In-Datacenter Performance Analysis of Tensor Processing Unit

Draft Paper Overview

Overview

- Motivation
- Purpose of the paper
- Summary of neural networks
- Overview of the proposed architecture
- Results and comparison between TPU, CPU & GPU

Motivation

- 2006: only a few applications could run on custom H/W (ASIC, FPGA, GPUs) —> thus you could easily find those resources in a datacenter (due to under-utilization)

- 2013: predictions on the wide applicability of another computational paradigm called Neural Networks (NN), could double the computation demands on datacenters.

    - It would be very expensive to increase the GPUs in order to satisfy those needs

Tensor Processing Unit (TPU)

- Proposal: Design a custom ASIC for the inference phase of NN (training still happens using GPUs)
- Principles:
  - improve cost-performance by 10X compared to GPUs
  - simple design for response time guarantees (single-thread, no prefetching, no OOO etc)
- Characteristics:
  - More like a co-processor to reduce time-to-market delays
    - Host sends instructions to TPU
  - connected through PCIe I/O bus

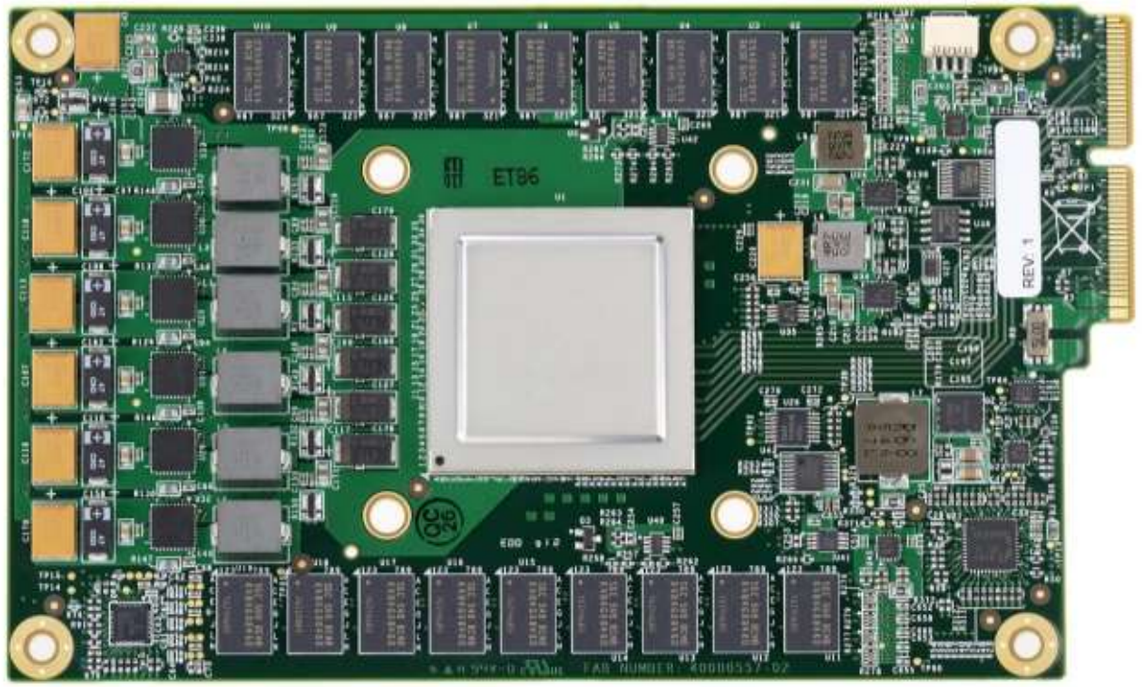
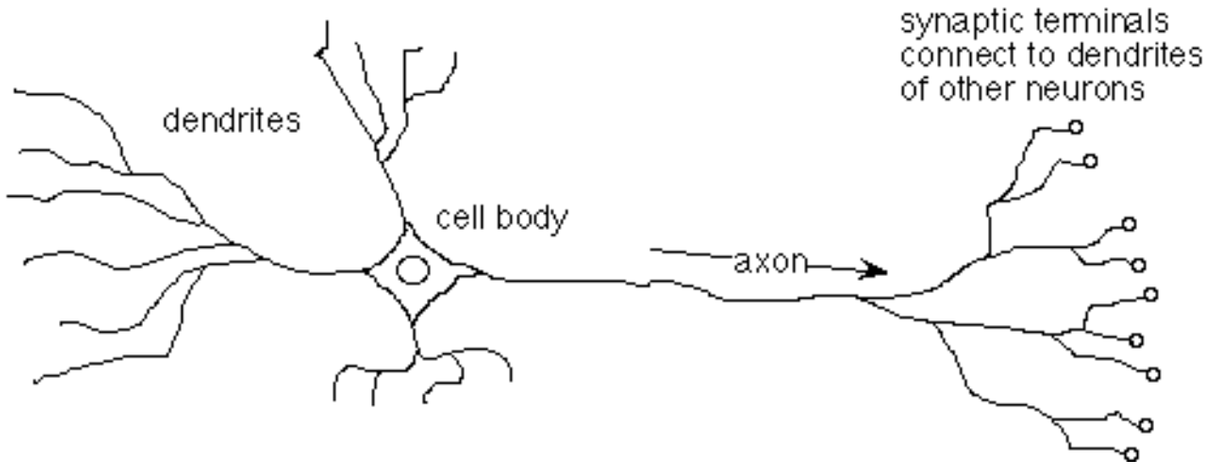


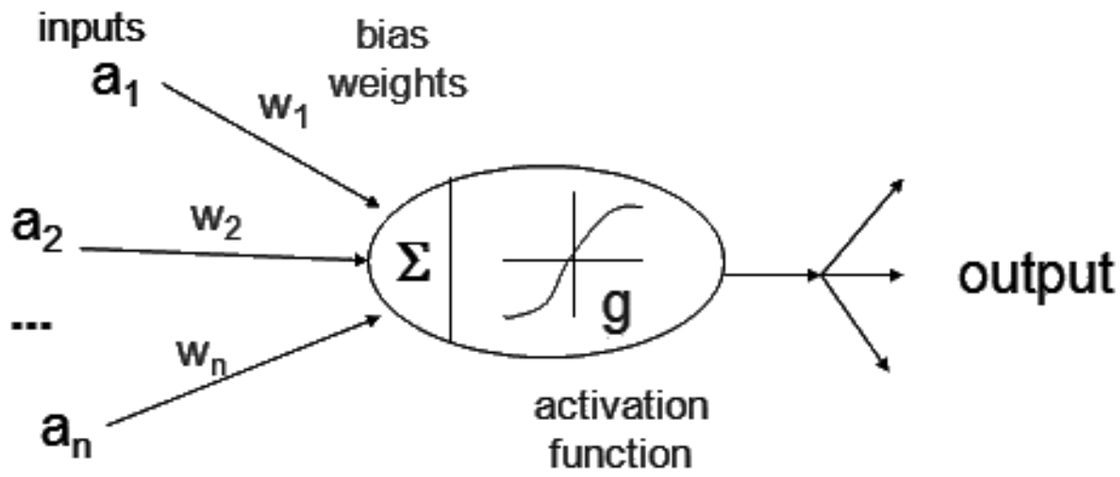
Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

