

Ten Lessons From Three Generations Shaped Google's TPUv4i

Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson, Google LLC

Published in 2021 at the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)

Presented by Jakub Mandula

Executive summary

Problem & Motivation

- **Growing demand** for AI applications
- State of the art DNN architectures **continuously changing**
- Training and *inference* **expensive**

Goal

- Flexible, cost effective, scalable HW for efficient, low latency inference

Key Ideas

- *Identify and exploit* **Lessons** from previous TPU versions
- Leverage **technological improvement**
 - Bring memory closer to processing elements
 - Exploit existing compiler optimizations
 - Expand computational capacity where appropriate

Results

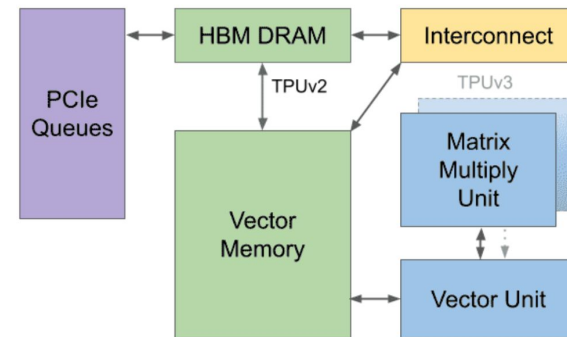
- Compared to *TPUv3*: **2.3x perf/TDP** using 1.6x transistors
- Compared to *Nvidia T4*: **1.3-1.6x speed** @ 0.9-1.0x perf/TDP

Outline

- Background - What is a TPU?
- Summary of the 10 Lessons
- How have these 10 Lessons shaped the design of *TPUv4i*
- Performance evaluation
- Strengths and Weaknesses
- Takeaways and Insights
- Discussion

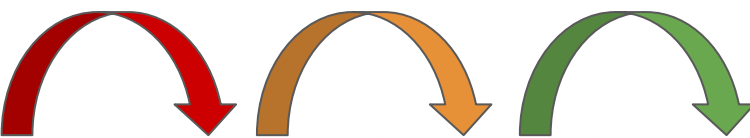
What is a TPU?

- **Tensor Processing Unit**
 - Domain-Specific Architectures DSA
 - Developed by **Google**
 - Training & Inference
 - *TPUv4i* - **inference only**
- **AI Workloads**
 - Matrix multiplications
 - Convolutions
 - Activation evaluation
- **Better efficiency compared to CPU or GPUs**
 - 30-80X higher performance/Watt



How did the TPU design evolve?

2.3x better
perf/TDP



Feature	TPUv1	TPUv2	TPUv3	TPUv4i	NVIDIA T4
Peak TFLOPS / Chip	92 (8b int)	46 (bf16)	123 (bf16)	138 (bf16/8b int)	65 (ieee fp16)/130 (8b int)
First deployed (GA date)	Q2 2015	Q3 2017	Q4 2018	Q1 2020	Q4 2018
DNN Target	Inference only	Training & Inf.	Training & Inf.	Inference only	Inference only
Network links x Gbits/s / Chip	--	4 x 496	4 x 656	2 x 400	--
Max chips / supercomputer	--	256	1024	--	--
Chip Clock Rate (MHz)	700	700	940	1050	585 / (Turbo 1590)
Idle Power (Watts) Chip	28	53	84	55	36
TDP (Watts) Chip / System	75 / 220	280 / 460	450 / 660	175 / 275	70 / 175
Die Size (mm ²)	< 330	< 625	< 700	< 400	545
Transistors (B)	3	9	10	16	14
Chip Technology	28 nm	16 nm	16 nm	7 nm	12 nm
Memory size (on-/off-chip)	28MB / 8GB	32MB / 16GB	32MB / 32GB	144MB / 8GB	18MB / 16GB
Memory GB/s / Chip	34	700	900	614	320 (if ECC is disabled)
MXU Size / Core	1 256x256	1 128x128	2 128x128	4 128x128	8 8x8
Cores / Chip	1	2	2	1	40
Chips / CPUHost	4	4	4	8	8

Outline

- Background - What is a TPU?

- **Summary of the 10 Lessons**

1. Lessons applying to any DSAs
2. Lessons Focusing on DNN DSAs
3. Lessons Regarding DNN applications

- **Strengths and Weaknesses**

- **Takeaways and Insights**

- **Discussion**

Summary of the 10 Lessons

1. Lessons applying to any DSA

... and potentially also to CPUs and GPUs in general

① Logic, wires, SRAM, & DRAM improve unequally

- Updated Horowitz's Energy per Operation
- **Logic improves faster** than wires or SRAM
 - Logic is relatively “free”
- High Bandwidth Memory (HBM) - short DRAM stacks close to DSA

