成绩计算

- 课堂参与（**15%**）
  - 出席：6%
    - Review提问点名缺席：-2%
  - Quiz：3% x 3
    - 提交但全错：-3%（即，当次quiz分为0）
    - 未提交：-4%（即，当次quiz分为-1）
    - 代提交：-5%（即，当次quiz分为-2）
- 作业（**35%**）
  - hw1: 10%（理论）
  - hw2: 10%（论文）
  - hw3: 15%（分析）
- 期末考试（**50%**）
  - 闭卷（中文，关键术语英文标注）

期末考试

- 时间地点
  - 2022.1.12(周三)、14:30 - 16:30
    - 注：4个班统一时间，不同试题
  - B201

- 试题类型
  - 选择题(30’): 3’ x 10
  - 简答题(20’): 5’ x 4
  - 应用题(30’): 15’ x 2
  - 综合题(20’): 20’ x 1

50’ + 50’ = 100’

考试内容(初步)

- Instruction-level Parallelism
  - Pipelining, branch prediction, dynamic scheduling, …
- Data-level Parallelism/GPU
  - SIMD/SIMT, GPU architecture, CUDA programming, …
- Memory System
  - Memory hierarchy, cache optimizations, …
  - (TLP) coherence and consistency, …
- Others
  - Quantitative evaluation and metrics
  - Warehouse scale computing
  - Interconnects and domain specific architectures
  - …

Advanced Computer Architecture
高级计算机体系结构

第13讲：Advanced Topics

张献伟

xianweiz.github.io

DCS5367, 12/28/2021

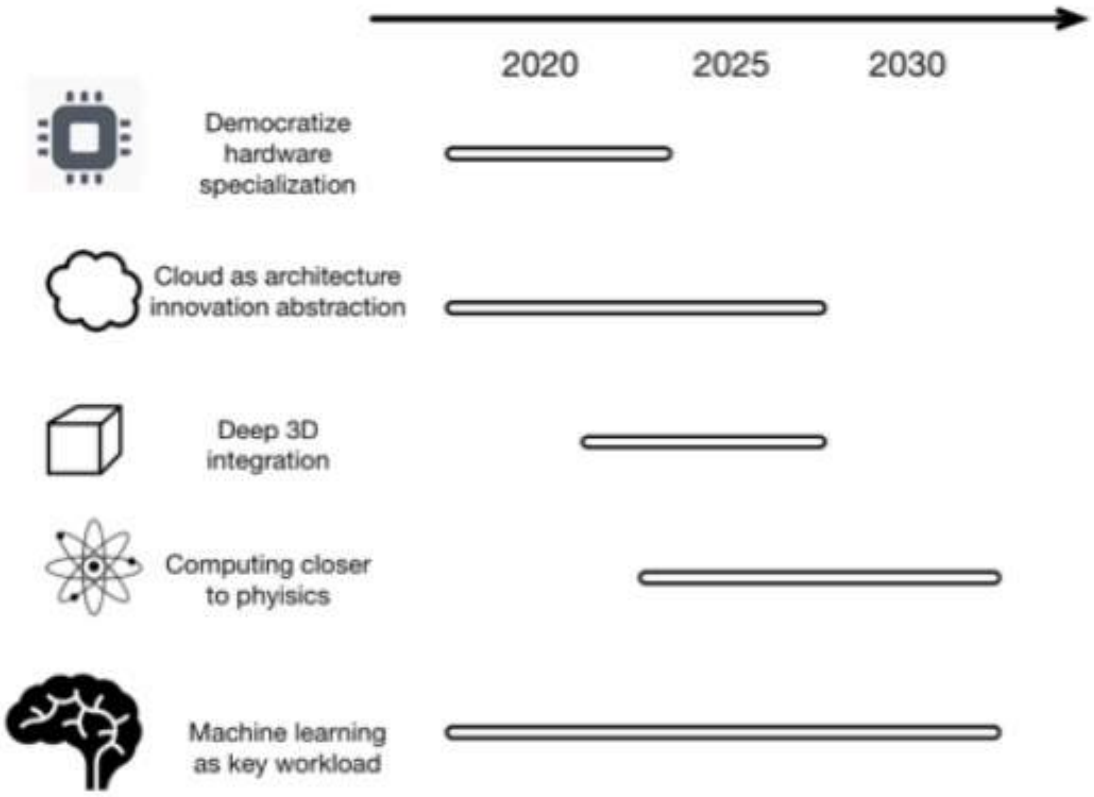
Industry[工业界]

- Nvidia, 04/2021: Grace, ARM-based data-center CPU[1]
- Apple, 11/2020: M1, ARM-based SoC[2]
- AMD, 10/2020: Acquire Xilinx[3]
- Intel, 09/2020: Xe GPU[4]
- Samsung, 11/2019: Cease CPU development[6]
- Amazon, 11/2018: AWS Graviton[5]
- Intel/IBM/ARM, 01/2018: Meltdown and Spectre
- Micron, 03/2021: Cease 3D-XPoint, invest CXL[7]
- AI Chips: Graphcore, Habana Labs, Cerebras, Cambricon…

[1] https://www.nvidia.com/en-us/data-center/grace-cpu/
[2] https://pdf.wondershare.com/macos/everything-about-apple-m1-chip.html
[3] https://www.amd.com/en/corporate/xilinx-acquisition
[4] https://www.intel.com/content/www/us/en/products/discrete-gpus/iris-xe-aic.html
[5] https://aws.amazon.com/ec2/graviton/
[6] https://www.anandtech.com/show/15061/samsung-to-cease-custom-cpu-development
[7] https://investors.micron.com/news-releases/news-release-details/micron-updates-data-center-portfolio-strategy-address-growing

Academia[学术界]

Architecture 2030 Workshop @ ISCA 2016



John L. Hennessy, David A. Patterson

- Current challenges
  - End of Moore's Law and Dennard Scaling
  - Overlooked security
- Future opportunities in computer architecture
  - Domain-specific architectures
  - Domain-specific languages
  - Open architectures
  - Agile hardware development

[1] Arch2030, https://arxiv.org/pdf/1612.03182.pdf (2016)
[2] A New Golden Age for Computer Architecture (2019)

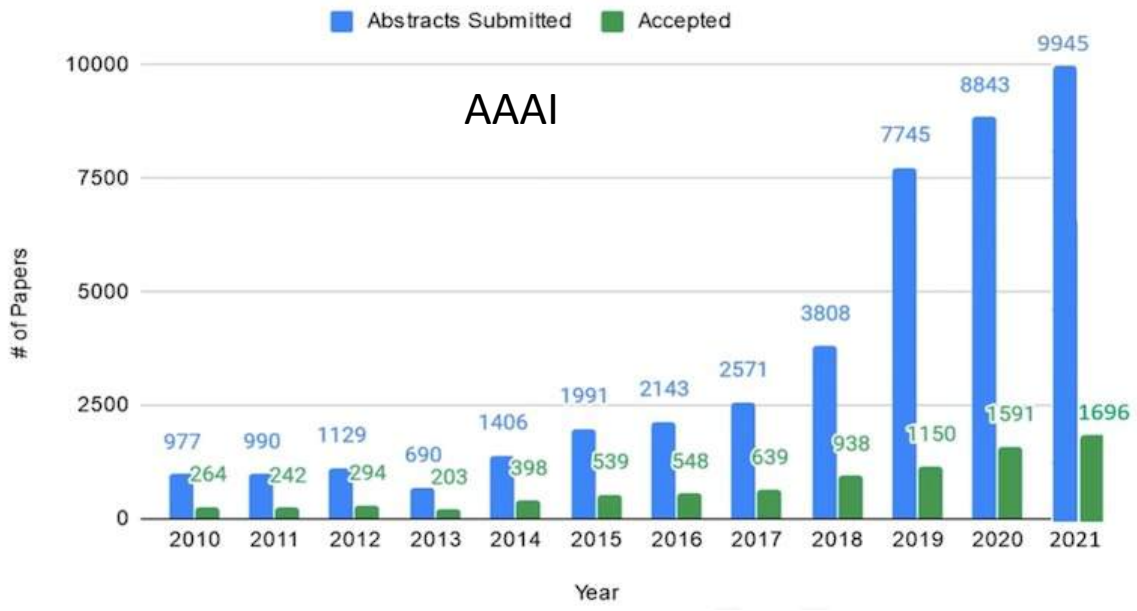
Top-tier Conferences[顶级会议]

- **ISCA**
  - The International Symposium on Computer Architecture (ISCA)
  - 2021: 48th, 76/407 papers (18.6% acceptance rate)
- **MICRO**
  - The IEEE/ACM International Symposium on Microarchitecture
  - 2021: 54th, 94/430 papers (21.8% acceptance rate)
- **HPCA**
  - IEEE International Symposium on High-Performance Computer Architecture
  - 2021: 27th, 63/258 papers (24.4% acceptance rate)
- **ASPLOS**
  - ACM International Conference on Architectural Support for Programming Languages and Operating Systems
  - 2021: 26th, 75/398 (18.8% acceptance rate)

Arch vs. AI

Year	Submitted	Accepted	Rate
ISCA '19	365	62	17%
ISCA '17	322	54	17%
ISCA '13	288	56	19%
ISCA '12	262	47	18%
ISCA '11	208	40	19%
ISCA '10	245	44	18%

Year	Submitted	Accepted	Rate
ASPLOS '19	351	74	21%
ASPLOS '18	319	56	18%
ASPLOS '17	320	53	17%
ASPLOS '16	232	53	23%
ASPLOS '15	287	48	17%
ASPLOS '14	217	49	23%



HPCA'2021 (High-Performance Computer Architecture)

- Sessions
  - Security Architecture
  - Security Attacks
  - Accelerators for Machine Learning (2)
  - Systems for Machine Learning (2)
  - Hardware Accelerators Beyond Machine Learning
  - Storage Systems
  - Memory and Storage Architectures
  - Cache Design
  - Network on Chip
  - High Throughput Architectures
  - Power Efficiency and Resiliency
  - Emerging Technologies and Applications
  - Quantum Computing

[1] https://hpca-conf.org/2021/main-program/

ASPLOS'2021 (Arch Support for Prog. Lang. and OS)

- Sessions
  - Memory Systems
  - Persistence (2)
  - Solid State Drives
  - Flow
  - Microservices
  - Systems Software
  - Pages and Machine Architecture
  - Language and Systems (2)
  - Towards Improved Throughputs
  - Tools and Frameworks
  - Mapping and Management of Quantum and Cloud
  - Quantum Abstractions
  - Beyond the Pixels
  - Races and Concurrency
  - Supporting Hardware Parallelism
  - Robots, Optimization, and Robo-optimization
  - Better Hardware through Compilers
  - Data Driven Optimization
  - Neural Net Optimization
  - Beyond Neural Nets