Neural Network

- Artificial NN approximately simulate the functionality of the brain
- Breakthroughs (of “Deep” NNs):
  - Beat human champion at Go
- Decreasing the error in
  - *image recognition* from 26 to 3.5%
  - *speech recognition* by 30% over other approaches

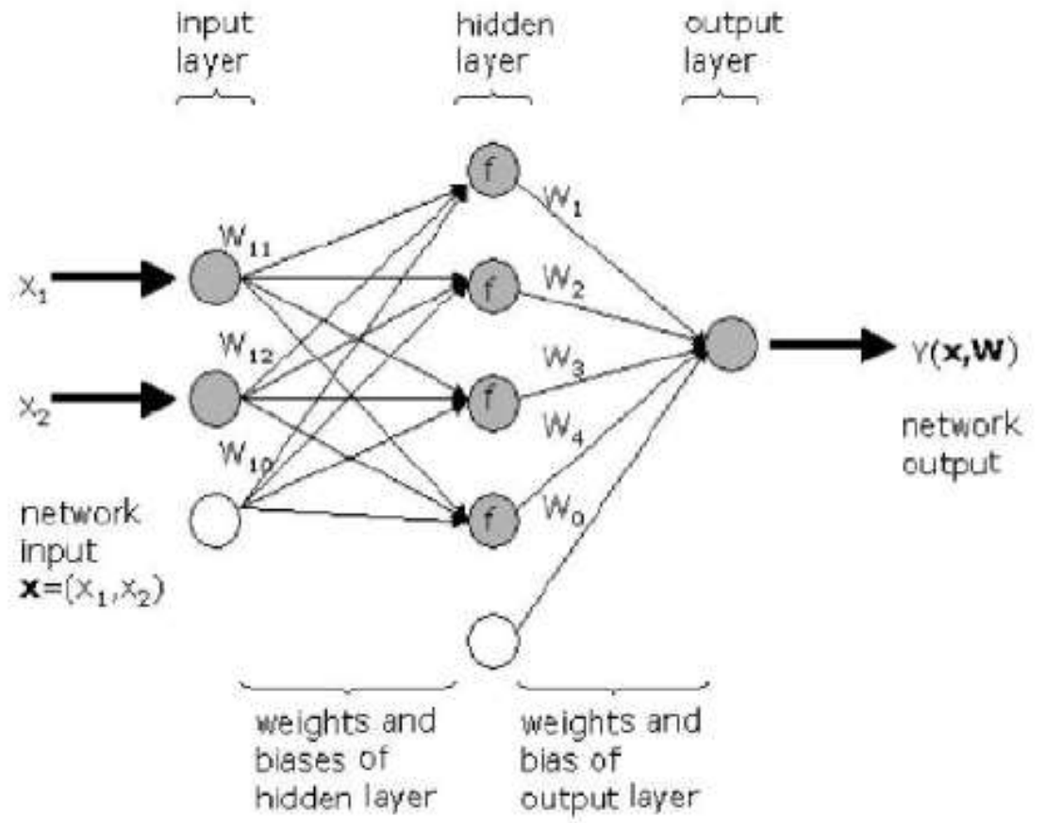


Natural VS artificial neuron



Neural Networks Structure

- Artificial neurons are divided into layers
- Three kind of layers:
  - Input Layer
  - Hidden Layer(s)
  - Output Layer
- “Deep” NN refers to many layers
  (higher level of patterns —> better accuracy)
- Results may require much computation but:
  - parallelism can be extracted
    in-between layers and
  - all of these computation follow
    the pattern of multiply and add



a_i^j = activation of unit ‘i’ in layer ‘j’

θ^j = matrix of weights controlling function mapping from layer j to j+1

$$a_1^2 = g(\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$$

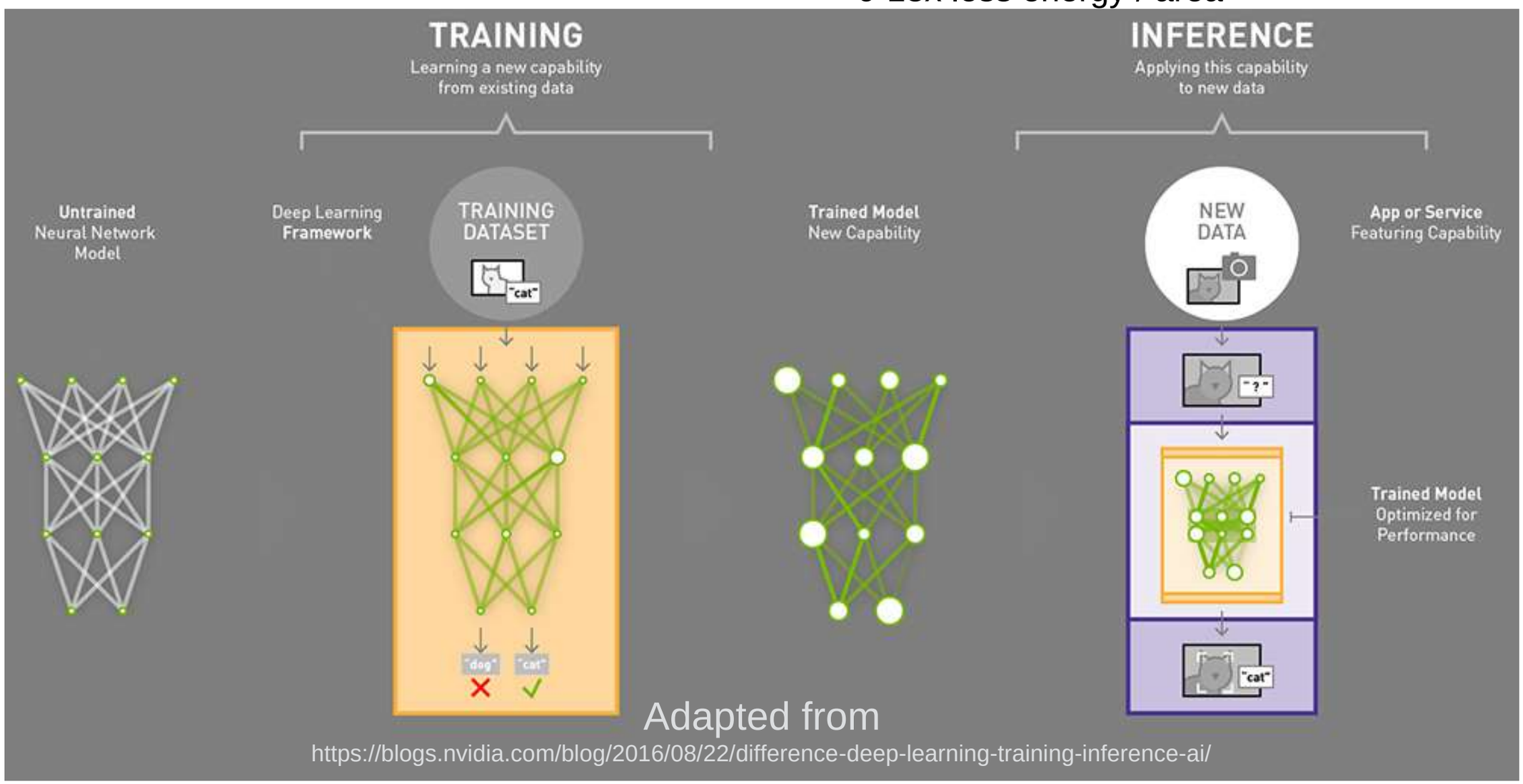
$$a_2^2 = g(\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$$

