Computer Architecture

计 算 机 体 系 结 构

第23讲：Domain Specific Arch (2)

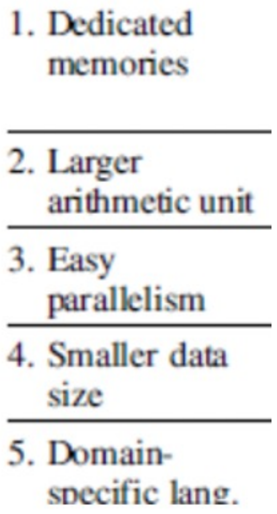
张献伟

xianweiz.github.io

DCS3013, 12/21/2022

Review

- Interconnection network (ICN)
  - ICN can be the system bottleneck, should be well designed
  - Topologies: Bus, Xbar, Multistage, Star, Linear, Mesh, Tree
  - Compute Express Link (CXL)
    - Unified, coherent memory space
    - High speed, low latency
- Domain specific architecture (DSA)
  - Past architecture targets general-purpose code
  - DSA is needed to provide 100sX performance
    - Hardware software co-design
  - DSA design guidelines
    - Lead to simpler designs
    - To achieve higher area and energy efficiency



1. Dedicated memories
2. Larger arithmetic unit
3. Easy parallelism
4. Smaller data size
5. Domain-specific lang.

DSA Design Guidelines

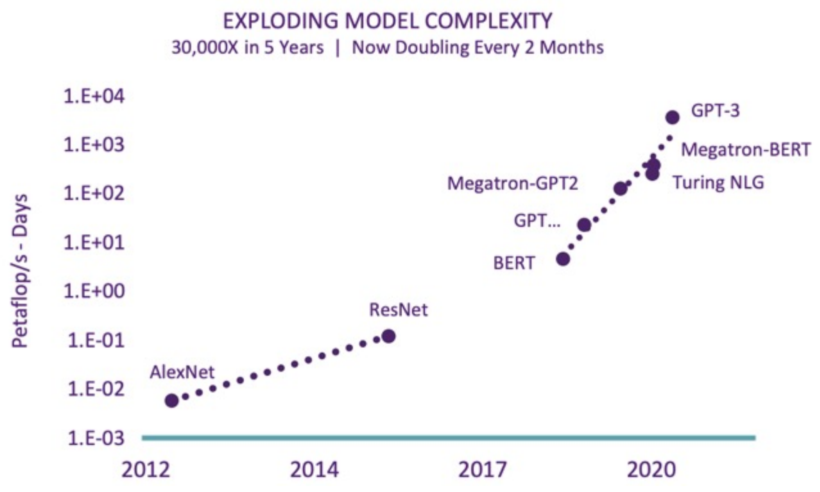
- *Use dedicated memories to minimize the distance over which data is moved*
  - Hardware cache → software-controlled scratchpad
    - Compiler writers and programmers of DSAs understand their domain
    - Software-controlled memories are much more energy efficient
- *Invest the resources saved from 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 (i.e., DLP > TLP), VLIW > OoO

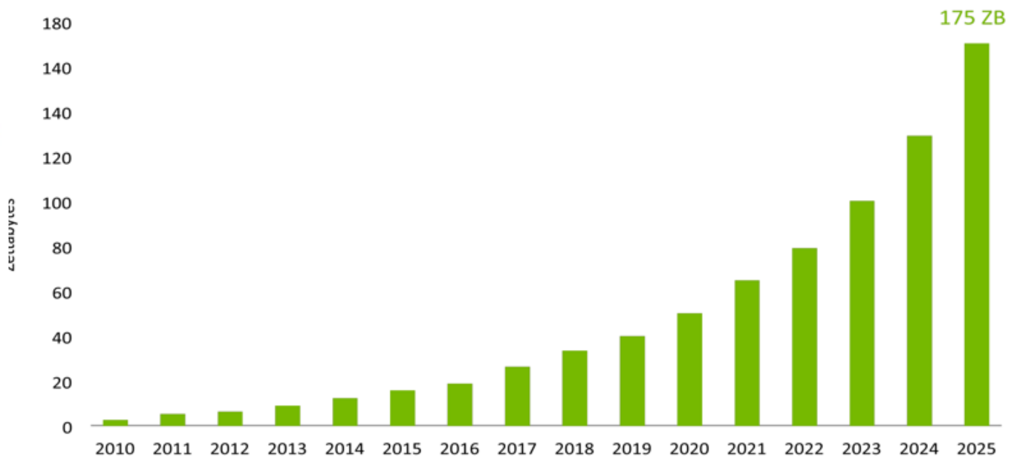
DSA Design Guidelines (cont.)

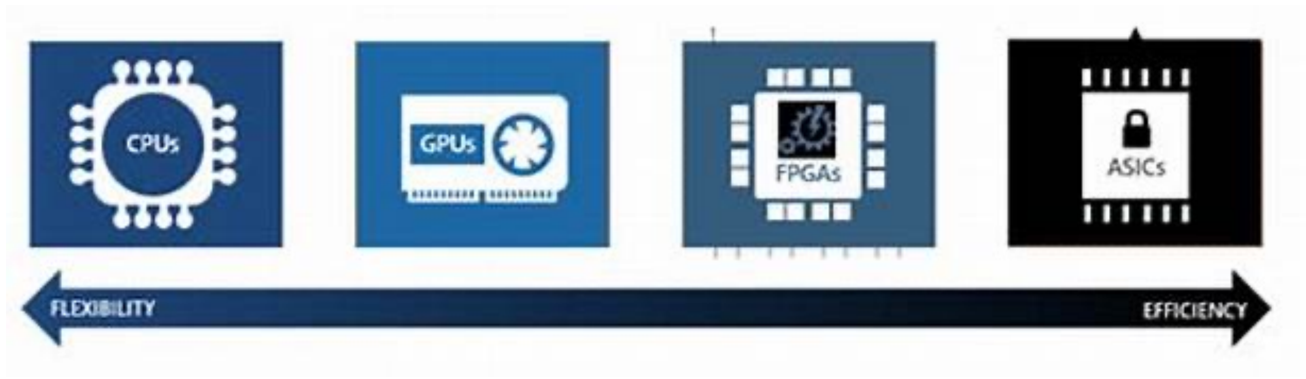
- *Reduce data size and type to the simplest needed for the domain*
  - Apps in many domains are typically memory-bound, using narrower data types helps increase the effective memory bandwidth and on-chip memory utilizations
  - Narrower and simpler data also enable to pack more arithmetic units into 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