[1] https://asplos-conference.org/program/

ISCA'2021 (Computer Architecture)

- Sessions
  - Industry Track
  - Microarchitecture (2)
  - Memory (3)
  - Machine Learning (2)
  - Processing In/Near Memory
  - Data Center
  - Security (2)
  - Accelerators (3)
  - Compilers
  - Graph Processing
  - Low Temperature/Low Energy Computing
  - Network Storage & Acceleration
  - Quantum/Photonics
  - Reliability & Security
  - DRAM/IO/Network
  - Sparse Processing

[1] https://www.iscaconf.org/isca2021/program/

MICRO'2021 (Microarchitecture)

- Sessions
  - Best Paper
  - Non-Volatile Memory
  - Energy Efficiency & Low Power
  - Security & Privacy (3)
  - Processing In/Near Memory
  - Parallelism
  - Accelerators (3)
  - Reliability & Verification
  - GPGPU
  - Microarchitecture (2)
  - Superconducting & Quantum
  - Sparse Processing
  - Graph Processing
  - Virtual Memory & Prefetching

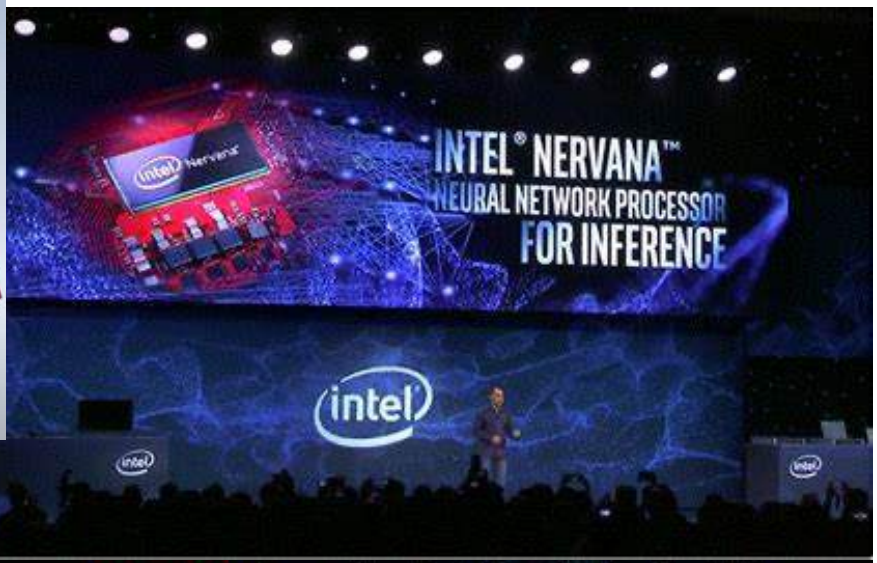
[1] https://www.microarch.org/micro54/program/

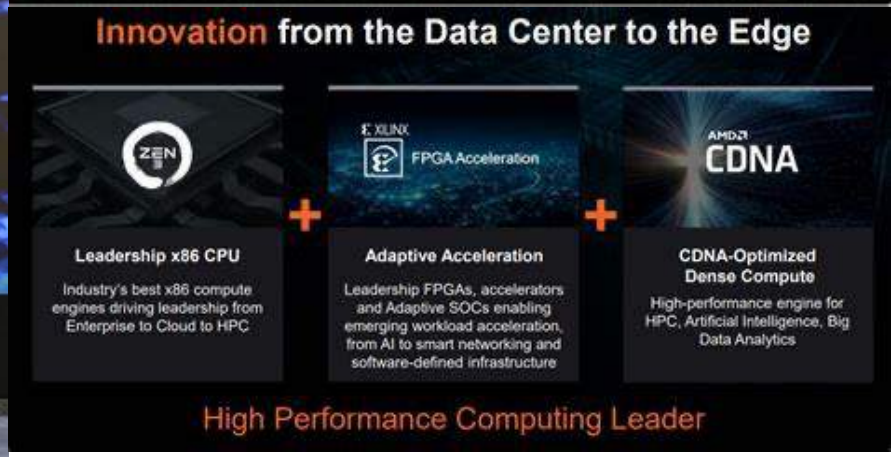
Selected Topics

- **HPCA'2021**
  - Accelerators for Machine Learning (2)
  - Systems for Machine Learning (2)
  - Hardware Accelerators Beyond Machine Learning
  - Quantum Computing
- **ASPLOS'2021**
  - Mapping and Management of Quantum and Cloud
  - Data Driven Optimization
  - Neural Net Optimization
  - Beyond Neural Nets
  - Quantum Abstractions
- **ISCA'2021**
  - Machine Learning (2)
  - Accelerators (3)
  - Graph Processing
  - Sparse Processing
  - Quantum/Photonics
- **MICRO'2021**
  - Accelerators (3)
  - Sparse Processing
  - Graph Processing
  - Superconducting & Quantum

Domain Specific Architecture

HW Companies Building Custom Chips



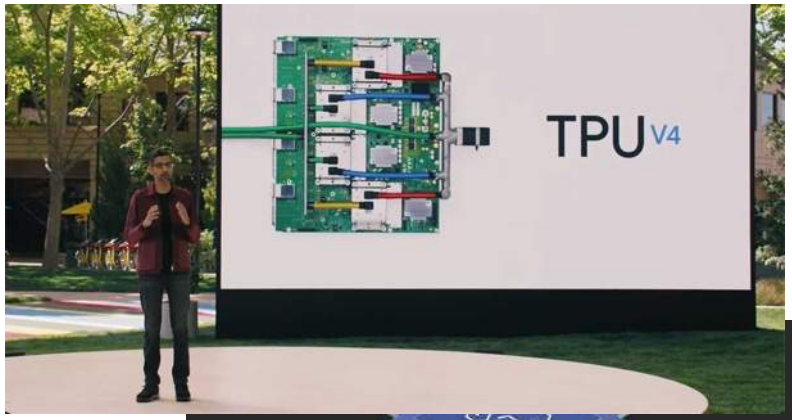


SW Companies are Building HW

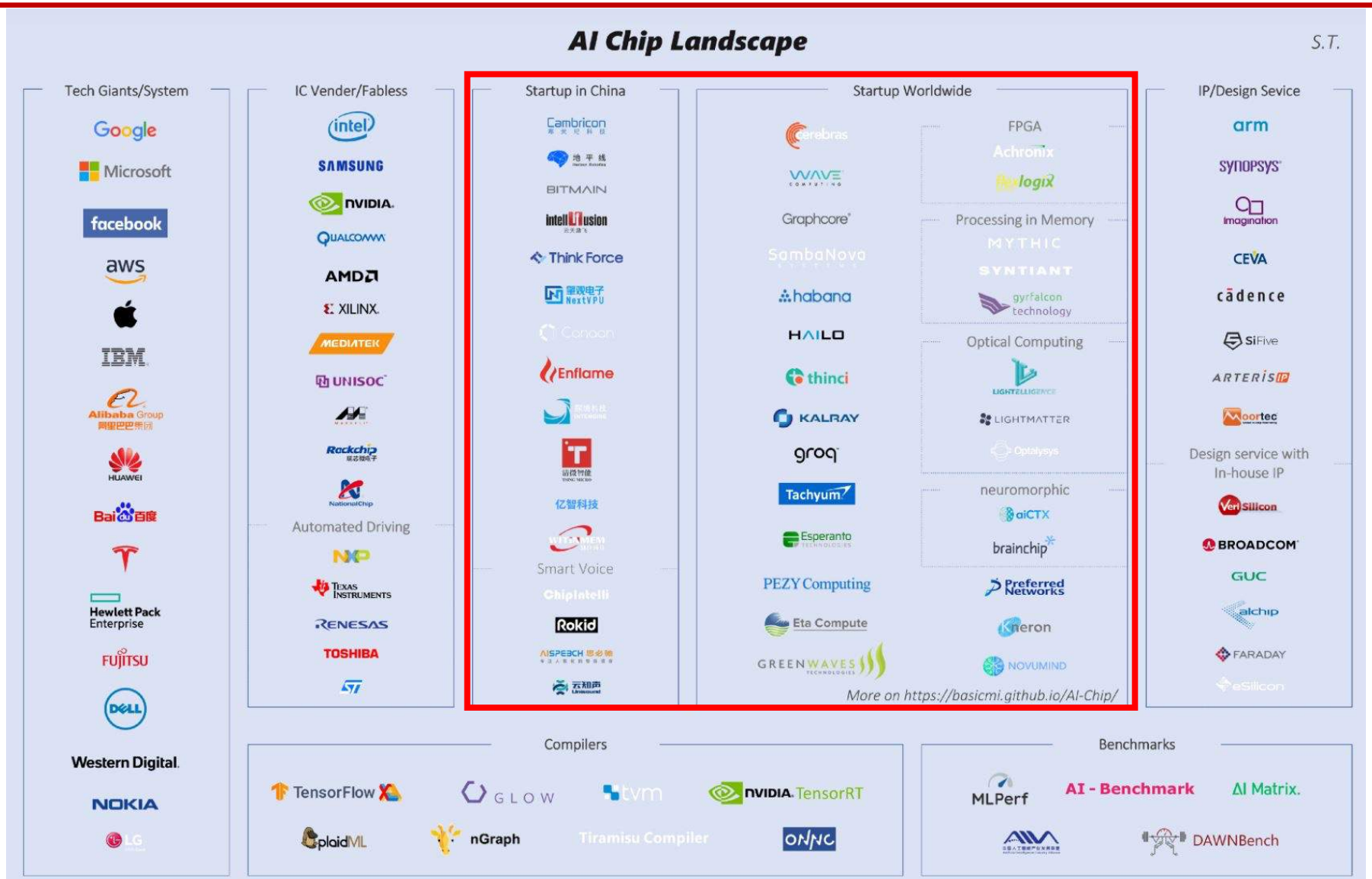
Chips Off the Old Block: Computers Are Taking Design Cues From Human Brains
(September 16, 2017)

After training a speech-recognition algorithm, for example, Microsoft offers it up as an online service, and it actually starts identifying commands that people speak into their smartphones. **G.P.U.s are not quite as efficient during this stage of the process. So, many companies are now building chips specifically to do what the other chips have learned.**

Google built its own specialty chip, a Tensor Processing Unit, or T.P.U. Nvidia is building a similar chip. **And Microsoft has reprogrammed specialized chips from Altera**, which was acquired by Intel, so that it too can run neural networks more easily.