$$a_3^2 = g(\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$$

$$h_\theta(x) = a_1^3 = g(\theta_{10}^2 a_0^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2)$$

Neural Network Phases

- Two phases of NN:
  - Training (calculation of weights)
    - Usually floating point operations
  - Inference (prediction)
    - You can quantize (transform floating points to 8-bit integers)
      - Trade accuracy (addition-multiplication):
        - 6-13x less energy / area

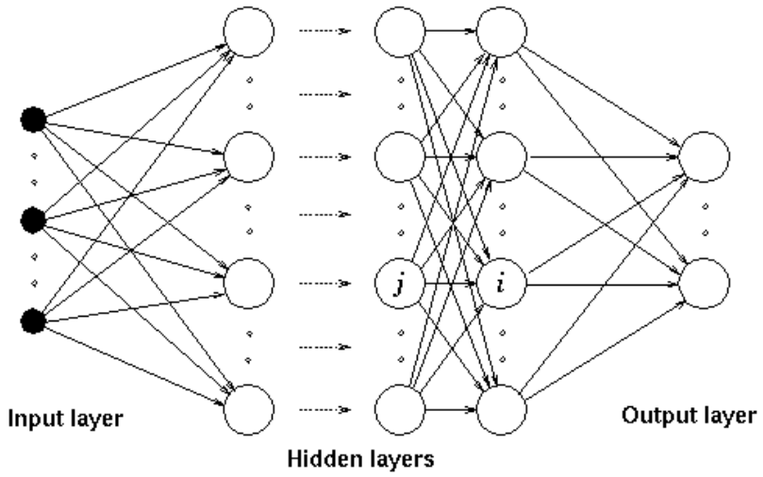


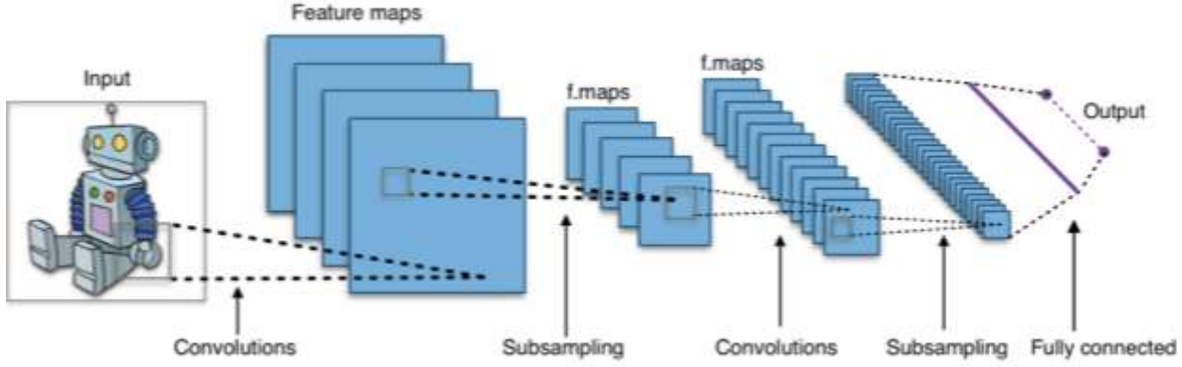
Adapted from

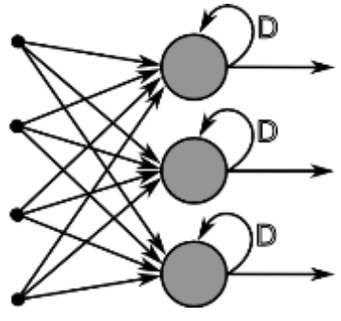
https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/

Popular NN Architectures

- 3 Kinds of NNs are popular today:
  - Multi-Layer Perceptrons (MLP): (fully connected) output of each layer feed into every neuron on the next layer
  - Convolutional Neural Networks (CNN): each neuron get as input the results of spatial nearby output of the previous layer
  - Recurent Neural Networks (RNN): some outputs of the previous layer and its previous state