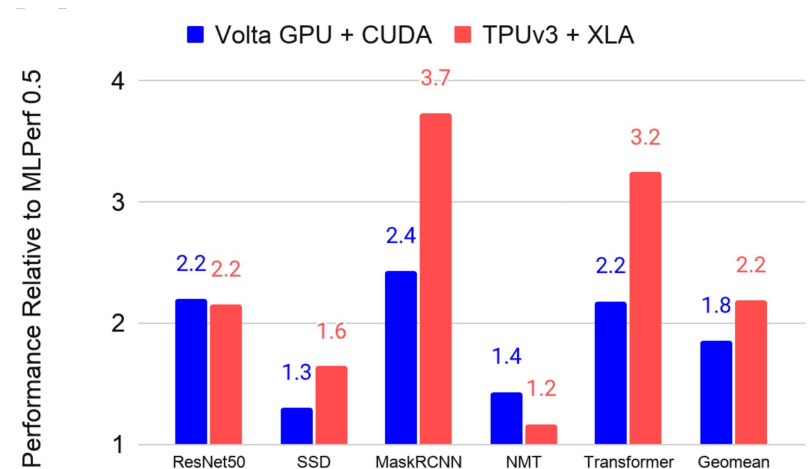
Energy per operation [pJ] 45nm vs 7nm

Operation		Picojoules per Operation		
		45 nm	7 nm	45 / 7
+	Int 8	0.03	0.007	4.3
	Int 32	0.1	0.03	3.3
	BFloat 16	--	0.11	--
	IEEE FP 16	0.4	0.16	2.5
	IEEE FP 32	0.9	0.38	2.4
×	Int 8	0.2	0.07	2.9
	Int 32	3.1	1.48	2.1
	BFloat 16	--	0.21	--
	IEEE FP 16	1.1	0.34	3.2
	IEEE FP 32	3.7	1.31	2.8
SRAM	8 KB SRAM	10	7.5	1.3
	32 KB SRAM	20	8.5	2.4
	1 MB SRAM ¹	100	14	7.1
GeoMean ¹		--	--	2.6
DRAM		Circa 45 nm	Circa 7 nm	
	DDR3/4	1300 ²	1300 ²	1.0
	HBM2	--	250-450 ²	--
	GDDR6	--	350-480 ²	--

② Leverage prior compiler optimizations

- C compilers improve 1-2% annually
 - Nvidia CUDA (2007) **1.8x**
 - Google TPU XLA (2016) **2.2x**
-
- **Performance of DSAs is compelled by quality of their compilers**
 - Significant compiler optimizations come after hardware is available
 - HW must stay compiler compatible to exploit the optimizations in future

Relative DSA compiler gains over 20 months on MLPerf benchmark



③ Design for performance per TCO vs per CapEx

- *Capital Expense (CapEx)* - Initial purchase cost for an item
- *Operation Expense (OpEx)* - The cost of electricity and provisioning over the lifetime of an item.

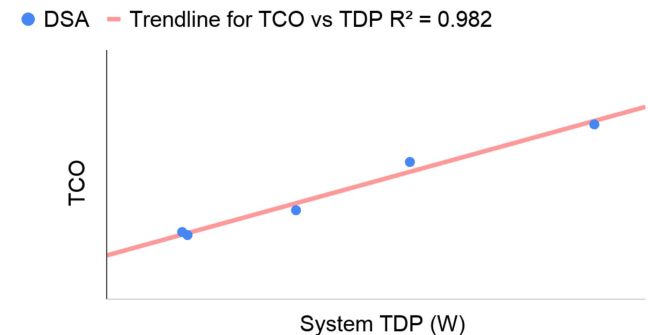
Computer hardware lifetime: 3-5 years

- *Total Cost of Ownership (TCO)*

$$TCO = CapEx + 3 \times OpEx$$

- CPUs and GPUs aim at best *performance/CapEx*
- Companies aim at good *performance/TCO*

Correlation of System TDP and TCO



Summary of the 10 Lessons

2. Lessons focusing on DNN DSAs

④ Support Backwards ML Compatibility

- Developers don't want to **change existing** DNNs
 - Quantization - time **costly** and **loss of accuracy**
 - Time-to-market constraints of deployed DNNs
 - **Minimize effort** in migrating to new hardware

⑤ Inference DSAs need air cooling for global scale

TPU version	TDP	Cooling	Peak TFLOPS/Chip
TPUv1 (inf)	75W	Air	92 (8b int)
TPUv2 (train + inf)	280W	Air	46 (bf16)
TPUv3 (train + inf)	450W	Liquid	123 (bf16)
TPUv4i (inf)	175W	Air	138 (bf16/8b int)

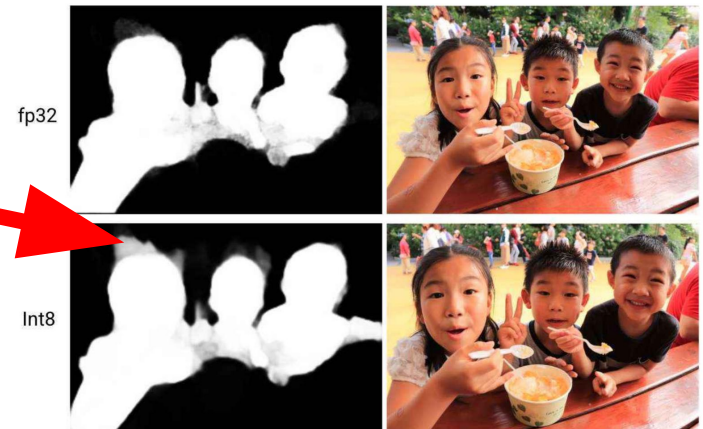
- High TDP requires Liquid cooling
 - Expensive at small scales

- Low latency worldwide user-facing inference
 - **Air Cooling - easier deployment**

⑥ Some inference apps need floating point arithmetic

- **Training** performed in FP (fp32, bfp16)
 - **Inference** sometimes quantized to int8
 - Better *area and power*
 - Reduced *accuracy and delayed deployment*
 - Some applications don't work with quantization
-
- ImageNet improved by 1% 2019-2020
 - ④ Support Backwards ML Compatibility
 - Inference Hardware should support **Floating Point Operations**

Inferior performance of quantized image segmentation model



Summary of the 10 Lessons

3. Lessons regarding DNN applications

⑦ Production inference normally needs multi-tenancy

- **Sharing** can *lower cost* and *reduce latency*
- Flexible SW engineering
- Hardware should support fast model switching