Startups Building Custom Hardware

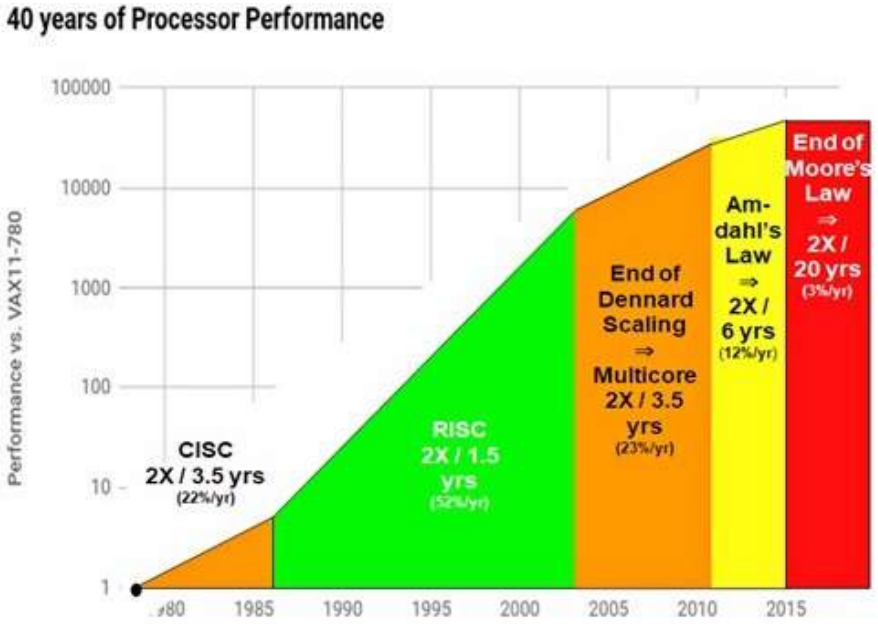


All information contained within this infographic is gathered from the internet and periodically updated, no guarantee is given that the information provided is correct, complete, and up-to-date.

https://www.servethehome.com/dell-emc-talks-deep-learning-and-ai-q3-2019/

Past General-Purpose[通用]

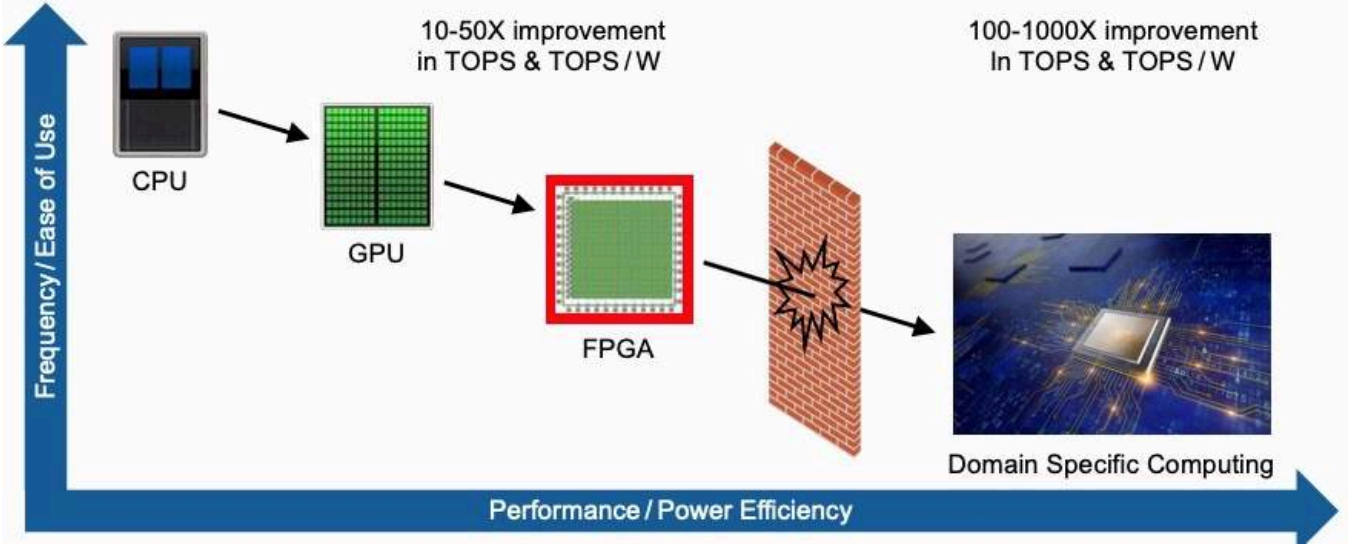
- Moore's Law enabled:
  - Deep memory hierarchy
  - Wide SIMD units
  - Deep pipelines
  - Branch prediction
  - Out-of-order execution
  - Speculative prefetching
  - Multithreading
  - Multiprocessing



- The sophisticated architectures targeted general-purpose code
  - Architects treated code as black boxes
  - Extract performance from software that is oblivious to architecture

Domain-Specific Architecture[领域专用]

- Hard to keep improving performance
  - More transistors means more power
  - Energy budget is limited: higher performance → lower energy/operation
  - Enhancing existing cores may only boost 10% performance
- Need factor of 100 improvements in number of operations per instruction
  - Requires domain specific architectures



Domain-Specific Architecture (cont.)

- Computers will be much more heterogeneous[异构]
  - Standard processors to run conventional large programs
    - E.g., operating system
  - Domain-specific processors doing only a narrow range of tasks
    - But they do them extremely well
- DSA opportunities[机遇]
  - Preceding architecture from the past may not be a good match to some domains
    - E.g., caches are excellenet general-purpose architectures but not necessarily for DSAs
  - Domain-specific algorithms are almost always for small compute-intensive kernels of larger systems
    - DSAs should focus on the subset and not plan to run the entire program

DSA Challenges[挑战]

- Architects must expand their areas of expertise
  - Must now learn application domains and algorithms
- *Nonrecurring engineering* (NRE) costs[一次性工程成本]
  - Find a target whose demand is large enough to justify allocating dedicated silicon on an SOC or even a custom chip
    - The costs are amortized over the number of chips manufactured, so unlikely to make economic sense if you need only 1000 chips
  - For smaller volume applications, use reconfigurable chips such as FPGAs
    - Several different applications may reuse the same reconfigurable hardware to amortize costs
    - However, the hardware is less efficient than custom chips, so the gains from FPGAs are more modest
- Port software[移植软件]
  - Programming languages and compilers

DSA Design Guidelines[设计准则]

- Why guidelines?
  - Lead to increased area and energy efficiency
  - Provide two valuable bonus effects
    - Lead to simpiler designs, reducing the cost of NRE of DSAs
    - For user-facing apps, better match the 99th-percentile response-time deadlines

Guideline	TPU	Catapult	Crest	Pixel Visual Core
Design target	Data center ASIC	Data center FPGA	Data center ASIC	PMD ASIC/SOC IP
1. Dedicated memories	24 MiB Unified Buffer, 4 MiB Accumulators	Varies	N.A.	Per core: 128 KiB line buffer, 64 KiB P.E. memory
2. Larger arithmetic unit	65,536 Multiply-accumulators	Varies	N.A.	Per core: 256 Multiply-accumulators (512 ALUs)
3. Easy parallelism	Single-threaded, SIMD, in-order	SIMD, MISD	N.A.	MPMD, SIMD, VLIW
4. Smaller data size	8-Bit, 16-bit integer	8-Bit, 16-bit integer 32-bit Fl. Pt.	21-bit Fl. Pt.	8-bit, 16-bit, 32-bit integer
5. Domain-specific lang.	TensorFlow	Verilog	TensorFlow	Halide/TensorFlow

DSA Design Guidelines (cont.)

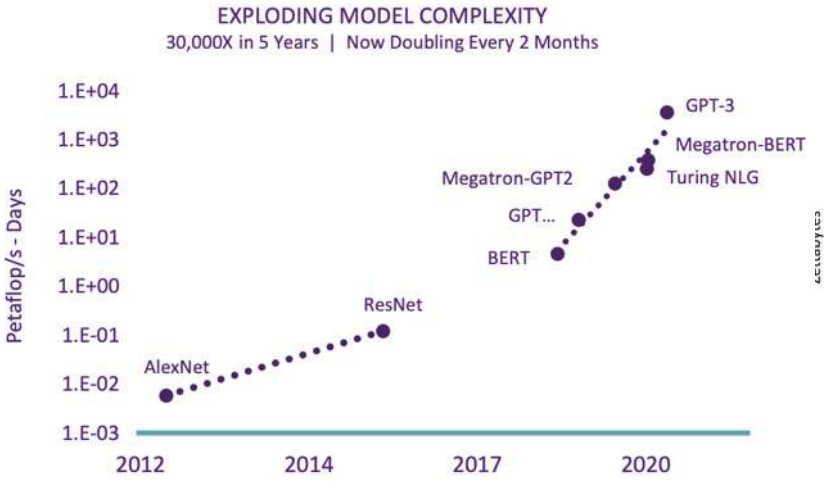
- *Use dedicated memories to minimize the distance over which data is moved*
  - Hardware cache → software-controlled scratchpad
    - Compiler writers and programmers of DSAs understand their domain
    - Software-controlled memories are much more energy efficient
- *Invest the resources saved from dropping advanced u-arch optimizations into more arithmetic units or bigger memories*
  - Owing to the superior understanding of the execution of programs
- *Use the easiest form of parallelism that matches the domain*
  - Target domains for DSAs almost always have inherent parallelism
    - How to utilize that parallelism and how to expose it to the software?
  - Design the DSA around the natural granularity of the parallelism and expose that parallelism simply in the programming model
    - SIMD > MIMD, VLIW > OoO

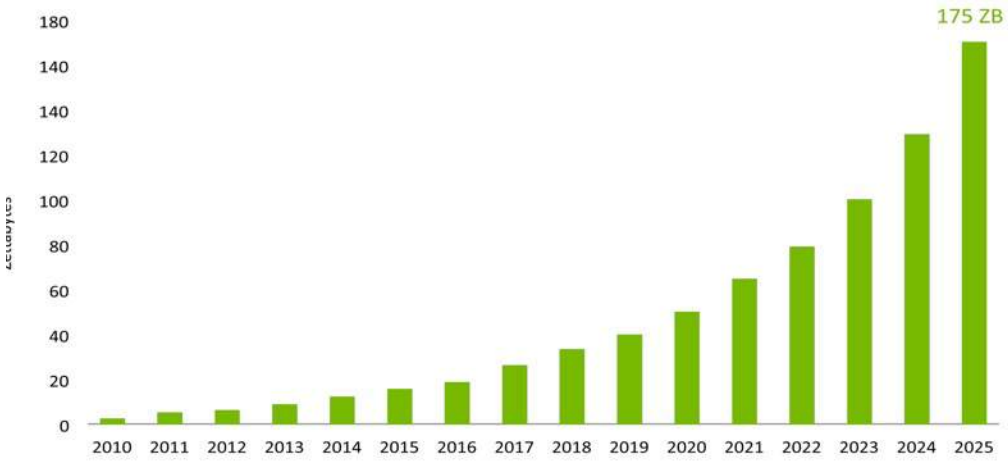
DSA Design Guidelines (cont.)