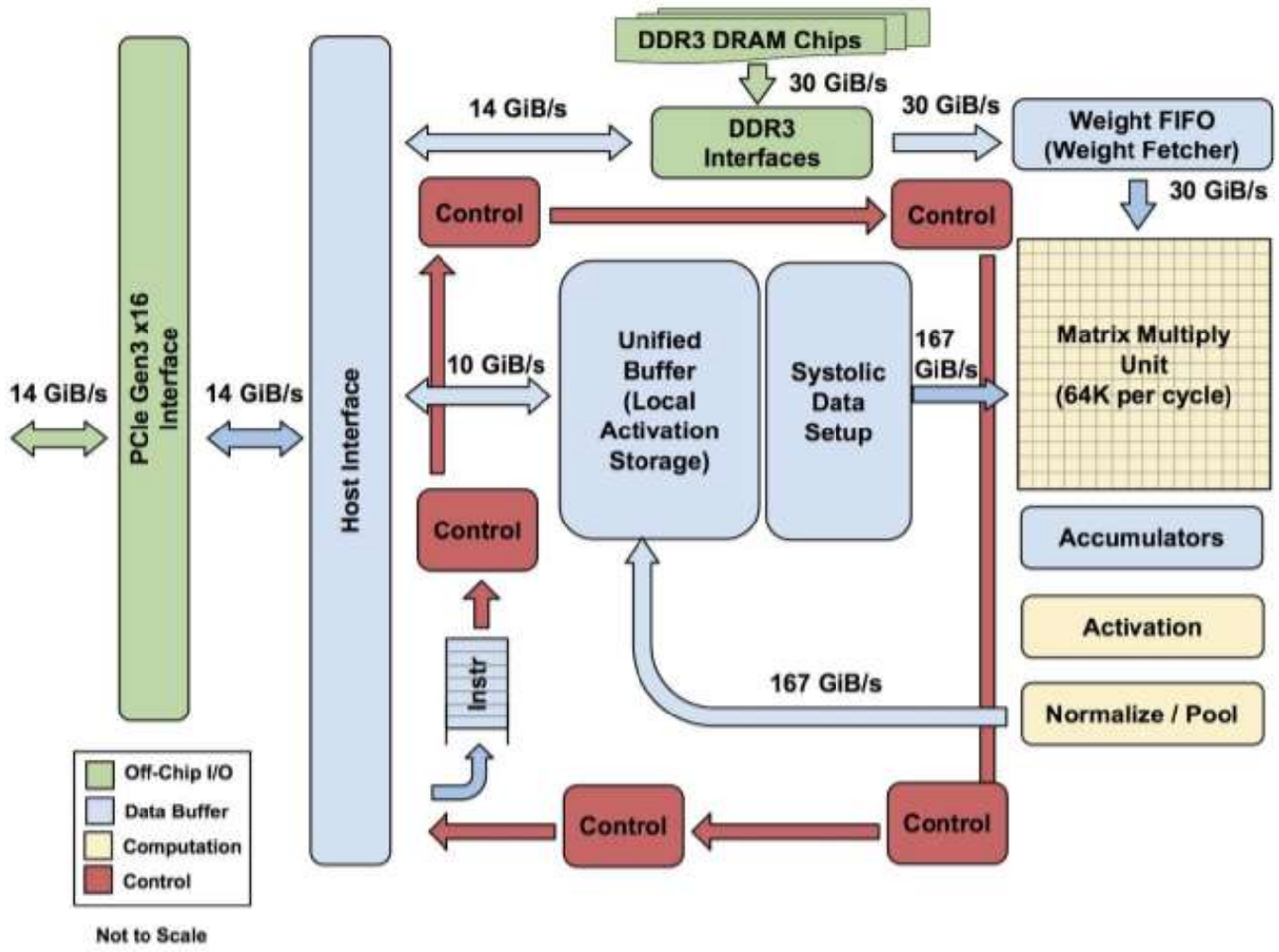
Paper Synopsis

- Description and Evaluation of the Tensor Processing Unit (TPU)
- By comparing power and performance between the TPU, a GPU and a CPU
- Using 6 representative NN applications (see table below).

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	

TPU Block Diagram

- The Matrix Multiply Unit (MMU) is the TPU's heart
  - contains 256 x 256 MACs
- Weight FIFO (4 x 64KB tiles deep) uses 8GB off-chip DRAM to provide weights to the MMU
- Unified Buffer (24 MB) keeps activation input/output of the MMU & host
- Accumulators: (4MB = 4096 x 256 x 32bit) collect the16 bit MMU products
  - 4096 (1350 ops/per byte to reach peak performance ~= 2048 x2 for double buffering)



Matrix Multiply Unit

- MMU uses a Systolic execution
- Using 256x256 MACs that perform 8-bit integer multiply & add (enough for results)
- Holds 64KB tile of weights + 1 more tile (hide 256 cycles that need to shift one tile in)
    - less SRAM accesses
    - lower power consumption
    - higher performance
- MatrixMultiply(B) A matrix instruction 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.