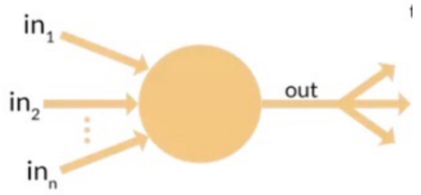




HPC
BIG DATA
AI

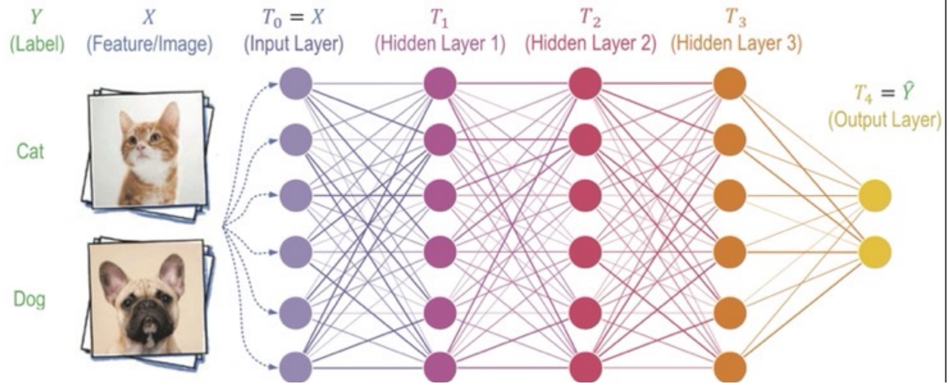
Example Domain

- 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

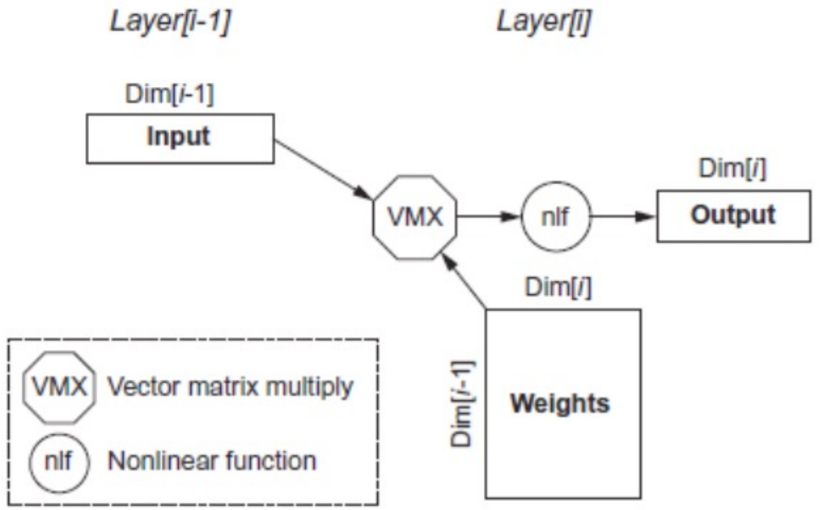
- Most practitioners will choose an existing design
  - Topology
  - Data type
- Training (learning)[训练]
  - Calculate weights using backpropagation algorithm
  - Supervised learning: stochastic gradient descent[随机梯度下降]
- Inference[推理]
  - Use neural network for classification



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, the original DNNs, is just a vector matrix multiply of the input vector times the weights array

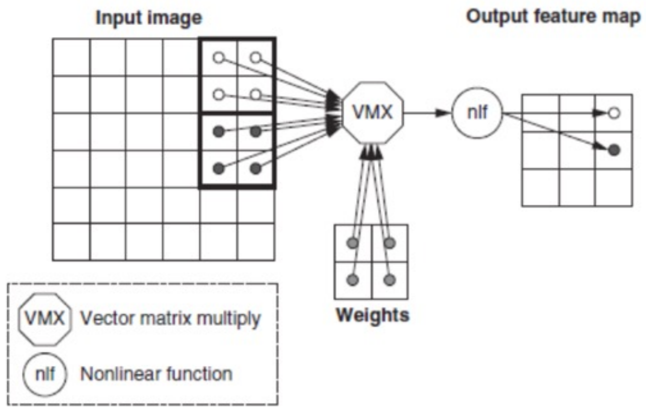


Parameters:

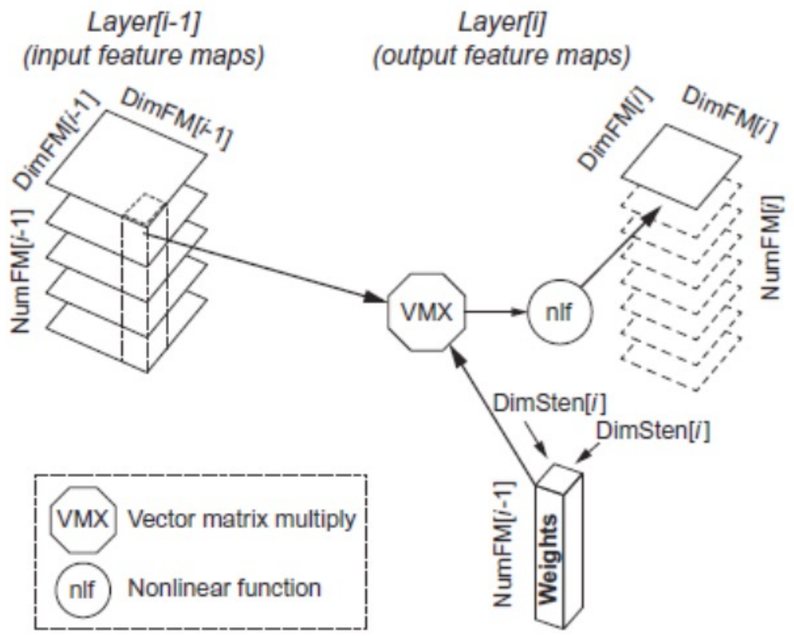
Dim[i]: number of neurons
Dim[i-1]: dimension of input vector
Number of weights: Dim[i-1] x Dim[i]
Operations: 2 x Dim[i-1] x Dim[i]
Operations/weight: 2

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**[特征图]: a set of 2D maps produced by each neural layer
  - Each cell is identifying one feature in the area of the input
- **Stencil computation**[模版计算]: uses neighboring cells in a fixed pattern to update all the elements of an array
  - 循环运算：遍历计算区域，每个位置均执行相同的计算操作



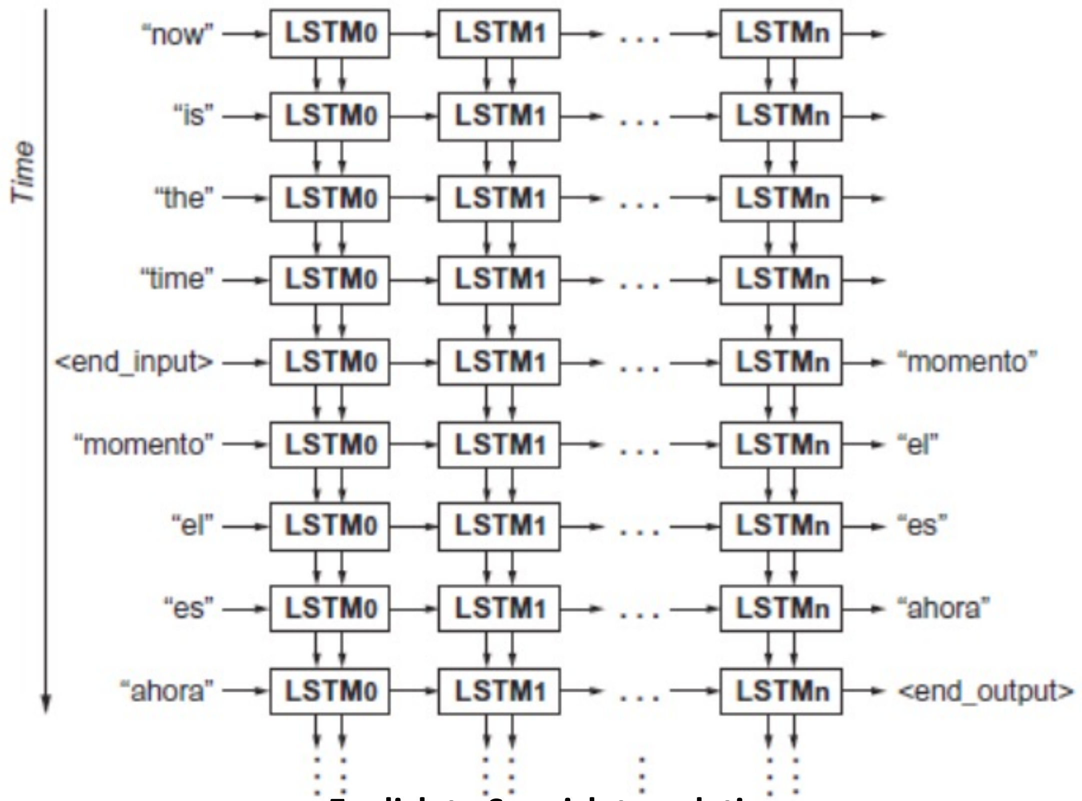
Convolutional Neural Network (cont.)



- 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

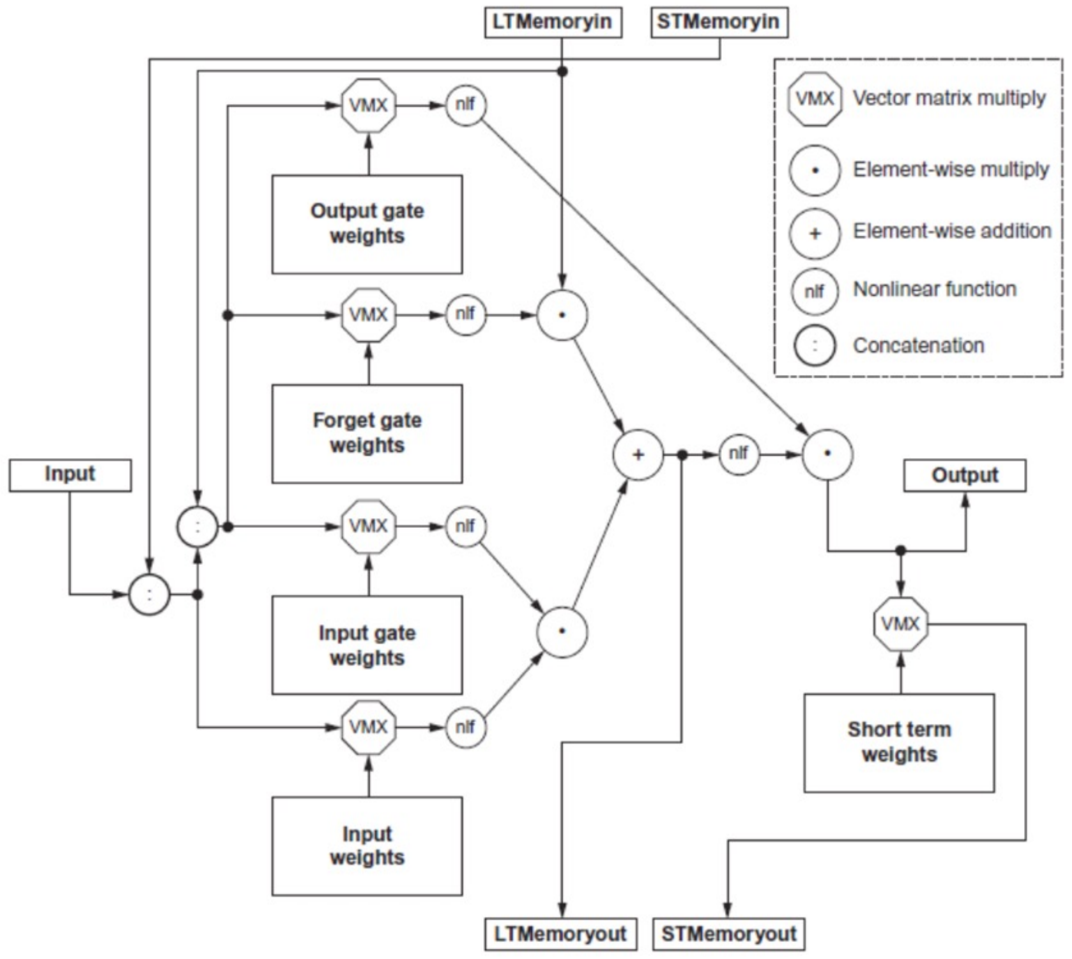
Recurrent Neural Network[循环]