- *Reduce data size and type to the simplest needed for the domain*
  - Apps in many domains are typically memory-bound, using narrower data types helps increase the effective memory bandwidth and on-chip memory utilizations
  - Narrower and simpler data also enable to pack more arithmetic units into the same chip area
- *Use a domain-specific programming language to port code to the DSA*
  - WRONG: you new arch is so attractive that programmers will rewrite their code just for you hw
  - Fortunately, domain-specific languages were popular even before architects' switched attentions
    - Halide for vision processing, TensorFlow for DNNs

The Trend

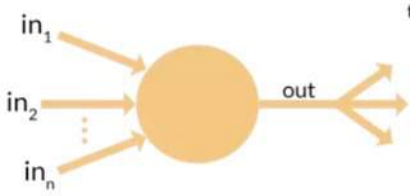
- The ABC of AI: Algorithm + Big-data + Computing





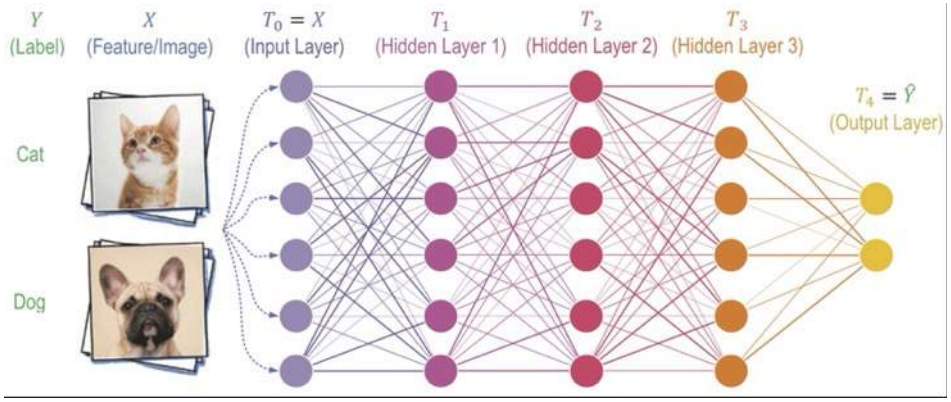
Example Domain

- Deep neural networks (DNNs)
  - Revolutioning many areas of computing today
  - Are applicable to a wide range of problems
    - So, a DNN-specific arch can be reused for solutions in speech, vision, language, translation, search ranking, and many more areas
- DNN structure
  - Inspired by neuron of the brain
    - Each neuron simply computes the sum over a set of products of weights or parameters and data values
      - E.g., pixels for image-processing
  - The sum is then put through a nonlinear function to determine its output
    - E.g., f(x) = max(x, 0) --- *rectified linear unit* (ReLU)
    - Output is called *activation*
      - The output of the neuron that has been "activated"



DNNs

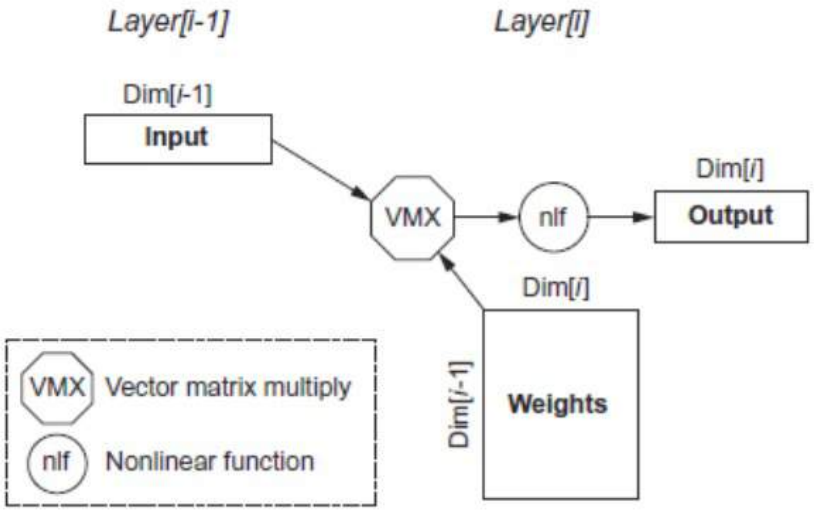
- Most practitioners will choose an existing design
  - Topology
  - Data type
- Training (learning)[训练]
  - Calculate weights using backpropagation algorithm
  - Supervised learning: stochastic graduate descent
- Inference[推理]
  - Use neural network for classification



Name	DNN layers	Weights	Operations/Weight
MLP0	5	20M	200
MLP1	4	5M	168
LSTM0	58	52M	64
LSTM1	56	34M	96
CNN0	16	8M	2888
CNN1	89	100M	1750

Multilayer Perceptron[多层感知机]

- Feed-forward neural networks
  - The units are arranged into a graph without any cycles
    - so that all the computation can be done sequentially
  - Fully connected: every unit in one layer is connected to every unit in the next layer
- MLP is just a vector matrix multiply of the input vector times the weights array

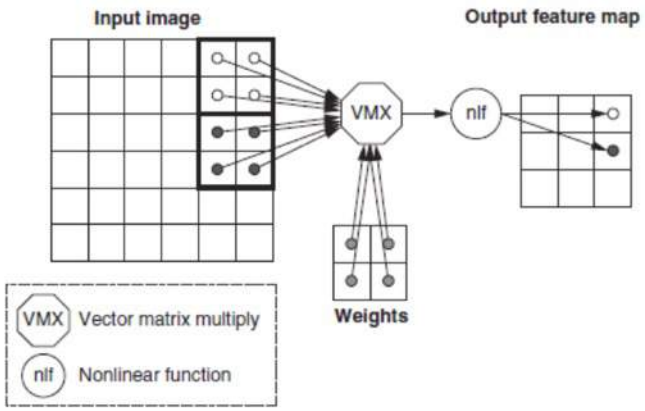


Parameters:

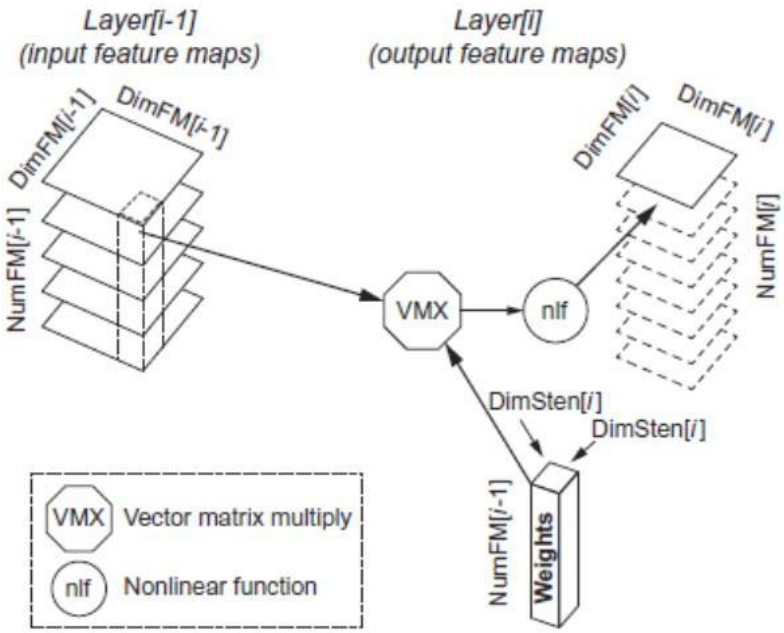
Dim[i]: number of neurons
Dim[i-1]: dimension of input vector
Number of weights: Dim[i-1] x Dim[i]
Operations: 2 x Dim[i-1] x Dim[i]
Operations/weight: 2

Convolutional Neural Network[卷积]

- CNNs are widely used for computer vision applications
- Each layer raises the level of abstraction
  - Lines → corners → shapes → …
- **Feature map**: a set of 2D maps produced by each neural layer
  - Each cell is identifying one feature in the area of the input
- **Stencil computation**: uses neighboring cells in a fixed pattern to update all the elements of an array



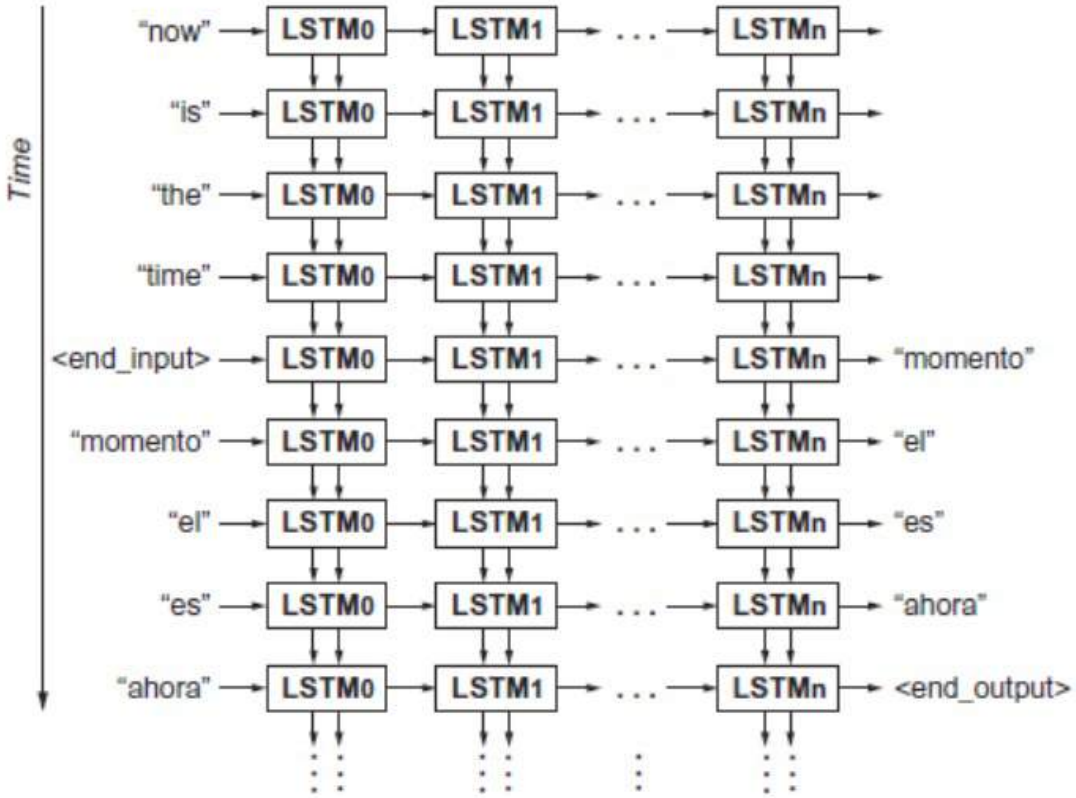
Convolutional Neural Network (cont.)