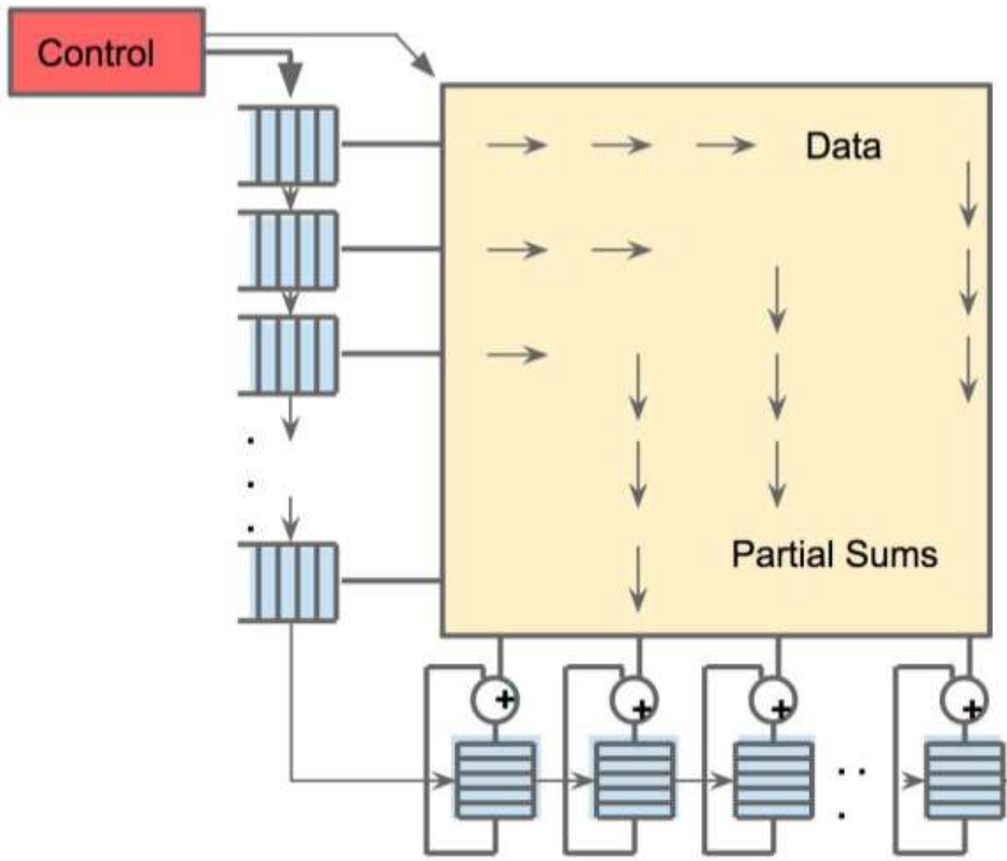
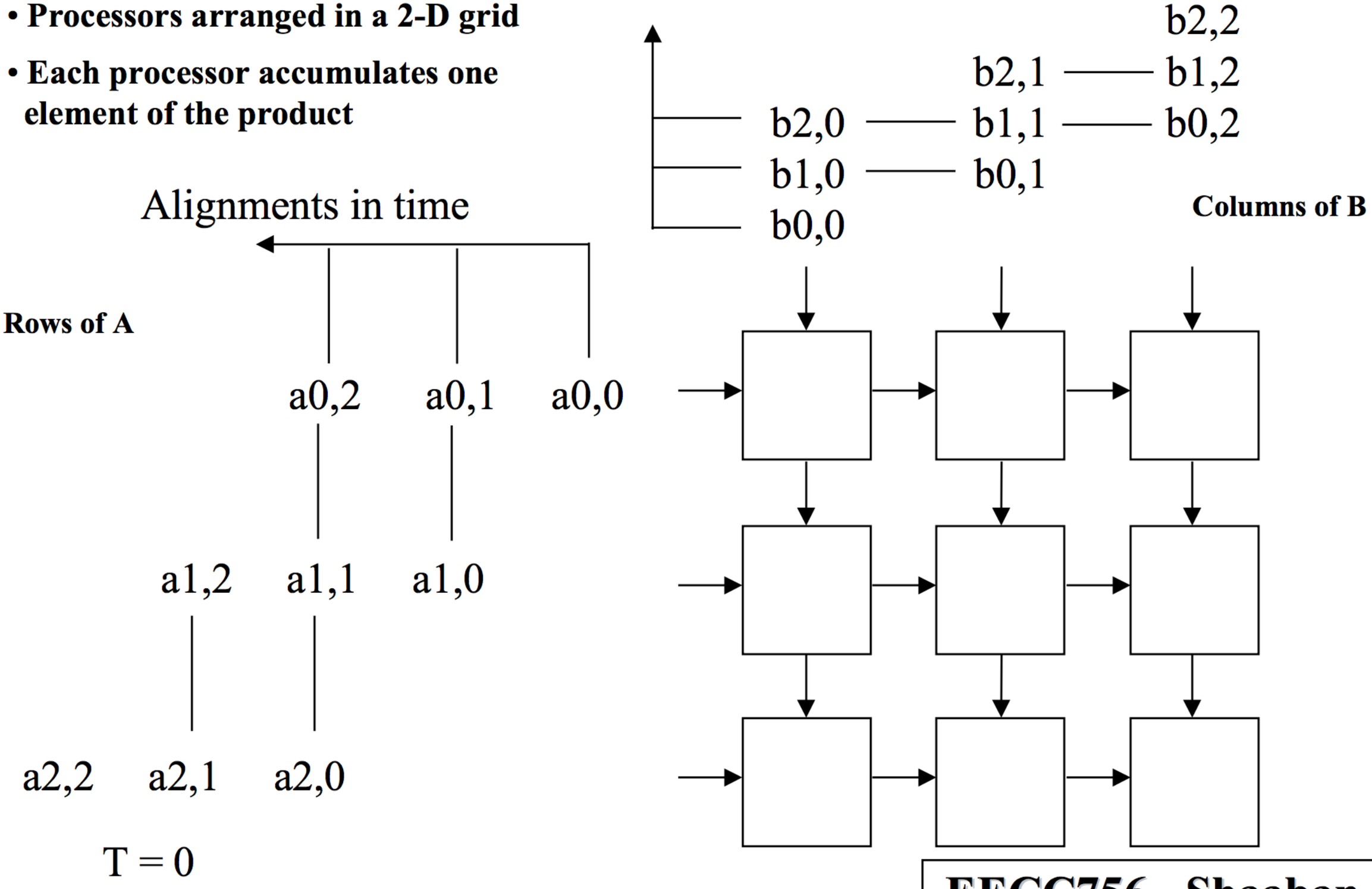


Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

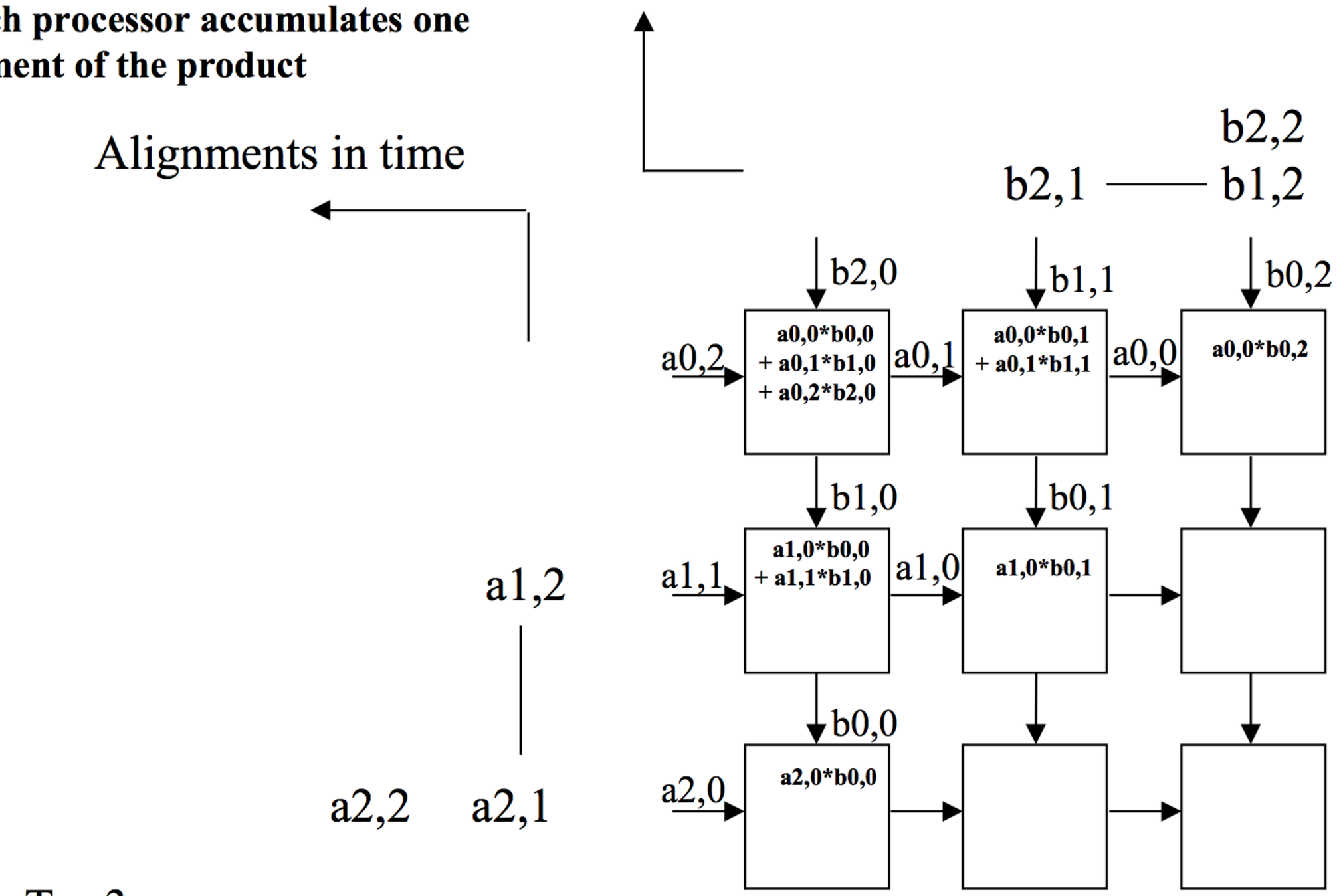
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Adapted from [Matrix-Multiplication-systolic.pdf]