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



English to Spanish translation

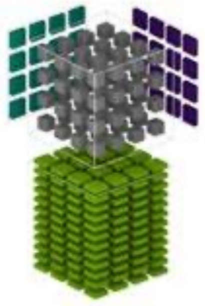
Recurrent Neural Network (cont.)

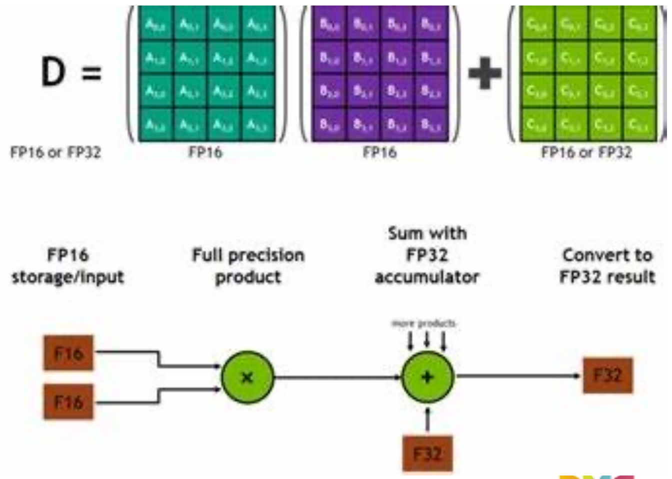


- Parameters:
  - Number of weights per cell: 3 x (3 x Dim x Dim)+(2 x Dim x Dim) + (1 x Dim x Dim) = 12 x Dim^2
  - Number of operations for the 5 vector-matrix multiplies per cell: 2 x Number of weights per cell = 24 x Dim^2
  - Number of operations for the 3 element-wise multiplies and 1 addition (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[双曲正切]