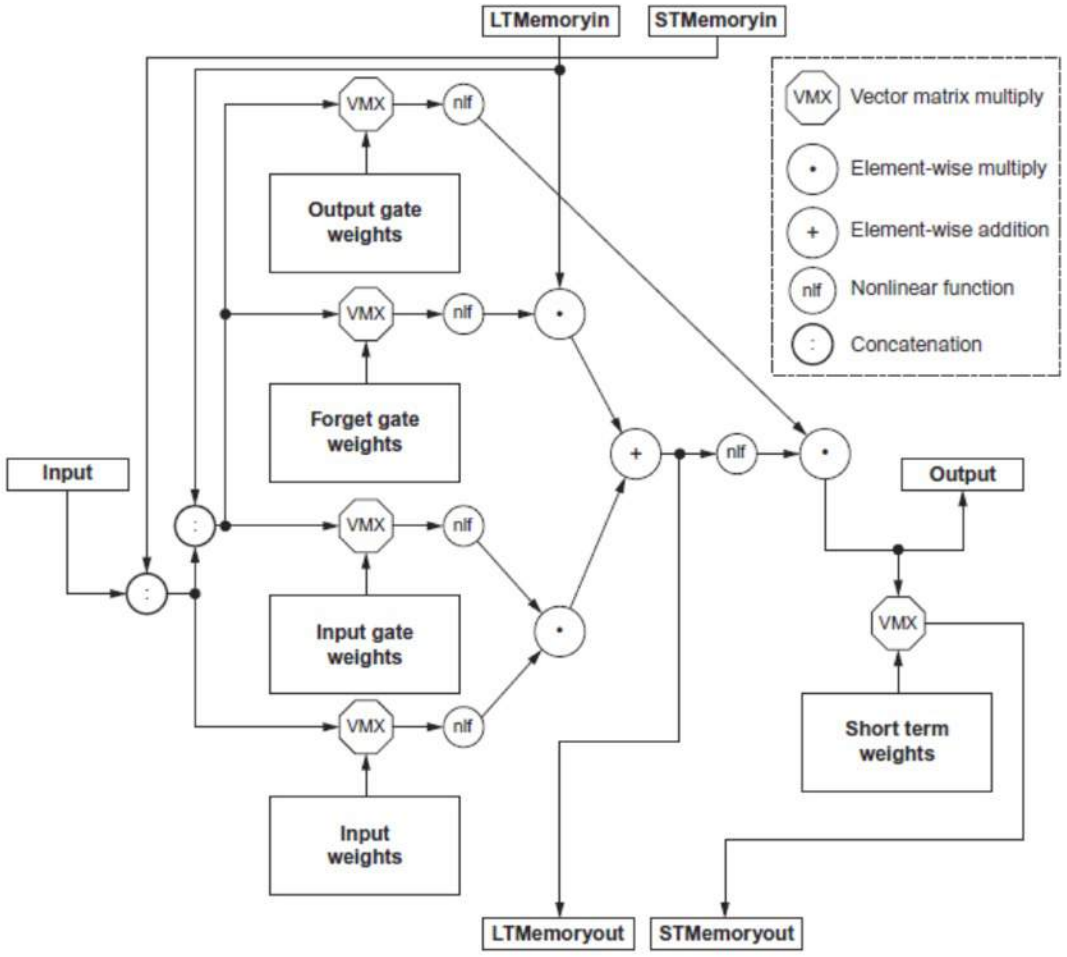
- Parameters:
  - DimFM[i-1]: Dimension of the (square) input Feature Map
  - DimFM[i]: Dimension of the (square) output Feature Map
  - DimSten[i]: Dimension of the (square) stencil
  - NumFM[i-1]: Number of input Feature Maps
  - NumFM[i]: Number of output Feature Maps
  - Number of neurons: NumFM[i] x DimFM[i]^2
  - Number of weights per output Feature Map: NumFM[i-1] x DimSten[i]^2
  - Total number of weights per layer: NumFM[i] x Number of weights per output Feature Map
  - Number of operations per output Feature Map: 2 x DimFM[i]^2 x Number of weights per output Feature Map
  - Total number of operations per layer: NumFM[i] x Number of operations per output Feature Map = 2 x DimFM[i]^2 x NumFM[i] x Number of weights per output Feature Map = 2 x DimFM[i]^2 x Total number of weights per layer
  - Operations/Weight: 2 x DimFM[i]^2

Recurrent Neural Network[循环]

- Popular for speech recognition on language translations
- RNNs can remember facts
  - Long short-term memory (LSTM) network



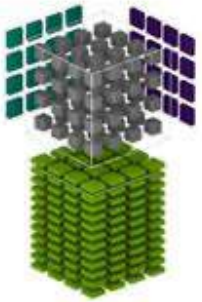
Recurrent Neural Network (cont.)

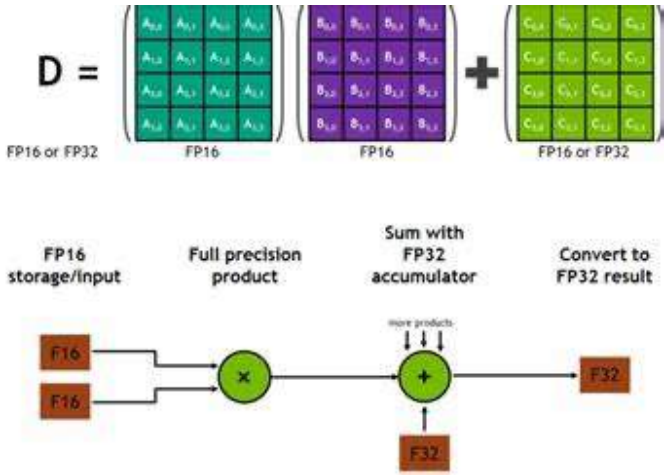


- Parameters:
  - Number of weights per cell: 3 x (3 x Dim x Dim)+(2 x Dim x Dim) + (1 x Dim x Dim) = 12 x Dim^2
  - Number of operations for the 5 vector-matrix multiplies per cell: 2 x Number of weights per cell = 24 x Dim^2
  - Number of operations for the 3 element-wise multiplies and 1 addition (vectors are all the size of the output): 4 x Dim
  - Total number of operations per cell (5 vector-matrix multiplies and the 4 element-wise operations): 24 x Dim^2 + 4 x Dim
  - Operations/Weight: ~2

Example Domain: DNNs

- Batches[批]
  - Reuse weights once fetched from memory across multiple inputs
    - Increases operational intensity
- Quantization[量化]
  - Numerical precision is less important for DNNs than for many applications
    - Use 8- or 16-bit fixed point
- Summary: need the following kernels
  - Matrix-vector multiply
  - Matrix-matrix multiply
  - Stencil
  - ReLU
  - Sigmoid
  - Hyperbolic tangent





Tensor Processing Unit (TPU)

- Google's first custom ASIC DSA for WSCs
  - Its domain is the inference phase of DNNs
  - It is programmed using the TensorFlow framework
  - The first TPU was been deployed in 2015
    - Originated as far back as 2006, to improve perf by 10x over GPUs

TPU v1

Launched in 2015

Inference only

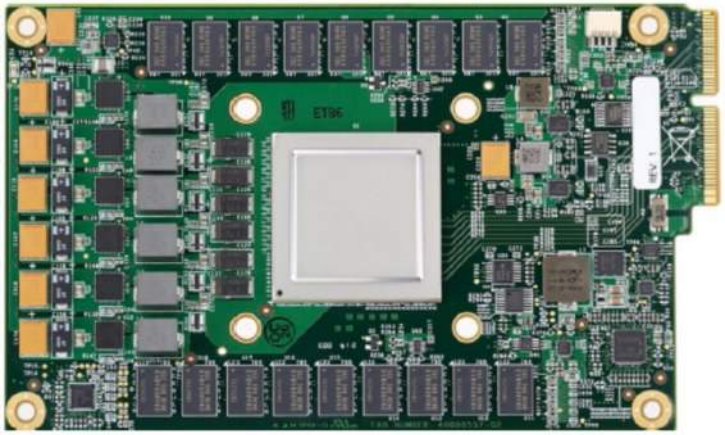
TPU v2

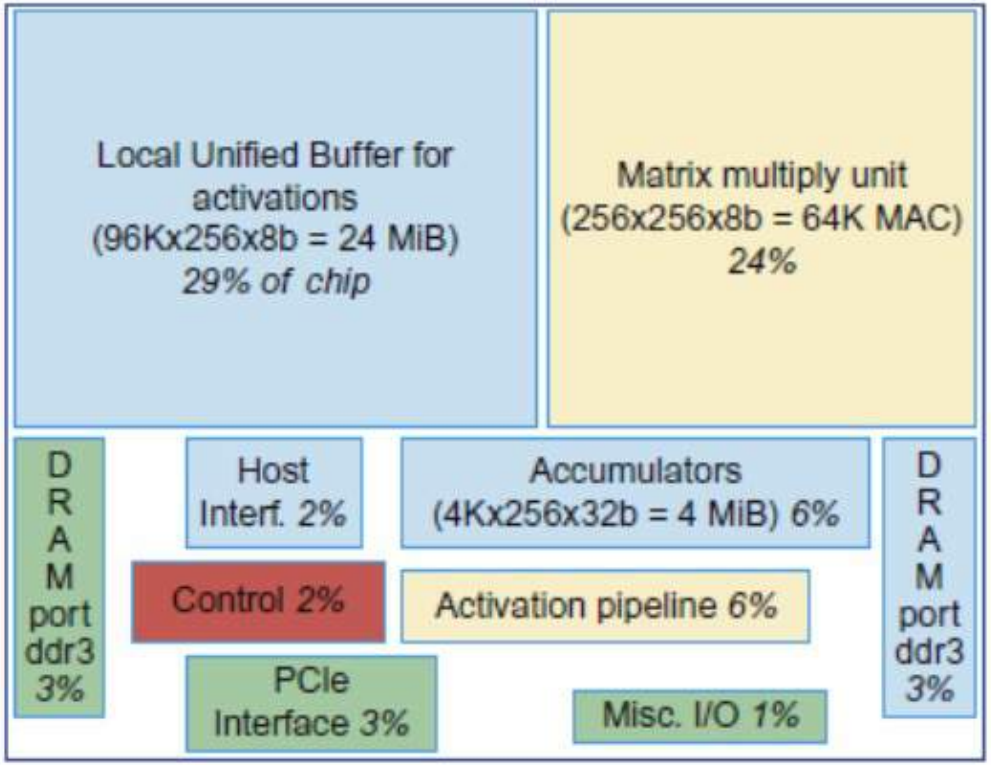
Launched in 2017

Inference and training

TPU Chip Overview

- TPU chip is half the size of the other chips
  - 28 nm process with a die size ≤ 331 mm
  - This is partially due to simplification of control logic
- Floor plan of TPU die
  - 50%+ on arithmetic and memory