Multi-tenancy Requirements across Google ML Workload

Name	Avg. Size (MB)	Max Size (MB)	Multi-tenancy?	Avg. Number of Programs (StdDev), Range	% Use 2016/2020
MLP0	580	2500	Yes	27 (± 17), 1-93	61%-25%
MLP1	90	N.A.	Yes	5 (± 0.3), 1-5	
CNN0	60	454	No	1	5%-18%
CNN1	120	680	Yes	6 (± 10), 1-34	
RNN0	1300	1300	Yes	13 (± 3), 1-29	0%-29%
RNN1	120	400	No	1	
BERT0	3000	3000	Yes	9 (± 2), 1-14	0%-28%
BERT1	90	N.A.	Yes	5 (± 0.3), 1-5	

Lessons Regarding DNN Application evolution

⑧ DNNs grow $\sim 1.5x/\text{year}$ in memory and compute

⑨ DNN workloads evolve with DNN breakthroughs

- DNNs continuously updated
- TPU needs sufficient hardware

- What will come next...?
- **Programmability and flexibility**

Google ML Workload

Name	Avg. Size (MB)	Max Size (MB)	Multi-tenancy?	Avg. Number of Programs (StdDev), Range	% Use 2016/2020
MLP0	580	2500	Yes	27 (± 17), 1-93	61%-25%
MLP1	90	N.A.	Yes	5 (± 0.3), 1-5	
CNN0	60	454	No	1	5%-18%
CNN1	120	680	Yes	6 (± 10), 1-34	
RNN0	1300	1300	No	13 (± 3), 1-29	0%-29%
RNN1	120	400		1	
BERT0	3000	3000	Yes	9 (± 2), 1-14	0%-28%
BERT1	90	N.A.	Yes	5 (± 0.3), 1-5	

⑩ Inference limited by latency, not throughput

- Batchsize size = Throughput
- In datacenters **Latency** is major limitation
- Hardware should hide Latency

Production						MLPerf 0.7		
<i>DNN</i>	<i>ms</i>	<i>batch</i>	<i>DNN</i>	<i>ms</i>	<i>batch</i>	<i>DNN</i>	<i>ms</i>	<i>batch</i>
MLP0	7	200	RNN0	60	8	Resnet50	15	16
MLP1	20	168	RNN1	10	32	SSD	100	4
CNN0	10	8	BERT0	5	128	GNMT	250	16
CNN1	32	32	BERT1	10	64			

Table 5. Latency limit in ms and batch size picked for TPUv4i.

Outline

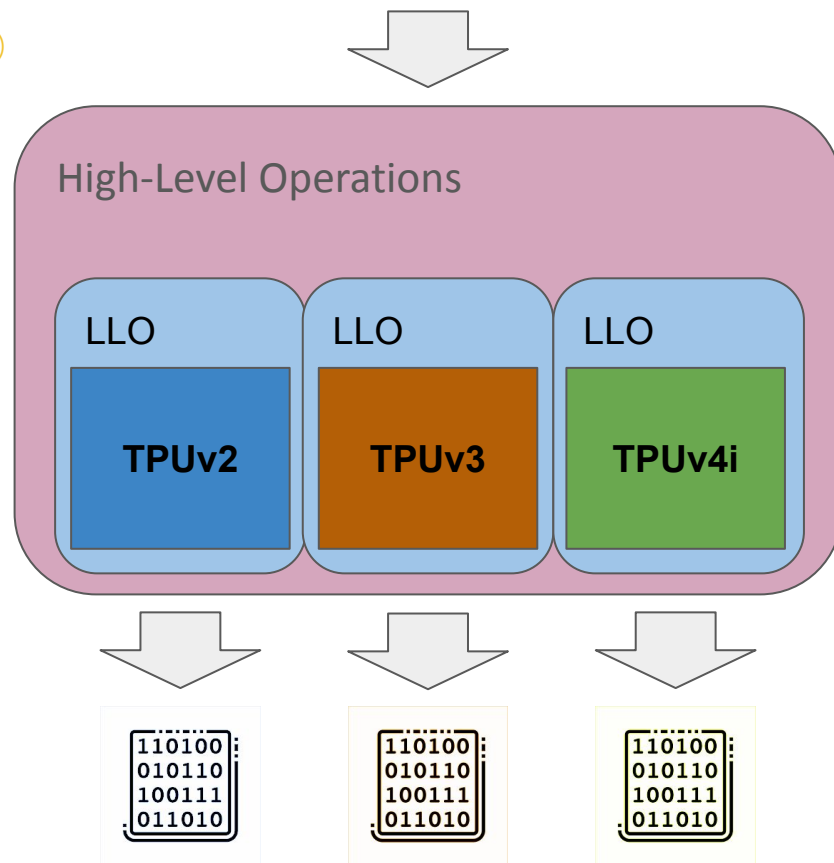
- Background - What is a TPU?
- Summary of the 10 Lessons
- **How have these 10 Lessons shaped the design of *TPUv4i***
- **Performance evaluation**
- **Strengths and Weaknesses**
- **Takeaways and Insights**
- **Discussion**

Compiler compatibility

- Based on TPUv3 HW design
 - compiler optimization ②
 - maintain backwards ML compatibility ④

