: Architects have neglected important NN tasks.*

We are pleased by the attention that the architecture community is paying to NN: 15% of the papers at ISCA 2016 were on hardware accelerators for NN [Alb16] [Che16a][Chi16][Han16][Kim16][LiK16][Liu16][Rea16] [Sha16]! Alas, all nine papers looked at CNNs, and only two mentioned other NNs. CNNs are more complex than MLPs and prominent in NN competitions [Rus15], which might explain their allure, but they are only about 5% of our datacenter NN workload. While CNNs may be common in edge devices, the volume of convolutional models hasn't yet caught up with MLPs and LSTMs in the datacenter. We hope that architects try to accelerate MLPs and LSTMs with at least as much gusto.

Thank you