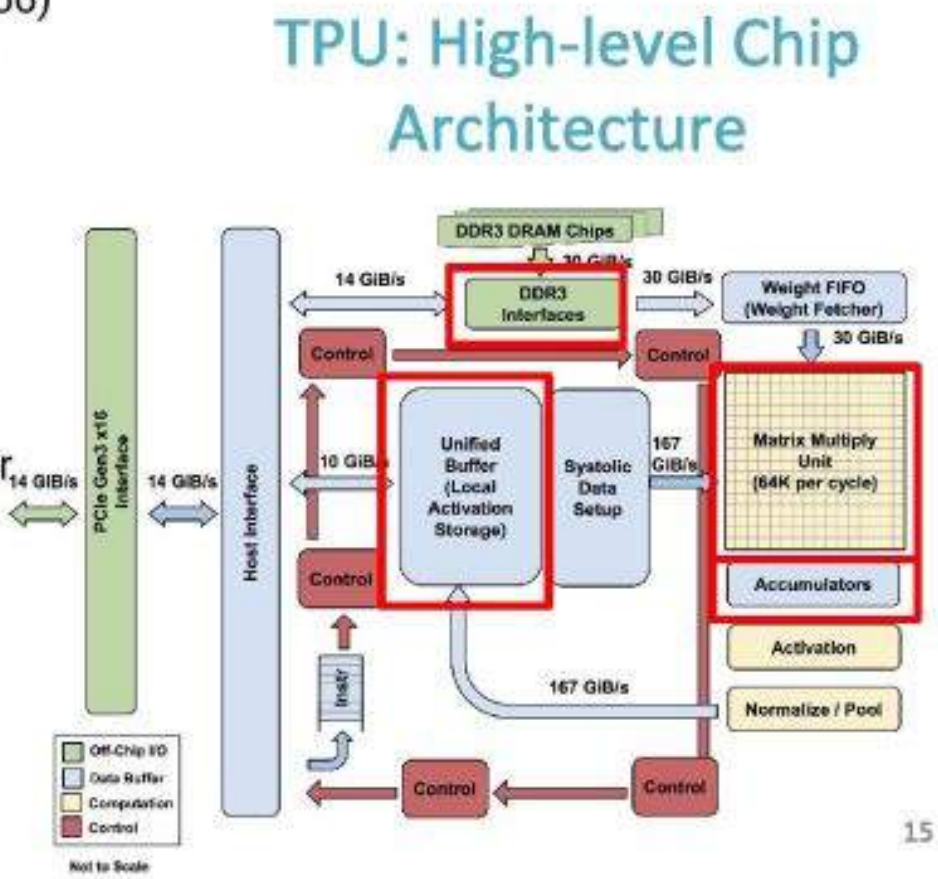




TPU Architecture[架构]

- A coprocessor on the PCIe I/O bus
- A large software-managed on-chip memory

- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 700M
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory
- 24 MiB of on-chip Unified Buffer (activation memory)
- 3.5X as much on-chip memory vs GPU
- Two 2133MHz DDR3 DRAM channels
- 8 GiB of off-chip weight DRAM memory



https://www.anandtech.com/show/11749/hot-chips-google-tpu-performance-analysis-live-blog-3pm-pt-10pm-utc

TPU ISA[指令]

- The host CPU sends TPU instructions over the PCIe bus into an instruction buffer[指令发送]
  - TPU has no PC, and it has no branch instructions
  - 5 main (CISC) instructions (11 in total)
    - Other: alternate host memory read/write, set configuration, two versions of synchronization, intrrupt host, debug-tag, nop and halt
- Instruction execution[指令执行]
  - Average clock cycles per instruction: > 10
  - 4-stage overlapped execution, 1 instruction type/stage
    - Execute other instructions while matrix multiplier busy
- Complexity in software[软件复杂性]
  - No branches, in-order issue
  - SW controlled buffers, SW controlled pipeline synchronization

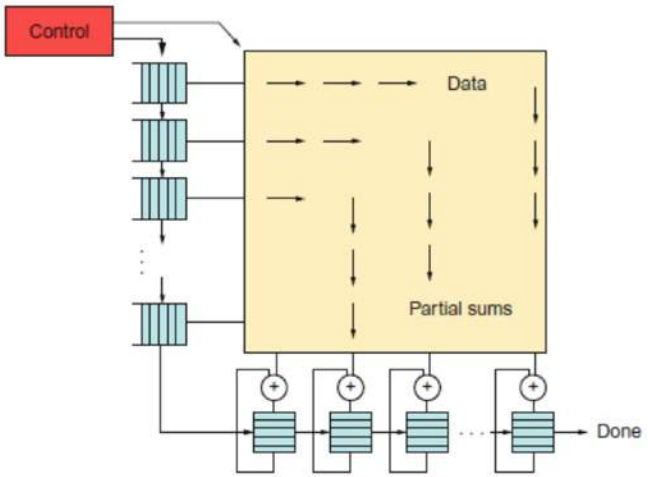
https://www.anandtech.com/show/11749/hot-chips-google-tpu-performance-analysis-live-blog-3pm-pt-10pm-utc

TPU ISA (cont.)

- **Read_Host_Memory**
  - Reads memory from the CPU memory into the unified buffer
- **Read_Weights**
  - Reads weights from the Weight Memory into the Weight FIFO as input to the Matrix Unit
- **MatrixMultiply/Convolve**
  - Perform a matrix-matrix multiply, a vector-matrix multiply, an element-wise matrix multiply, an element-wise vector multiply, or a convolution from the Unified Buffer into the accumulators
    - Takes a variable-sized B*256 input, multiplies it by a 256x256 constant input, and produces a B*256 output, taking B pipelined cycles to complete
- **Activate**
  - Computes activation function
- **Write_Host_Memory**
  - Writes data from unified buffer into host memory

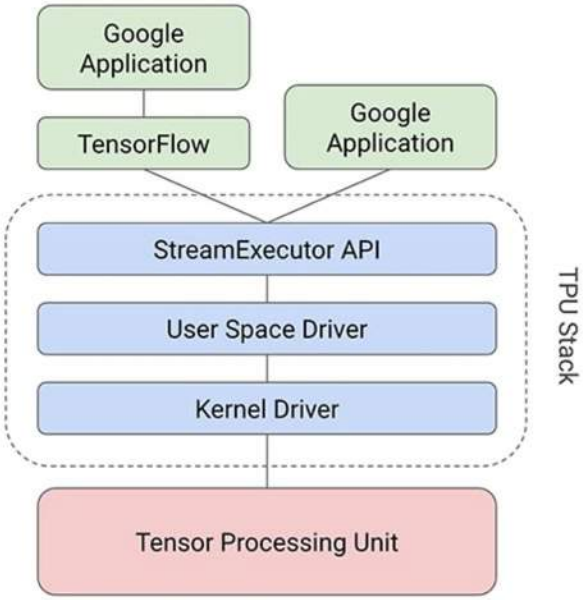
TPU Microarchitecture[微架构]

- The *u*-arch philosophy of TPU is to keep the Matrix Multiply Unit busy
  - Hide the execution of the other insts by overlapping with the MatrixMultiply inst
    - Each of the other 4 insts have separate execution hw
- Problem: energy/time for repeated SRAM accesses of matrix multiply
  - Solution: “Systolic execution” to compute data on the fly in buffers by pipelining control and data[脉动阵列执行]



TPU Software[软件]

- Software stack had to be compatible with CPUs/GPUs[兼容]
  - So that applications could be ported quickly
  - The portion of the app run on the TPU is typically written using TensorFlow and is compiled into an API that can run on CPUs/GPUs
- Like GPUs, the TPU stack is split into[分层]
  - **Kernel Driver**: lightweight and handles only memory management and interrupts
    - Designed for long-term stability
  - **Use Space Driver**: changes frequently, and handles the following
    - Sets up and controls TPU execution
    - Reformats data into TPU order
    - Translates API calls into TPU insts and turns them into an app binary



How TPU Follows the Guidelines

- *Use dedicated memories*
  - 24 MB dedicated buffer, 4 MB accumulator buffers
- *Invest resources in arithmetic units and dedicated memories*
  - 60% of the memory and 250X the arithmetic units of a server-class CPU
- *Use the easiest form of parallelism that matches the domain*
  - Exploits 2D SIMD parallelism
- *Reduce the data size and type needed for the domain*
  - Primarily uses 8-bit integers
- *Use a domain-specific programming language*
  - Uses TensorFlow

TPU Performance[性能]

- Compare using six benchmarks
  - Representing 95% of TPU inference workload in Google data center in 2016
  - Typically written in TensorFlow, pretty short (100-1500 LOCs)
- Chips/servers being compared
  - CPU server: Intel 18-core, dual-socket Haswell; host server for GPUs/TPUs
  - GPU accelerator: Nvidia K80

Inference Datacenter Workload (95%)