- Compiler compatibility
 - XLA compiler separates:
 - High-Level Operations (HLO)
 - Hardware agnostic
 - Low-Level Operations (LLO)
 - Hardware dependent
 - Maintains compiler compatibility ②

```
class LeNet(nn.Module):  
    def __init__(self):  
        super(LeNet, self).__init__()  
        # 1 input image channel (black & white), 6 output channels, 3x3 square convolution  
        # kernel  
        self.conv1 = nn.Conv2d(1, 6, 3)  
        self.conv2 = nn.Conv2d(6, 16, 3)  
        # an affine operation: y = Wx + b  
        self.fc1 = nn.Linear(16 * 6 * 6, 120) # 6*6 from image dimension  
        self.fc2 = nn.Linear(120, 84)  
        self.fc3 = nn.Linear(84, 10)  
  
    def forward(self, x):  
        # Max pooling over a (2, 2) window  
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))  
        # If the size is a square you can only specify a single number  
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
```



On-chip storage & DMA

- **HBM kept from TPUv3**
 - Multi-tenancy ⑦
 - Rapid DNN growth ⑧
 - Better energy efficiency to DRAM ①
- **SRAM - CMEM**
 - 128MB CMEM 28% of die
 - 20x more efficient than DRAM ①
 - - Significant fraction of TCO ③
- **4D tensor DMA**
 - Programmable & Flexible ⑧⑨
 - Support for various striding techniques
 - Compiler compatible ②
 - Synchronizing partial completion progress, hiding DMA ramp-up, ramp-down latency ⑩②

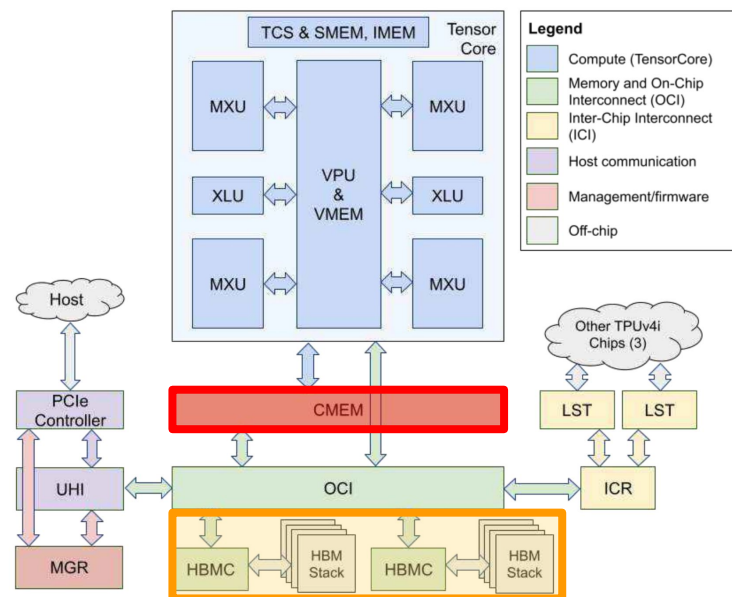


Figure 5. TPUv4i chip block diagram. Architectural memories

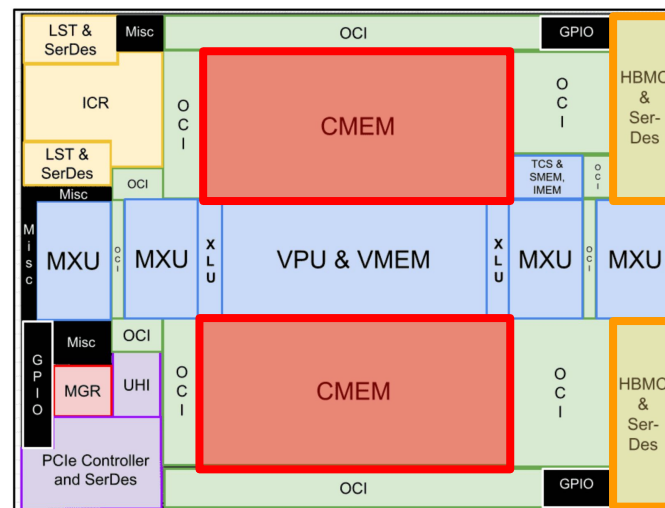


Figure 6. TPUv4i chip floorplan.

Custom on-chip interconnect (OCI)

- Point-to-point routing infeasible ①

Custom OCI

- Connects all components on die
- **Flexible & scalable** topology ⑧ ⑨
- HBM Bandwidth/core
 - **1.3x over TPUv3** ⑧
- 4 non-overlapping network groups
 - Provides locality + reduced latency ①
 - 153GB/s HBM bandwidth