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

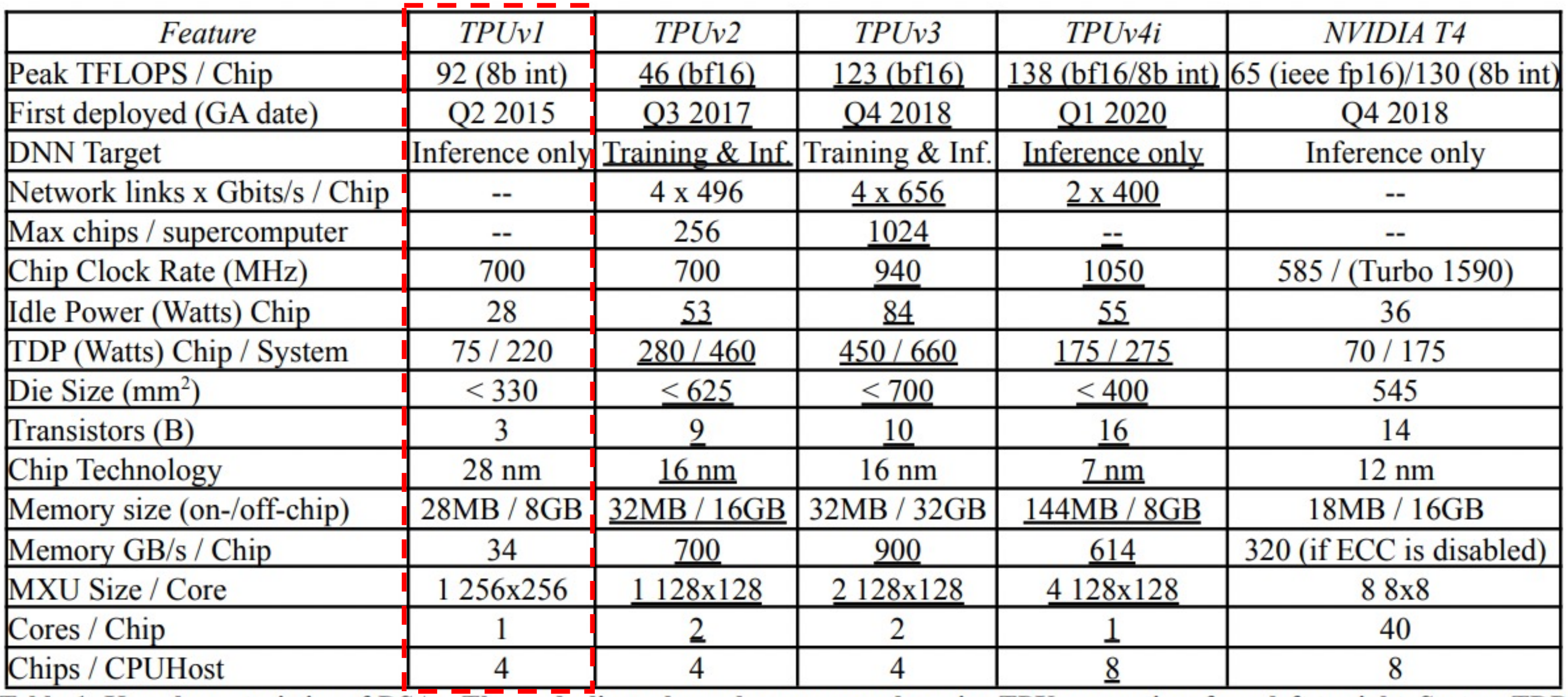
Inference only

TPU v2

Launched in 2017

Inference and training

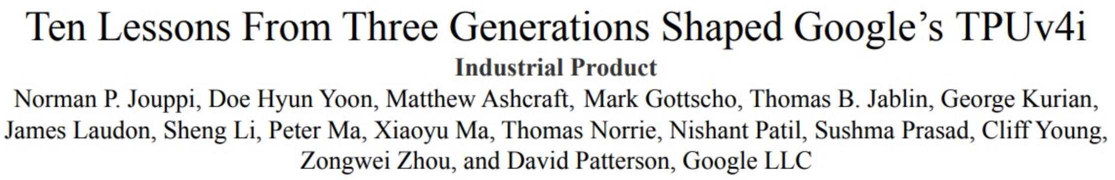
Tensor Processing Unit (cont'd)



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

Table 1. Key characteristics of DSAs. The underlines show changes over the prior TPU generation, from left to right. System TDP includes power for the DSA memory system plus its share of the server host power, e.g., add host TDP/8 for 8 DSAs per host.

2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)



Ten Lessons From Three Generations Shaped Google's TPUv4i

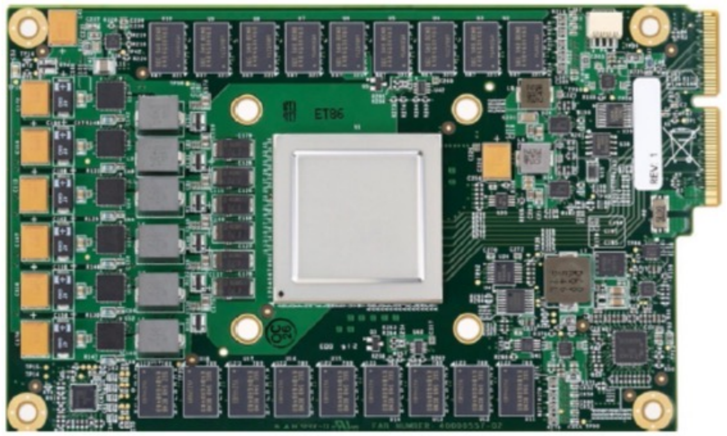
Industrial Product

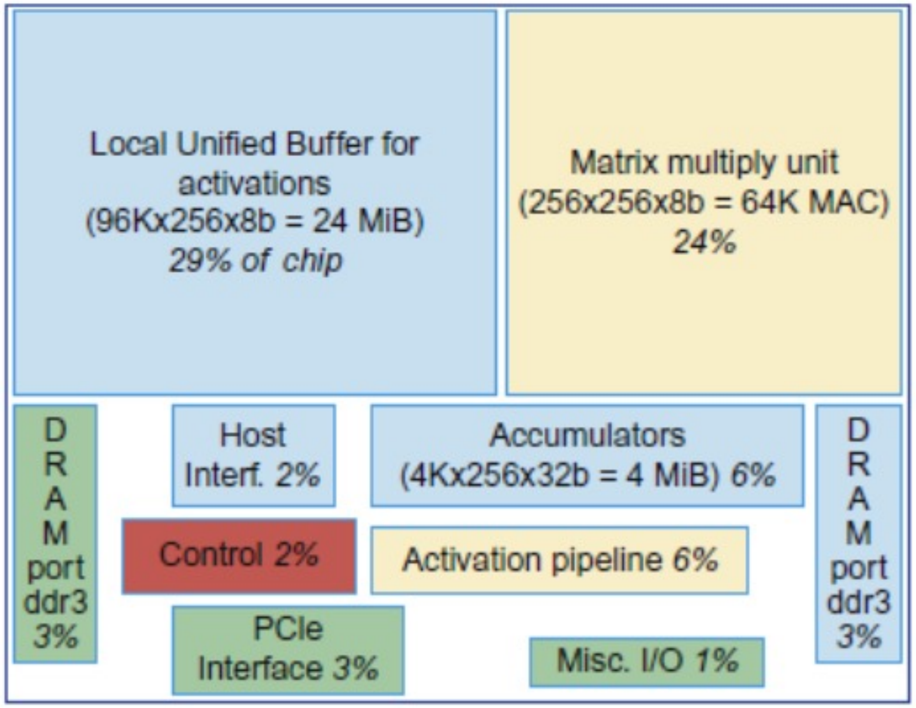
Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson, Google LLC

https://www.gwern.net/docs/ai/scaling/hardware/2021-jouppi.pdf

TPU Chip Overview

- TPU chip is half the size of the other chips
  - 28 nm process with a die size ≤ 331 mm^2
  - This is partially due to simplification of control logic
- Floor plan of TPU die
  - 50%+ on arithmetic and memory