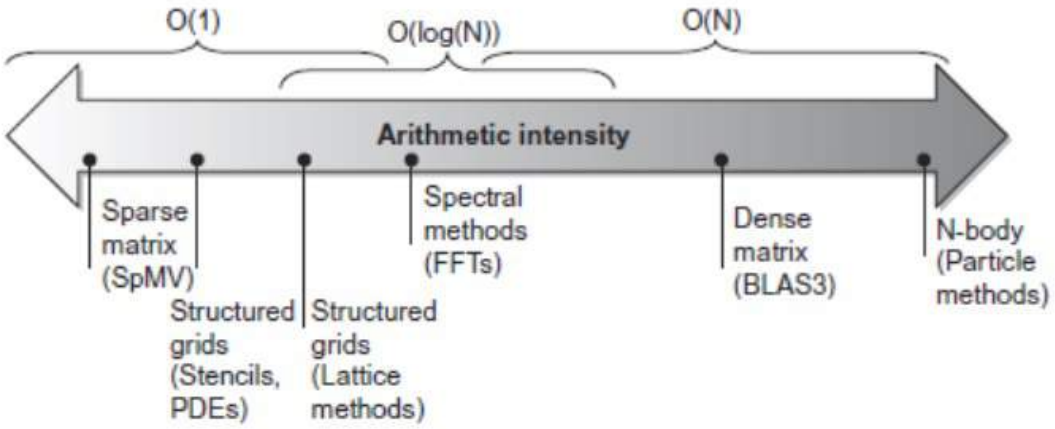
Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% Deployed*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	0.1k	5				5	ReLU	20M	200	200	61%
MLP1	1k	4				4	ReLU	5M	168	168	
LSTM0	1k	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1k		16			16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	89	ReLU	100M	1750	32	

Roofline Performance Model[屋顶线]

- The roofline model was introduced in 2009
  - Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM
- It provides an easy way to get performance bounds for compute and memory bandwidth bound computations
- It relies on the concept of **Computational Intensity** (CI)
  - Sometimes also called Arithmetic or Operational Intensity
- The model provides a relatively simple way for performance estimates based on the computational kernel and hardware characteristics
  - Performance [GF/s] = function (hardware and software characteristics)

Roofline Performance Model(cont.)

- Basic idea
  - Plot peak FP throughput as a function of arithmetic intensity
  - Ties together FP performance and memory performance for a target machine
- Arithmetic intensity[运算密度]
  - Ratio of FP operations per byte of memory accessed
    - (total #FP operations for a program) / (total data bytes transferred to main memory during program execution)



Arithmetic Intensity

- $A.I. = \frac{W}{Q}$ (FLOP/Byte)
  - W: amount of work / i.e floating point operations required
  - Q: memory transfer / i.e access from DRAM to lowest level cache
- Examples

```
for (i = 0; i < N; ++i)
    z[i] = x[i]+y[i]
```

1 ADD
2 (8 byte) loads
1 (8 byte) write
AI = 1 / (2*8 + 8) = 1/24

```
for (i = 0; i < N; ++i)
    z[i] = x[i]+y[i]*x[i]
```

1 ADD
1 MUL
2 (8 byte) loads
1 (8 byte) write
AI = 2 / (2*8 + 8) = 1/12

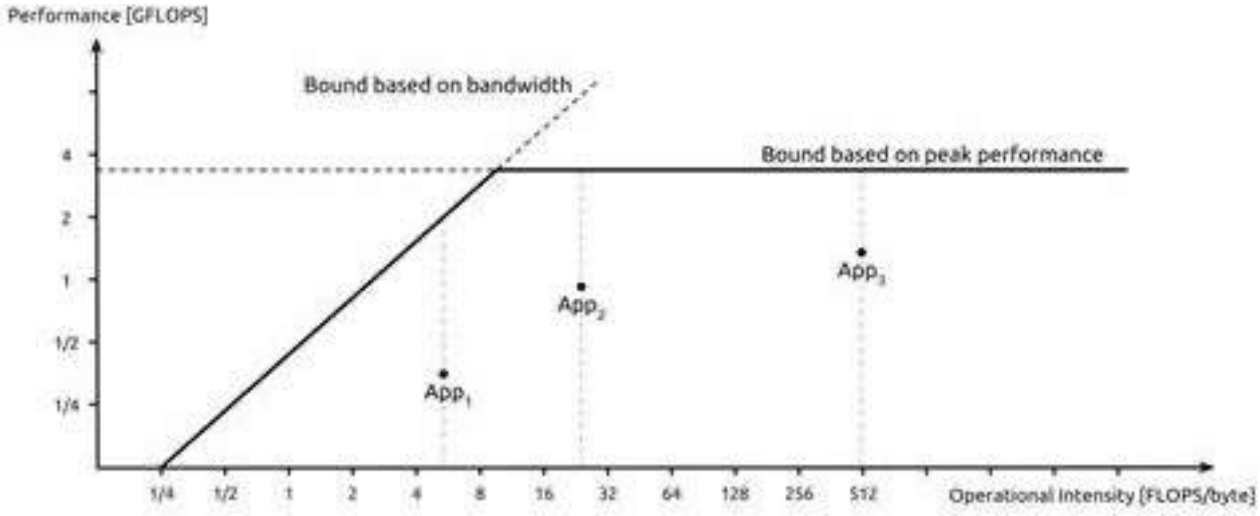
Example

```
float in[N], out[N];
for (int i=1; i<N-1; i++)
    out[i] = in[i-1]-2*in[i]+in[i+1];
```

- Amount of FLOPS: 3(N-2)
  - For every i: out[I] = in[i-1]-2*in[i]+in[i+1] → 3 flop
  - Loop over: for (int i=1; i<N-1; i++) → (N-2) repetitions
- Memory accesses Q: depends on cache size
  - No cache (read directly from slow memory) → every data accessed is counted
    - 4 accesses x (N-2) repetitions x 4 bytes → A.I. = 3/16
  - Perfect cache (infinite sized cache) → data is read & written only once
    - 2 accesses x (N-2) repetitions x 4 bytes → A.I. = 3/8

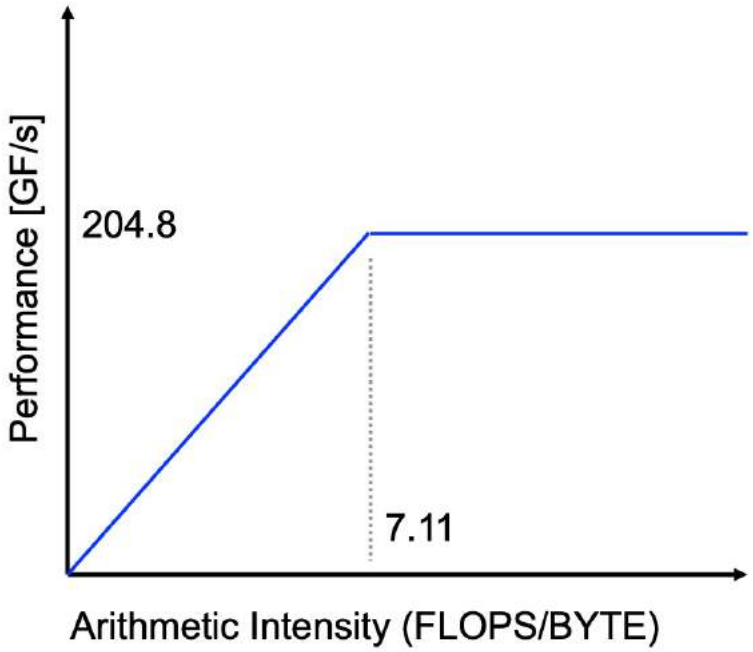
Roofline Analysis

- "**Roofline**" sets an upper bound on perf of a kernel depending on its arithmetic intensity
  - Think of arithmetic intensity as a pole that hits the roof
    - Hits the flat part: perf is computationally limited
    - Hits the slanted part: perf is ultimately limited by memory bandwidth
- **Ridge point**: the diagonal and horizontal roofs meet
  - Far to right: only very intensive kernels can achieve max perf
  - Far to left: almost any kernel can potentially hit max perf



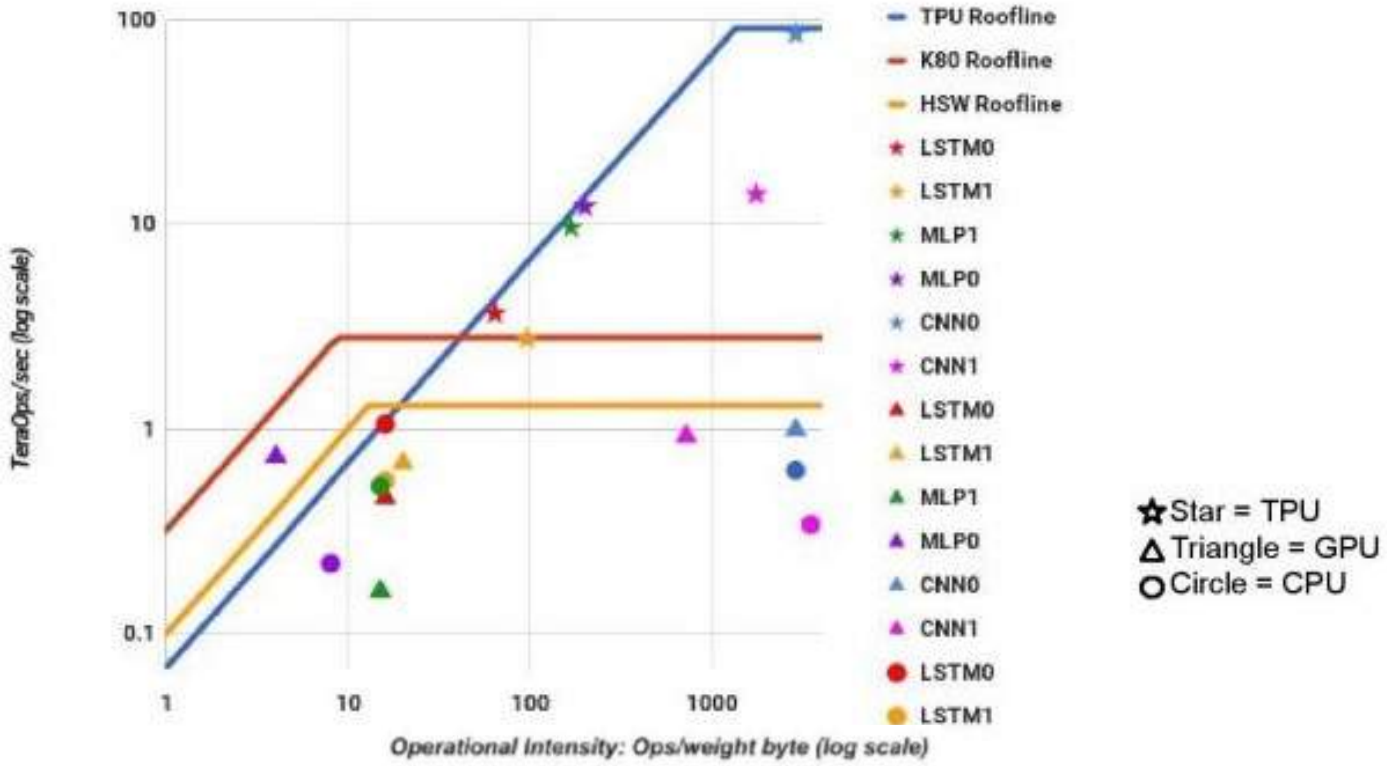
Example

- Consider: for (i = 0; i < N; ++i) y[i] = a*x[i]+y[i]
  - For each “i” :
    - 1 addition, 1 multiplication
    - 2 loads of 8 bytes each
    - 1 store
- Execution on BlueGene/Q
  - Peak 204.8 GFLOP/node
- Performance estimates:
  - AI = 2/(3*8) = 1 / 12 1/12 < 7.11 → limited area on the Roofline plot
  - 7.11/(1/12)= 85.32
  - 204.8 / 85.32 = 2.4 GF/s