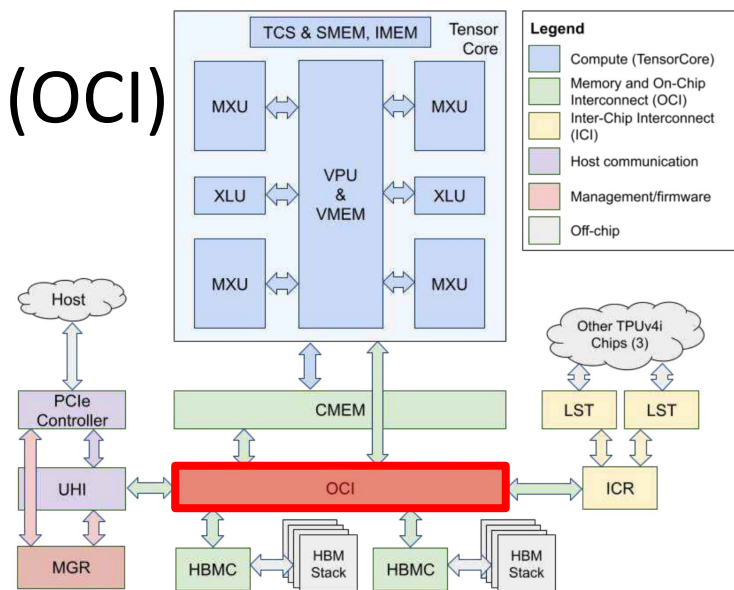


Figure 5. TPUv4i chip block diagram. Architectural memories

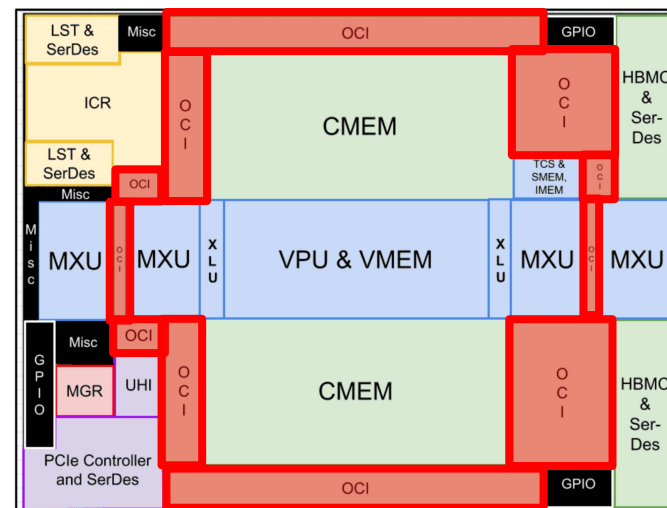


Figure 6. TPUv4i chip floorplan.

Arithmetic unit & TDP

- Retain both **int8** and **bf16** support
 - Logic is “free” ①
 - TPUv1 - TPUv3 ML compatibility ④
 - **Not requiring** quantization ⑥
- 4 MXUs per chip
 - XLA can handle 2x MXUs ①②
- Custom 4-input FP adders
 - Minimal numerical difference ④
 - 40% area saved reducing CapEx ③
 - 12% lower peak power ➤ Lower TDP ⑤
- TDP 175W @ 1.05GHz
 - closer to TPUv1 (75W)
 - Allows Air cooling ⑤
 - Reduces TCO (OpEx) ③

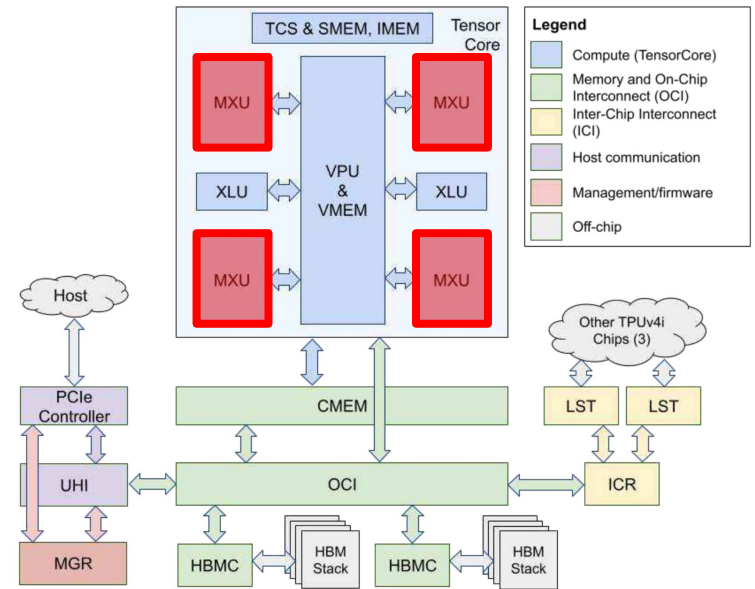


Figure 5. TPUv4i chip block diagram. Architectural memories

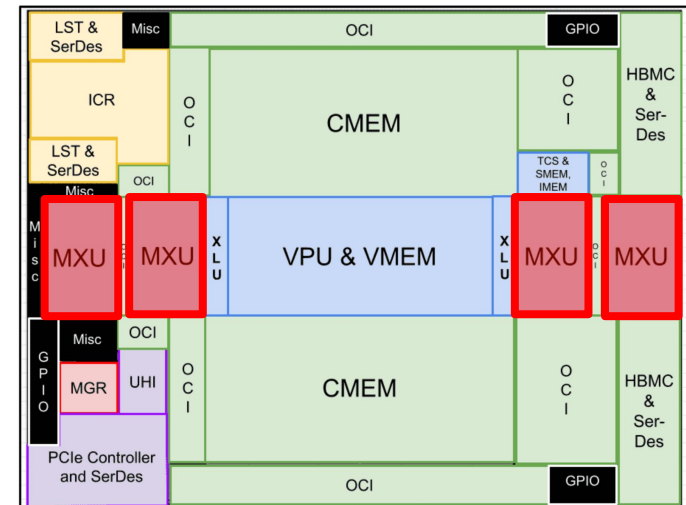


Figure 6. TPUv4i chip floorplan.

ICI scaling & Workload analysis

- 2 ICI links
 - 4 chips/board access nearby memory quickly for future DNN growth ⑧
- Extensive tracing a HW performance counters included
 - Analyze system-level bottlenecks
 - Increased Design Time but worth it since target is perf/TCO, not perf/CapEx ③
 - System-level performance improvements (compiler) ②
 - Boost developer productivity ⑦ ⑧ ⑨

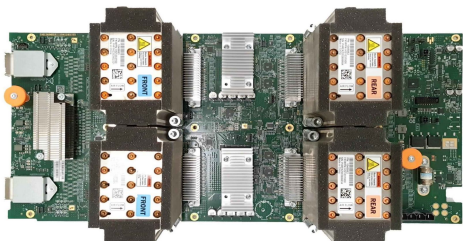


Figure 7. TPUv4i board with 4 chips that are connected by ICI.