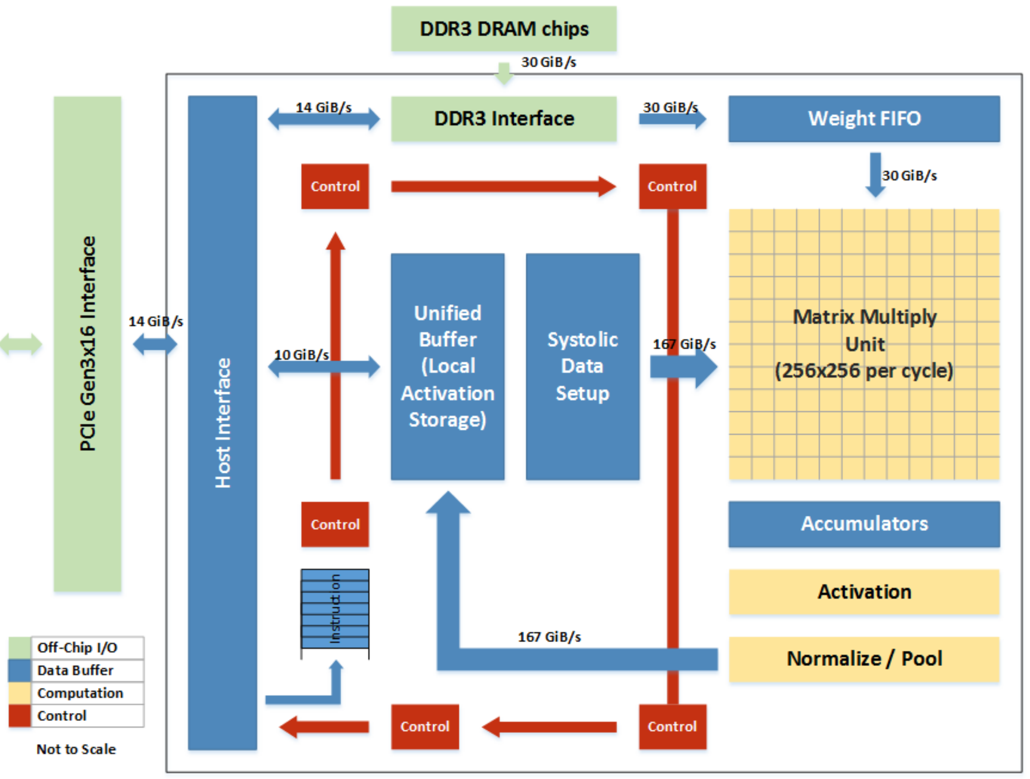



Local Unified Buffer for activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix multiply unit (256x256x8b = 64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

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



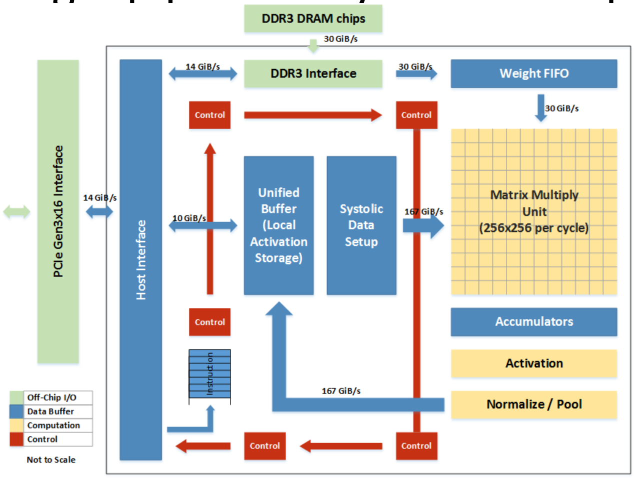
https://www.anandtech.com/show/11749/hot-chips-google-tpu-performance-analysis-live-blog-3pm-pt-10pm-utc

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 6: alternate host memory read/write, set configuration, two versions of synchronization, interrupt 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

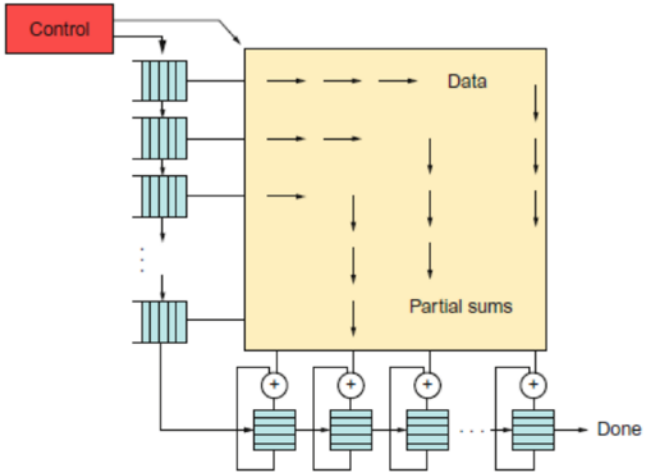
TPU ISA (cont.)

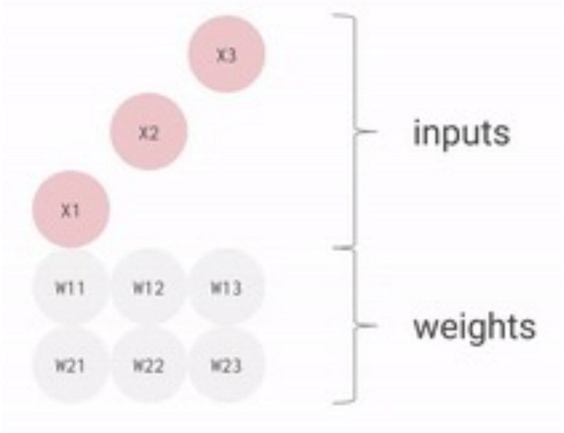
- **Read_Host_Memory**
  - Reads data from the CPU memory into the unified buffer
- **Read_Weights**
  - Reads weights from the Weight Memory (DDR3) into the Weight FIFO as input to the Matrix Unit
- **MatrixMultiply/Convolve**
  - Perform a matrix-matrix multiply, a vector-matrix multiply, an element-wise 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