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Adapted from [Matrix-Multiplication-systolic.pdf]

Evaluation

- Haswell CPU: 18-cores (no turbo, 2.3GHz with AVX)
- K80 GPU: Boost mode of is disabled (enabling it would increases the total cost of ownership, since due to cooling demands they should deploy less GPUs than 8)
- Roofline model adapted (from HPC)
  (assuming the application data does not fit in on-chip memory then you are either memory BW bottlenecked or computation-limited)
- Results are normalised per die

Model	*Die*										*Benchmarked Servers*				
	mm²	*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.

TPU micro-bench results

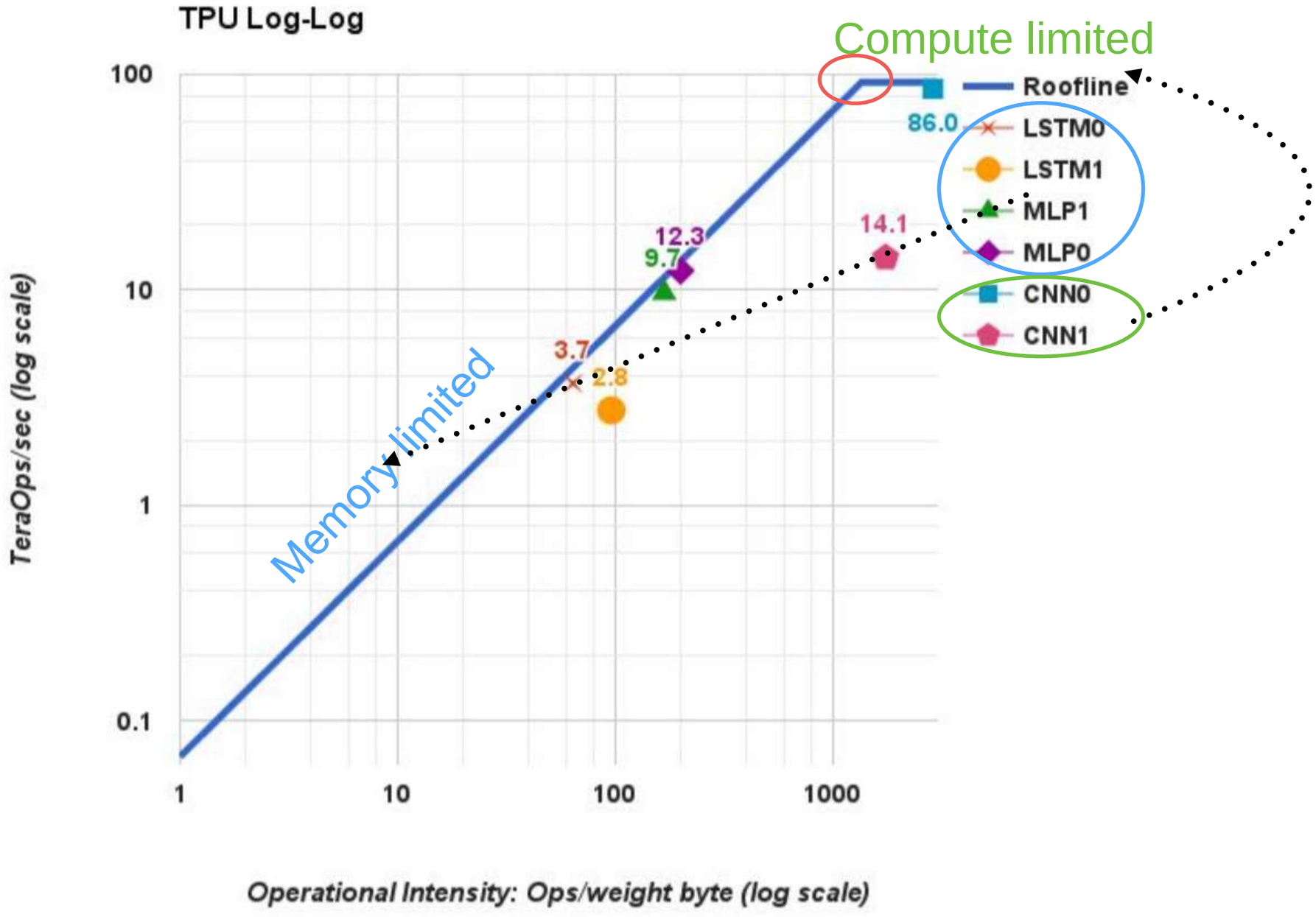


Figure 5. TPU (die) roofline. Its ridge point is far to the right at 1350 operations per byte of weight memory fetched.

- Without sufficient operational intensity, a program is memory bandwidth-bound and lives under the slanted part of the roofline.

TPU Micro-bench Breakdown

- The table below doesn't account for host server time, which can be divided into:
  - running the host share of the application
  - and talking to the TPU.

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

Stalls due 2 Memory

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.

Low utilization

CPU & GPU micro-bench

- Inference, in contrast with training, prefers latency over throughput.
(end-user facing services)

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.

- Response time guarantees is the reason that the applications are generally further below their ceilings than was in the TPU.

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).

Minimalism is a virtue of domain-specific processors

- CPU & GPU have higher throughput, which is wasted if they don’t meet the response time limits
- TPU is also affected by response time but only at 80%
- The single-threaded TPU has none of the sophisticated micro-architectural 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, multithreading, context switching, and so forth.)

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).