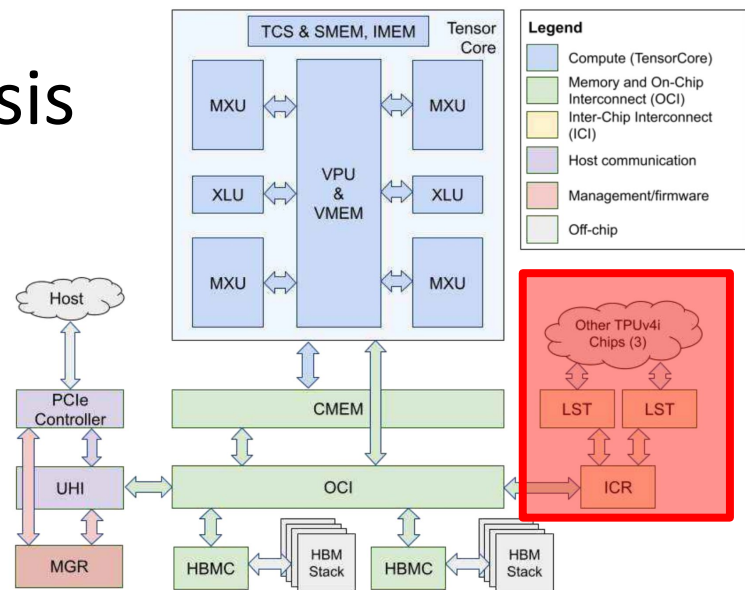


Figure 5. TPUv4i chip block diagram. Architectural memories

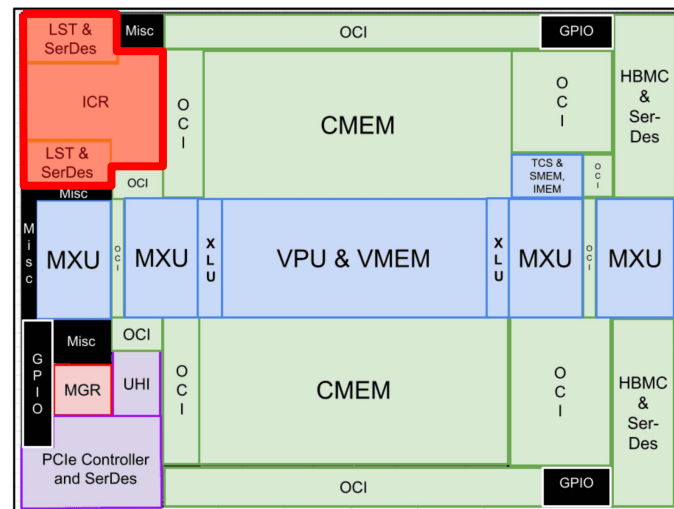


Figure 6. TPUv4i chip floorplan.

Outline

- Background - What is a TPU?
- Summary of the 10 Lessons
- How have these 10 Lessons shaped the design of *TPUv4i*

- **Performance evaluation**

- **Strengths and Weaknesses**
- **Takeaways and Insights**
- **Discussion**

Performance/Watt

- TPUv3 and TPUv4i ~1.9x faster
 - TPUv2 & TPUv3 have 2 cores
 - TPUv4i has 1 - winning perf/TCO
- TPUv4i has 2.3x perf/TDP vs TPUv3

Breakdown:

- **CMEM ~1.5x**
- **7nm ~1.3x**
- **Other contributions ~1.2x**

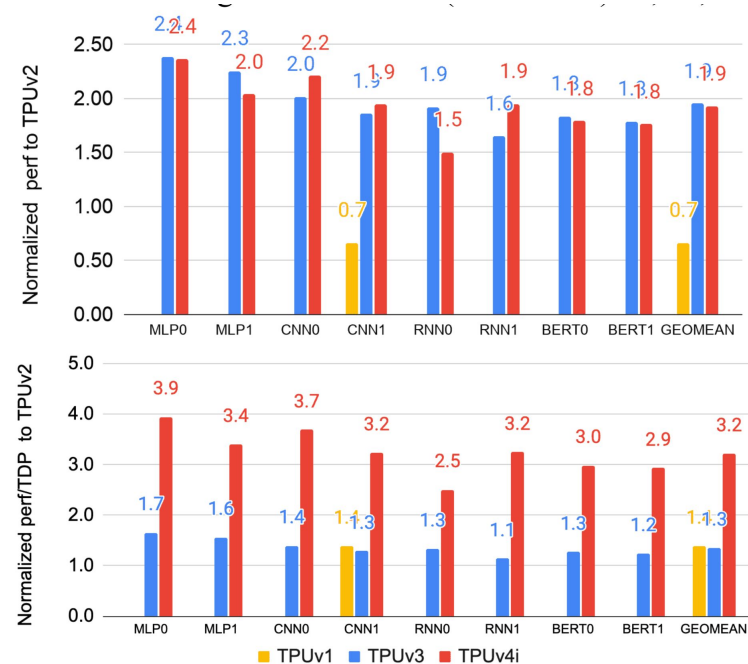


Figure 8. Performance (top) and performance/system Watt ③ for production apps ⑨ relative to TPUv2 for the other TPUs.