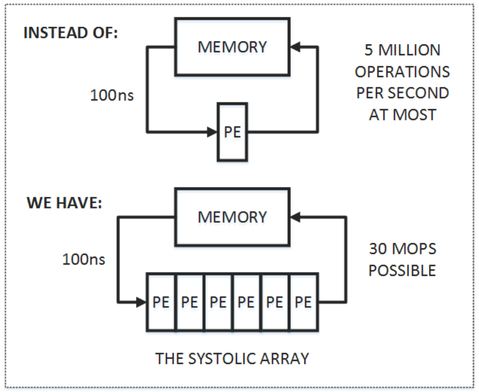


TPU Microarchitecture[微架构]

- 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]



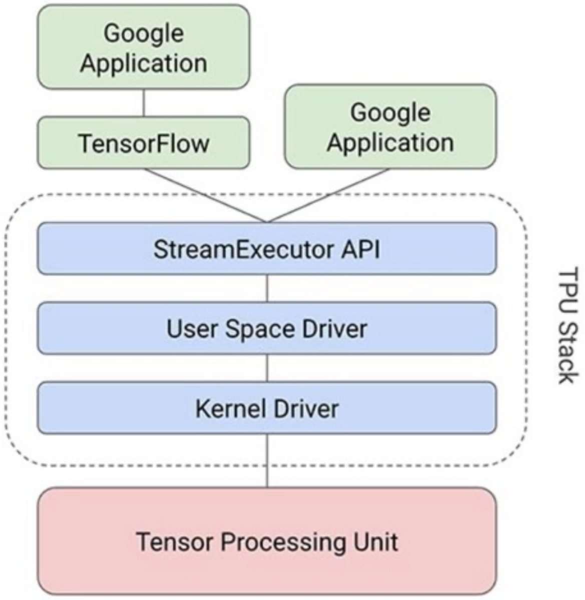




脉动阵列 - 因Google TPU获得新生，
https://zhuanlan.zhihu.com/p/26522315

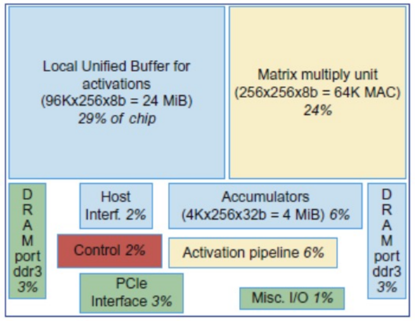
TPU Software[软件]

- 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

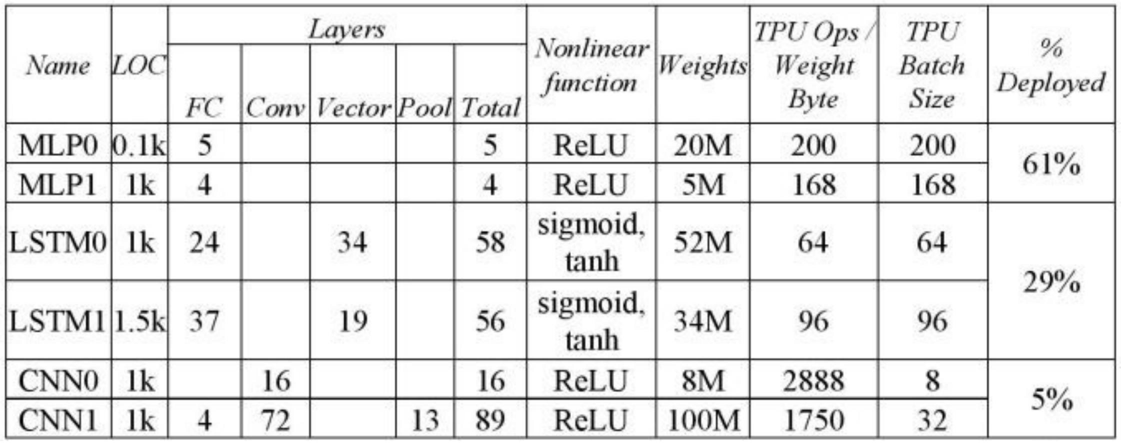


Local Unified Buffer for activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix multiply unit (256x256x8b = 64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

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	

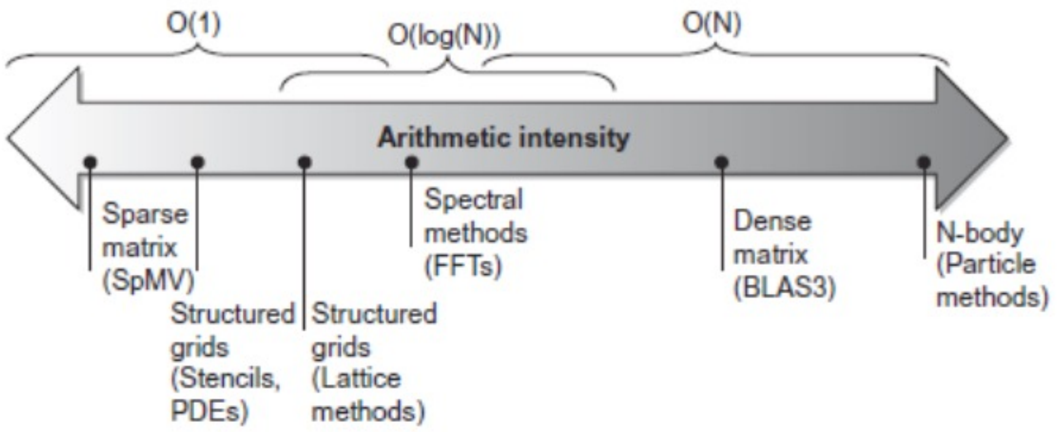
Roofline Performance Model[屋顶线]