Energy Proportionality

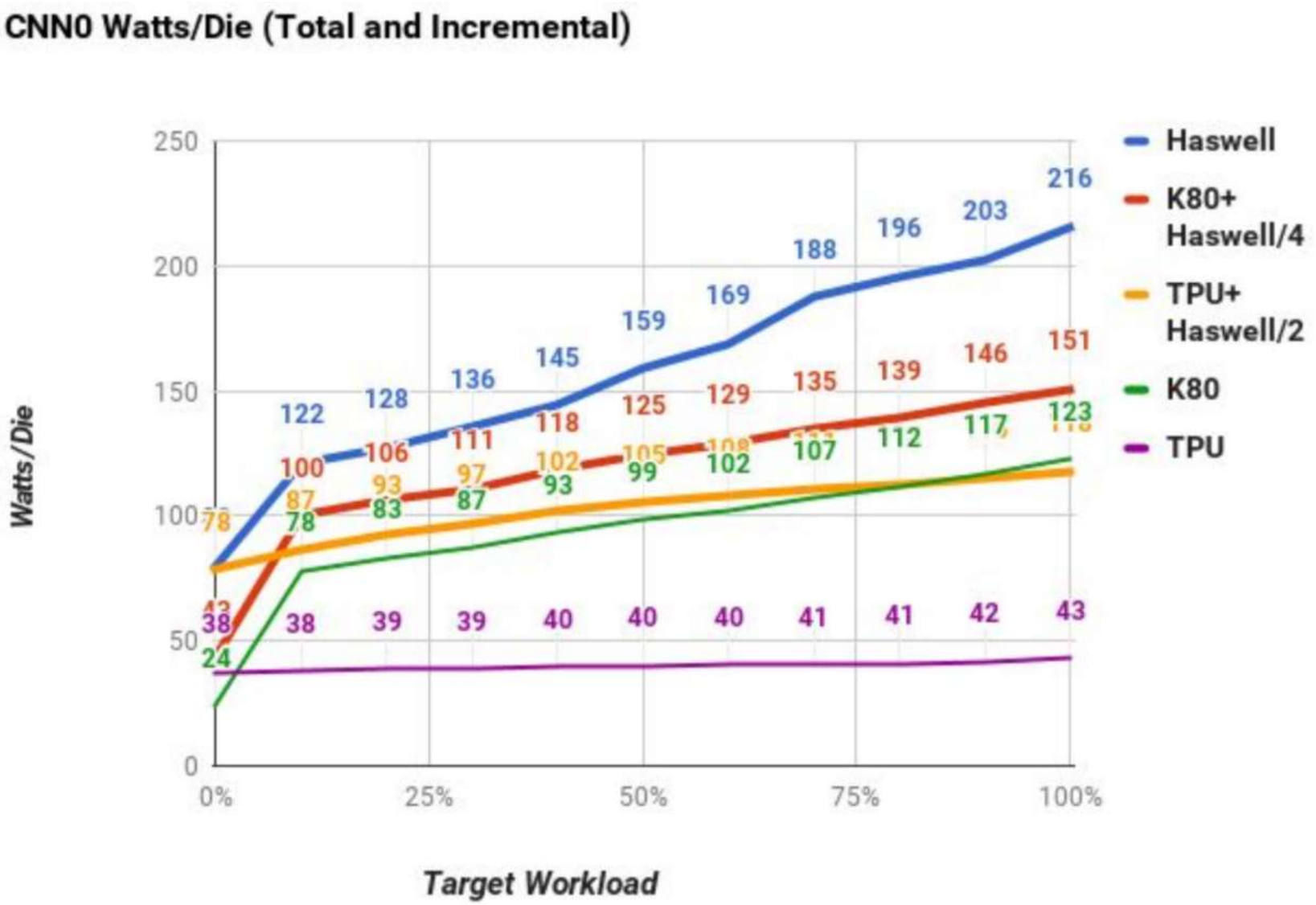


Figure 10. Watts/die for CNN0 as target platform utilization varies from 0% to 100%. The Total GPU and TPU power are the red and orange lines and their Incremental power are the green and purple lines. (The blue line is Haswell, which by definition is total power.) A server has 2 CPUs and 8 GPUs or 4 TPUs, so we normalize power by dividing by 2, 8, and 4, respectively.

Evaluation of Alternative TPU Design

- The graph below is created by a performance model of the TPU for the 6 NN apps which is about 10% accurate compared to the existing TPU results.

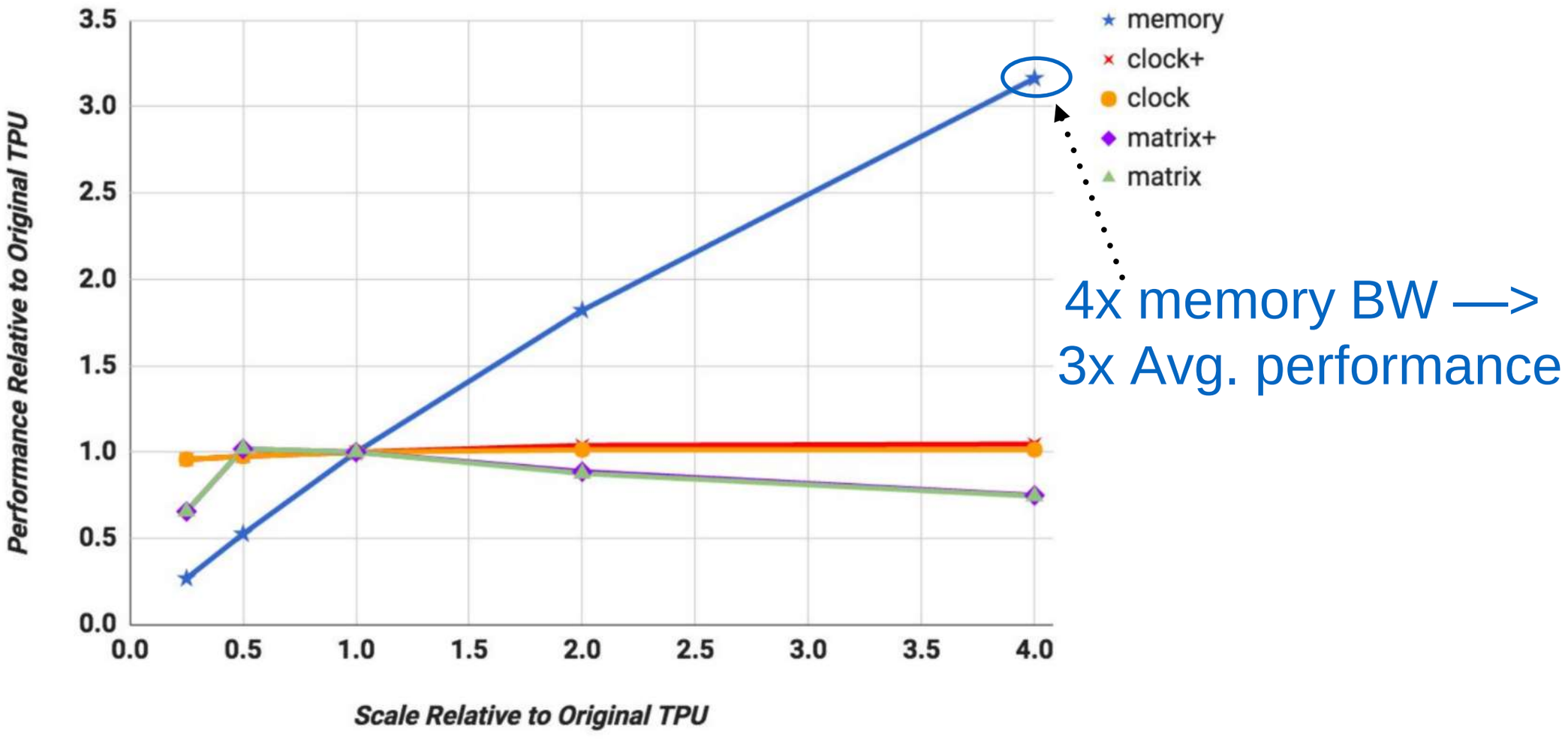


Figure 11. Weighted mean TPU performance as metrics scale from 0.25x to 4x: memory bandwidth, clock rate + accumulators, clock rate, matrix unit dimension + accumulators, and matrix unit dimension. The weighted mean makes it hard see to contributions of individual DNNs, but MLPs and LSTMs improve 3X 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.