Strengths

- **5 years** of Design Team's **experience**
- Detailed overview of **main principles** guiding modern DSA architecture decisions
- Detailed analysis of **workload requirements** and TPU benchmarks
- Pragmatic, **Production-focused**, industry approach

Weaknesses

- Extremely **broad coverage**
 - Design details are covered **superficially**
- Some Lessons are extremely restricting
 - Production & compatibility focus ②④ ➤ Restricts innovation
- **Limited benchmark** comparison to competition DSAs, and alternative architectures CPU,GPU (Nvidia T4)
- Some Lessons redundant
 - ④ Support Backwards ML Compatibility & ⑥ Some inference apps need floating point arithmetic
 - ⑧ DNNs grow ~1.5x/year in memory and compute & ⑨ DNN workloads evolve with DNN breakthroughs
- Benchmarks mainly on **Google Workloads**

Key Takeaways and insights

- Documentation of **unequal technological improvement** (①)
- Significance of **compiler** optimization (②)
- **Design** for *perf/TCO* vs *perf/CapEx* (③)
- Significance of **Backwards ML compatibility** in production (④)
- **Know your workload**
- **Iterative improvement** is key for evolving problems

Discussion and Questions

As Moore's law reaches plateau and Dennard scaling finishes, what are the next steps to **keep pace with growing DNN models?**

Discussion and Questions

Is compiler, ML and hardware backwards compatibility
restricting innovation in production hardware?

Discussion and Questions

Is **sacrificing flexibility**, that Google seeks in its TPU hardware, a viable approach in extracting more performance from current hardware?

As Moore's law reaches plateau and Dennard scaling finishes, what are the next steps to **keep pace with growing DNN models?**

- Alternative model architectures?
- ICI scaling - high bandwidth interchip communication
- Near, in memory computing?
- New computing paradigms?
 - Spiking neural networks
 - Memristor computing

Is compiler, ML and hardware backwards compatibility **restricting innovation** in production hardware?

- Production hardware generally has slower adoption rate.
- Exploit compilers to provide bridge between new paradigms

Is **sacrificing flexibility**, that Google seeks in its TPU hardware, a viable approach in extracting more performance from current hardware?

- Edge processing hardware does not always require flexibility
- Personal devices have a shorter lifespan + don't care about OpExp

Thank you to my advisors



Gagandeep Singh



Joël Lindegger



Nika Mansouri Ghiasi

Backup Slides: Roofline model

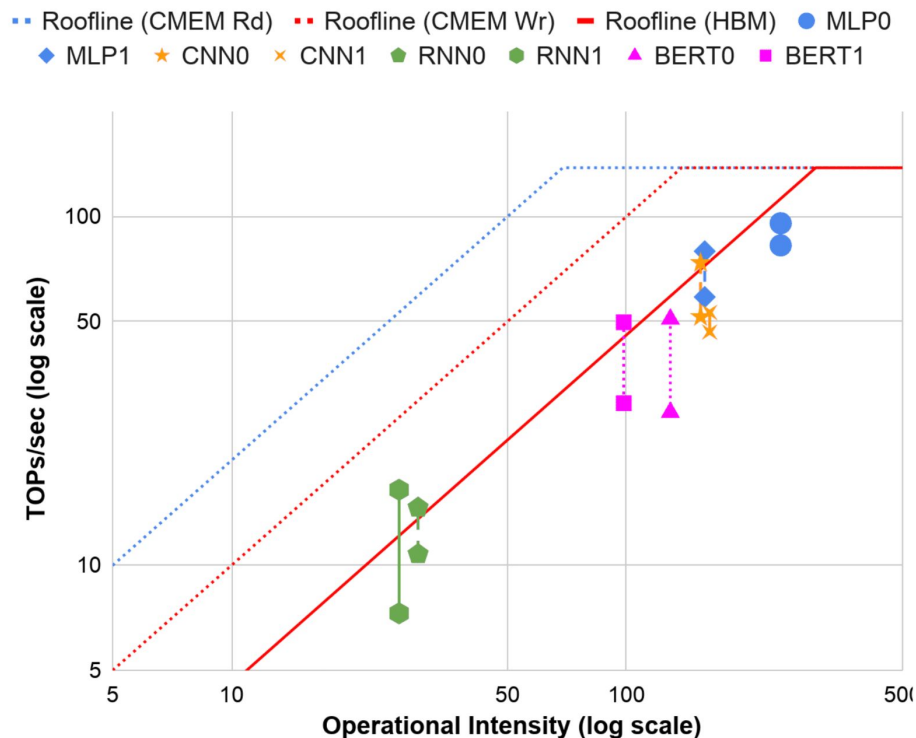


Figure 12. Roofline model showing apps without CMEM (low point) vs with CMEM (high point). Operational intensity (OI) here is operations divided by memory accesses to HBM *or* to CMEM. If OI were relative to HBM only, CMEM would increase OI and move the points to the right as well as up.