- 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)

https://www.dam.brown.edu/people/lgrinb/APMA2821/Lectures_2015/APMA2821H-L_roof_line_model.pdf

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[运算密度]

- $A.I. = \frac{W}{Q}$ (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]
```

$\Rightarrow$

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]
```

$\Rightarrow$

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

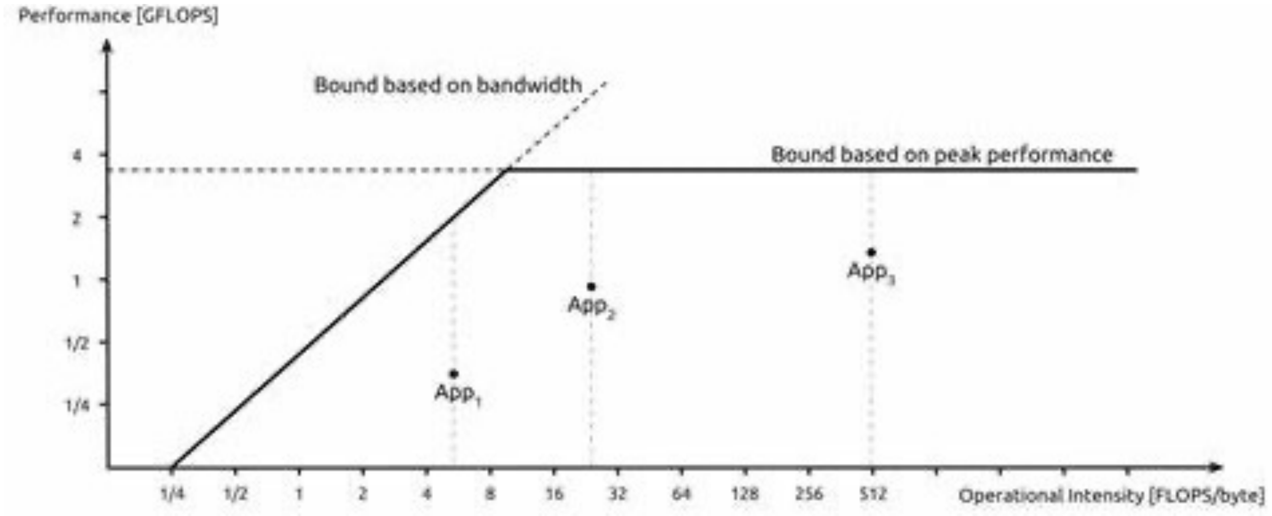
Example

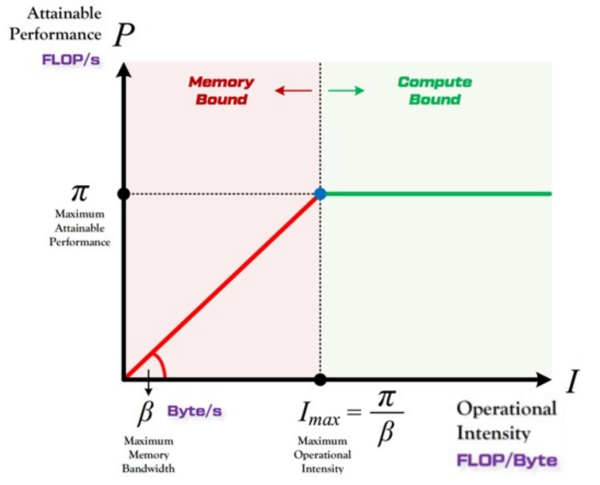
```
float in[N], out[N];
for (int i=1; i<N-1; i++)
    out[i] = in[i-1]-2*in[i]+in[i+1];
```

- 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

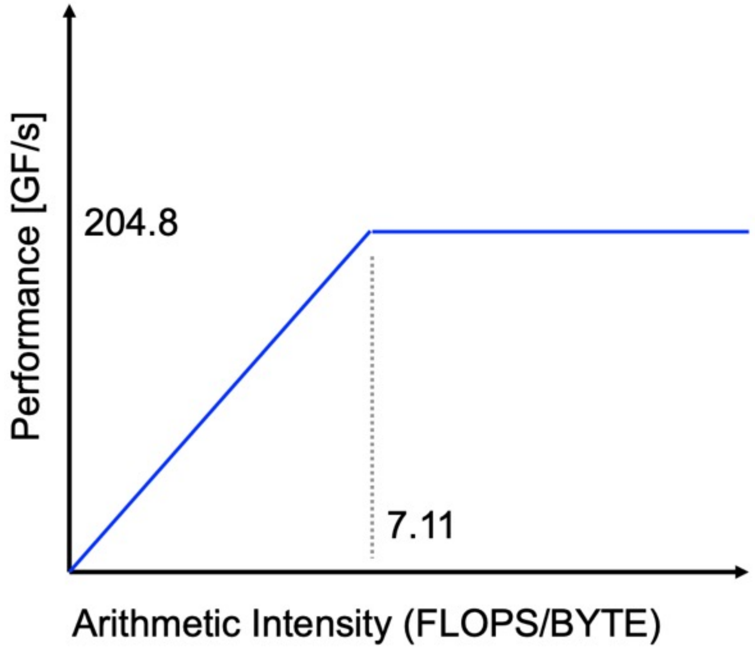
- ”**Roofline**” sets an upper bound on perf of a kernel depending on its arithmetic intensity
  - Think of arithmetic intensity as a pole that hits the roof
    - Hits the flat part: perf is computationally limited
    - Hits the slanted part: perf is ultimately limited by memory bandwidth
- **Ridge 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


Performance [GF/s]
Arithmetic Intensity (FLOPS/BYTE)