Cost-Performance Results

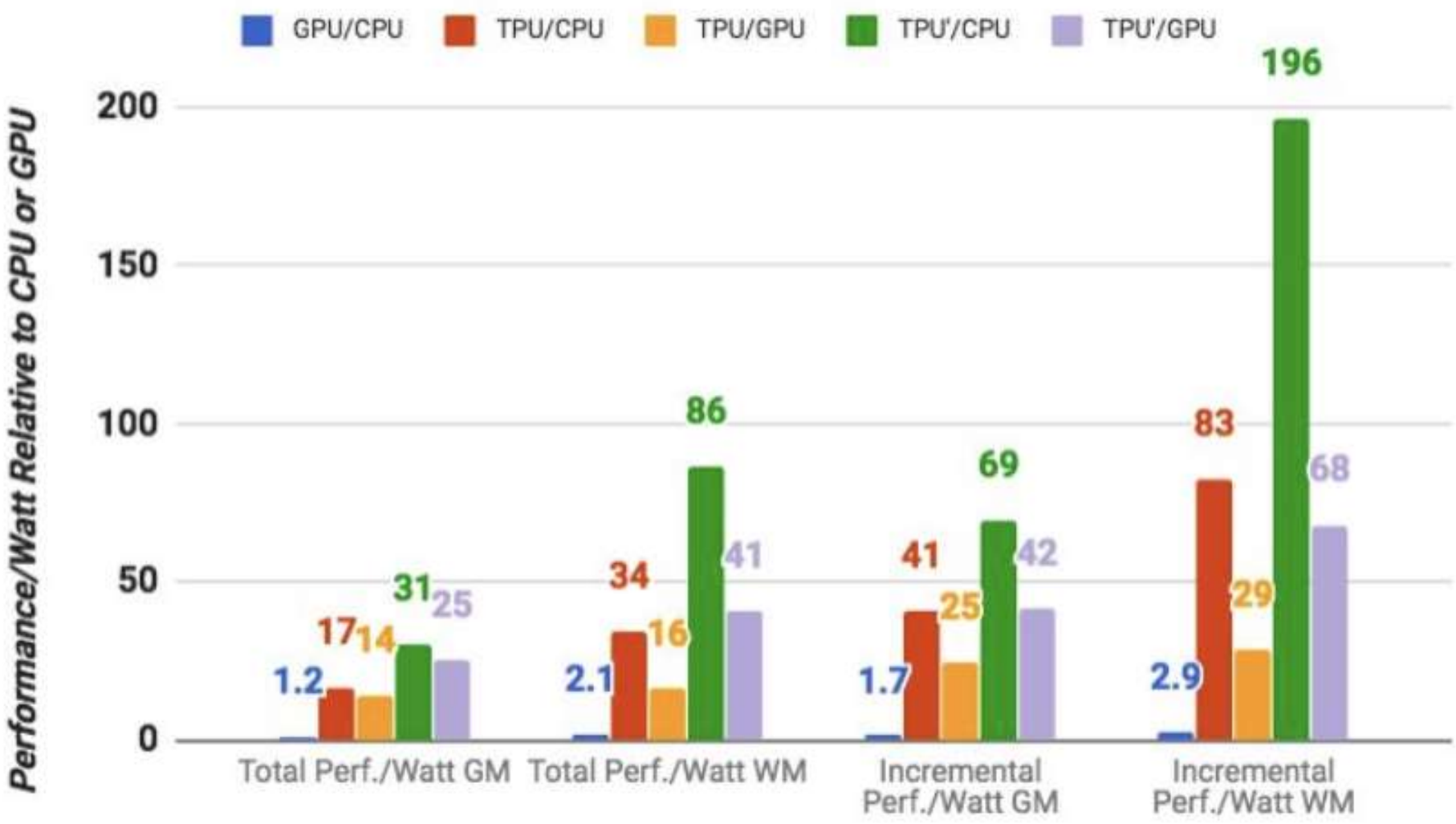


Figure 9. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU (Sec. 7). The green bar shows its ratio to the CPU server and the lavender bar shows its relation to the GPU server. Total includes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

Conclusion

- Although most researchers are optimizing CNNs, they represent just 5% of google's datacenter workload.
- Since Inference apps are user-facing, they emphasize response-time over throughput
- Due to latency limits, the K80 GPU is just a little faster than the CPU, for inference.
- Despite having a much smaller and lower power chip, the TPU has 25 times as many MACs and 3.5 times as much on-chip memory as the K80 GPU.
- The TPU is about 15X - 30X faster at inference than the K80 GPU and the Haswell CPU.
- Four of the six NN apps are memory-bandwidth limited on the TPU; if the TPU were revised to have the same memory system as the K80 GPU, it would be about 30X - 50X faster than the GPU and CPU.
- The performance/Watt of the TPU is 30X - 80X that of contemporary products; the revised TPU with K80 memory would be 70X - 200X better.

TPU vs FPGA

- Jouppi says, that the hardware engineering team did look to FPGAs to solve the problem of cheap, efficient, and high performance inference early on before shifting to a custom ASIC.

- FPGA vs ASIC: there ends up being a big difference in performance and performance per watt

- However, the TPU is programmable like a CPU or GPU, it is not for just one neural network.

NVIDIA response

- Unfair comparison (out-dated GPU)
- TPU is limited to only Inference phase

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	$^1/_{13}$X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

References

- http://www.cs.iusb.edu/~danav/teach/c463/12_nn.html
- https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/
- https://cloudplatform.googleblog.com/2017/04/quantifying-the-performance-of-the-TPU-our-first-machine-learning-chip.html
- *Draft Paper:* https://drive.google.com/file/d/0Bx4hafXDDq2EMzRNcy1vSUxtcEk/view
- https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/
- *Systolic Execution:* http://web.cecs.pdx.edu/~mperkows/temp/May22/0020.Matrix-multiplication-systolic.pdf