Backup Slides: CMEM size

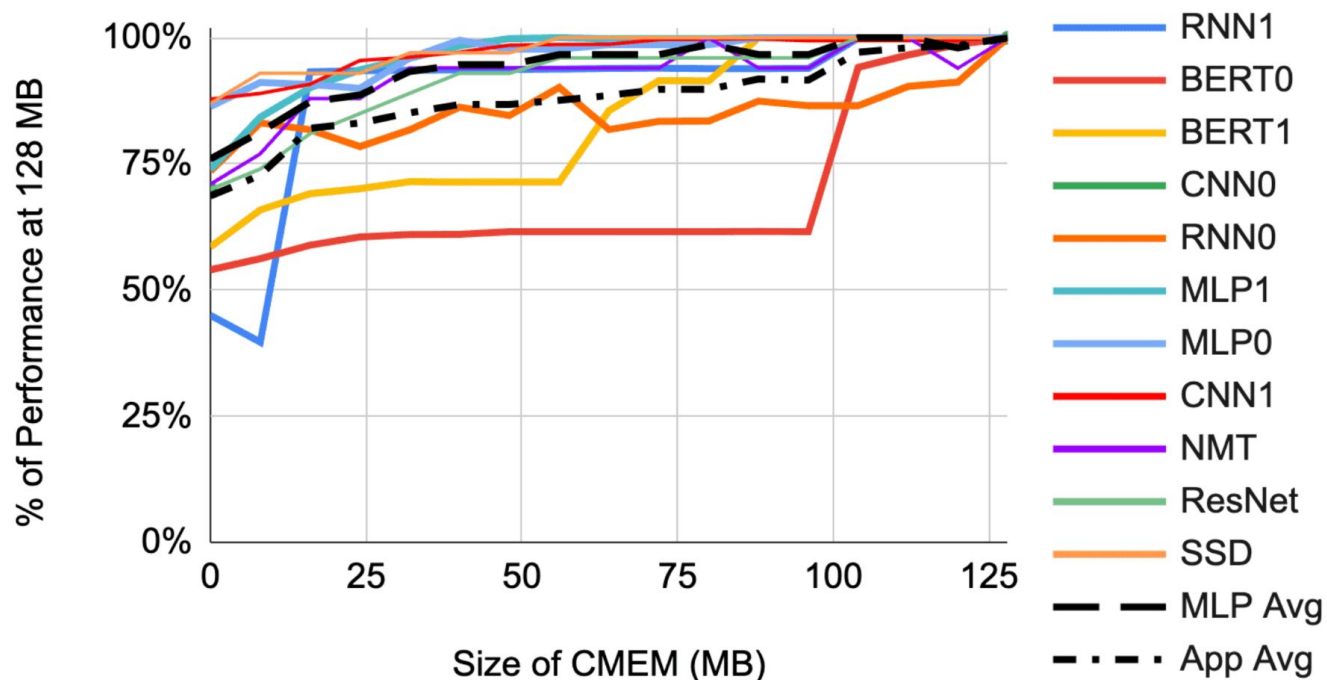


Figure 13. Percent of 128 MB speed as CMEM varies 0–128 MB for the apps and MLPerf Inference 0.5-0.7 server code.

Comparison to NVIDIA T4

- TPUs used bf16
- NVIDIA used int8 (fp16 on NMT)
- TPUv4i wins on speed
 - 1.3-1.6x faster
 - 0.9-1.0x for perf/TDP
 - 1.3x perf/TDP for NMT (both use fp)
- Measuring average power
 - 1.6-2x of T4 for NMT
- For Google, backwards ML compatibility more important than small int8 perf/TDP

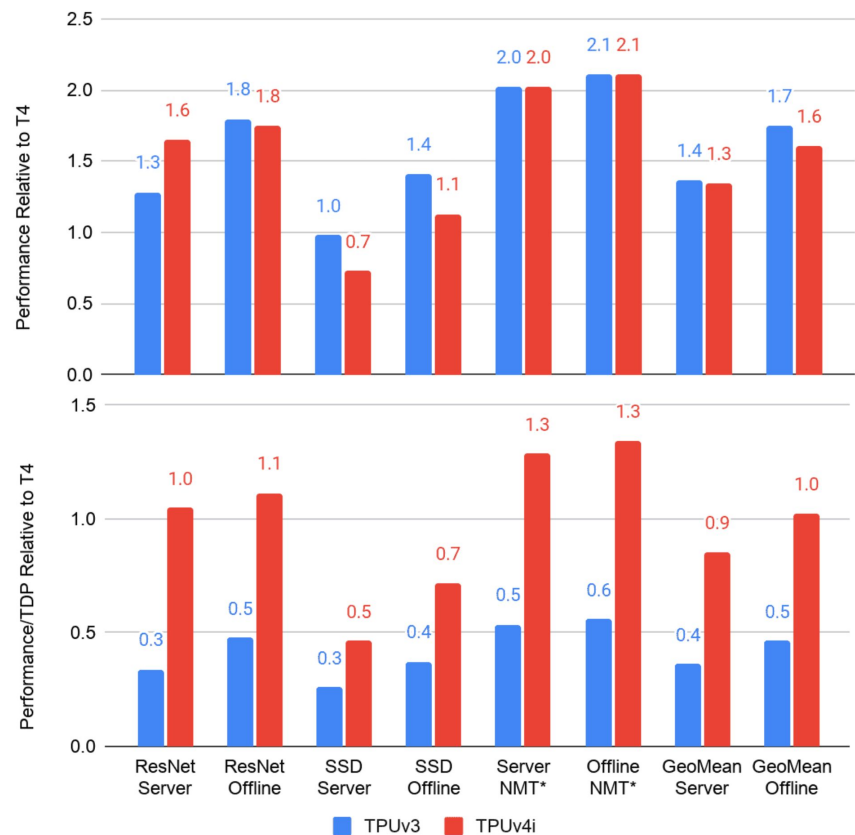


Figure 9. Performance (top) and performance/system TDP ③ relative to T4 for TPUv3/v4i in our datacenter (\$7.B).

Backup Slides: 10 Lessons summary

- ① Logic, wires, SRAM, & DRAM improve unequally
- ② Leverage prior compiler optimizations
- ③ Design for performance per TCO vs per CapEx
- ④ Support Backwards ML Compatibility
- ⑤ Inference DSAs need air cooling for global scale
- ⑥ Some inference apps need floating point arithmetic
- ⑦ Production inference normally needs multi-tenancy
- ⑧ DNNs grow $\sim 1.5x$ /year in memory and compute
- ⑨ DNN workloads evolve with DNN breakthroughs
- ⑩ Inference SLO limit is P99 latency, not batch size