TPU Roofline Performance

- TPU: its ridge point is far to the right at 1350
  - CNN1 is much further below its Roofline than the other DNNs
    - Waiting for weights to be loaded into the matrix unit
  - Ridge point comparison:
    - CPU: 13, GPU: 9 → better balanced, but perf a lot lower



Cost-Performance

- Cost metric: performance per watt
  - "Total": includes the power consumed by the host CPU server when calculating perf/watt for the GPU and TPU
  - "Incremental": subtracts the host CPU power from the total
- Total: GPU is 2.1x CPU, TPU is 34x CPU
- Incremental: TPU is 83x CPU, 29x GPU

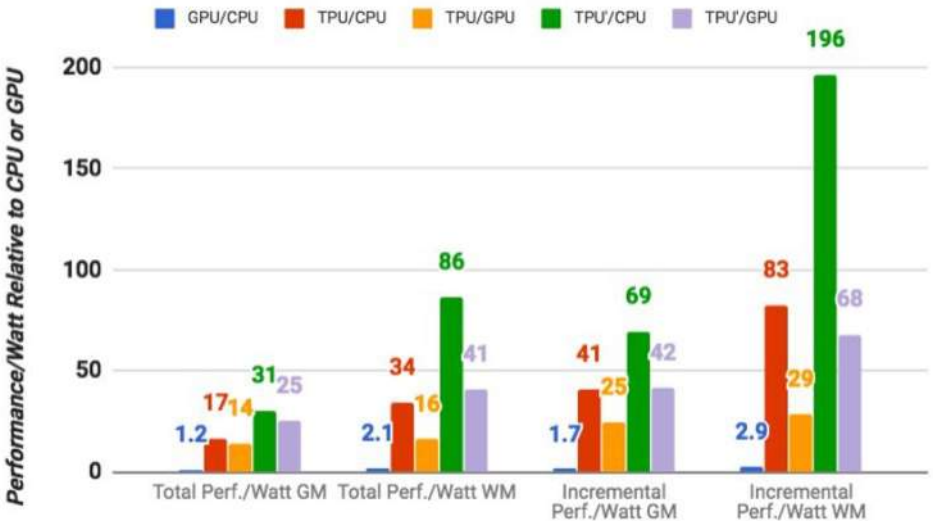
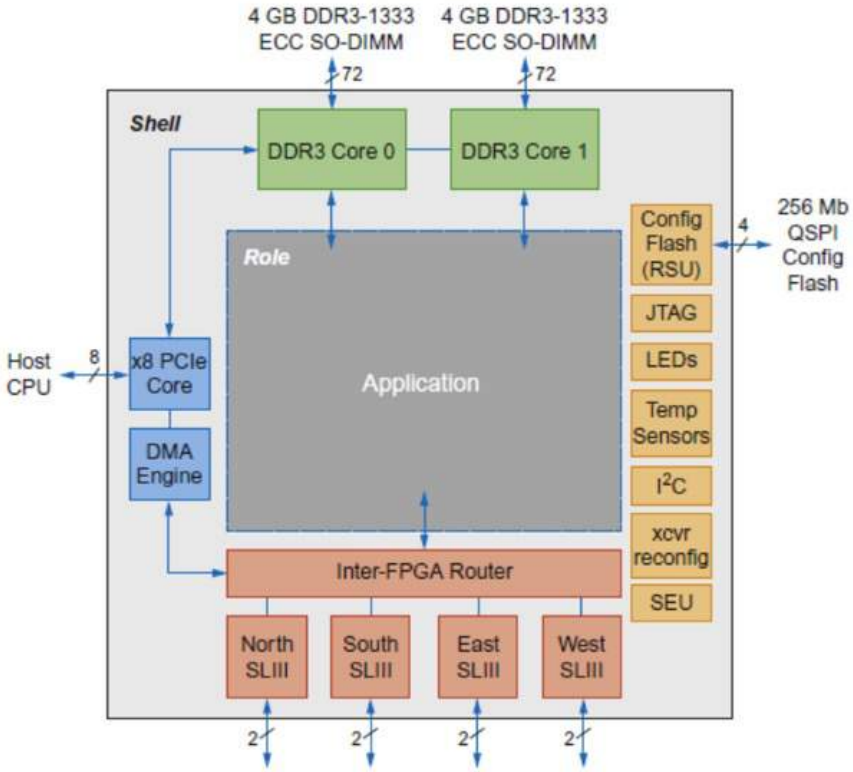


Figure 9. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU (Sec. 7). The green bar shows its ratio to the CPU server and the lavender bar shows its relation to the GPU server. Total includes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

https://www.extremetech.com/computing/247199-googles-dedicated-tensorflow-processor-tpu-makes-hash-intel-nvidia-inference-workloads

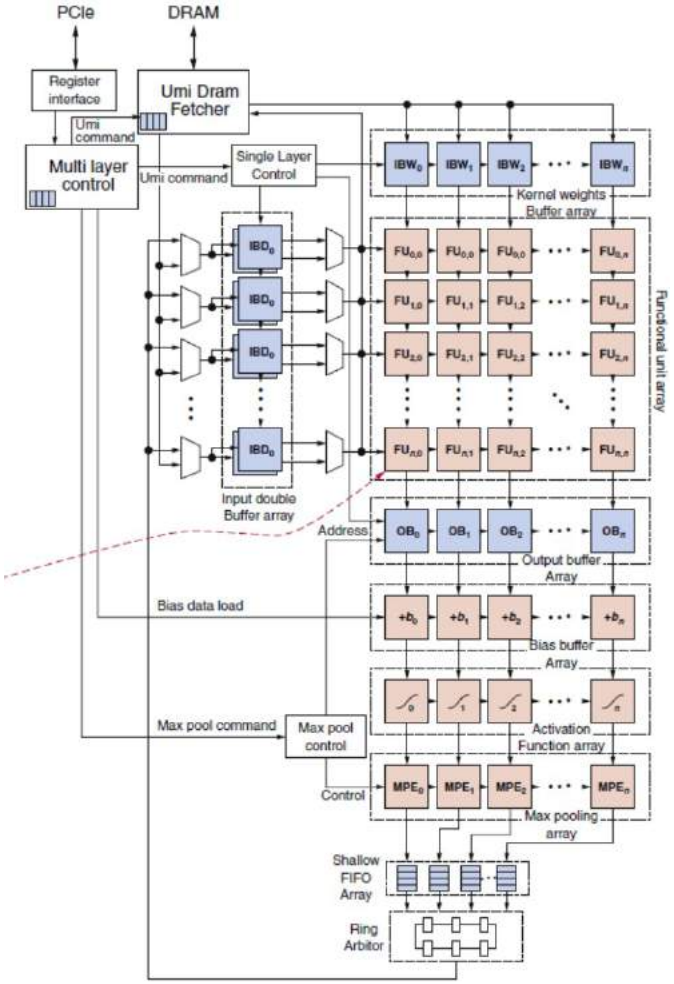
Microsoft Catapult

- Needed to be general purpose and power efficient
  - Uses FPGA PCIe board with dedicated 20 Gbps network in 6 x 8 torus
  - Each of the 48 servers in half the rack has a Catapult board
  - Limited to 25 watts
  - 32 MB Flash memory
  - Two banks of DDR3-1600 (11 GB/s) and 8 GB DRAM
  - FPGA (unconfigured) has 3962 18-bit ALUs and 5 MB of on-chip memory
  - Programmed in Verilog RTL
  - Shell is 23% of the FPGA

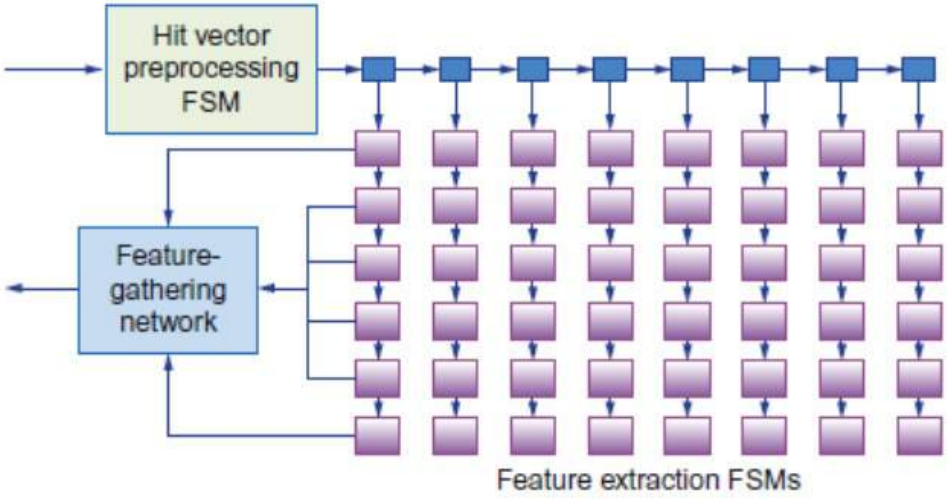


Catapult Applications

- The processing element (PE) of the CNN Accelerator for Catapult



- The architecture of FPGA implementation of the Feature Extraction stage in search acceleration



How Catapult Follows the Guidelines

- *Use dedicated memories*
  - 5 MB dedicated memory
- *Invest resources in arithmetic units and dedicated memories*
  - 3926 ALUs
- *Use the easiest form of parallelism that matches the domain*
  - 2D SIMD for CNN, MISD parallelism for search scoring
- *Reduce the data size and type needed for the domain*
  - Uses mixture of 8-bit integers and 64-bit floating-point
- *Use a domain-specific programming language*
  - Uses Verilog RTL; Microsoft did not follow this guideline