Computer Architecture and Operating Systems
Lecture 16: Domain-specific architectures. Tensor Processing Unit.

Andrei Tatarnikov

atatarnikov@hse.ru
@andrewt0301

Motivation

- Modern performance tuning techniques:
  - Deep memory hierarchy
  - Wide SIMD units
  - Deep pipelines
  - Branch prediction
  - Out-of-order execution
  - Speculative prefetching
  - Multithreading
  - Multiprocessing
- Further improvement:
  - Domain-specific architectures

Guidelines for DSAs

- Use **dedicated memories** to minimize data movement
- Invest resources into **more arithmetic units** or bigger memories
- Use the easiest form of **parallelism** that matches the domain
- Reduce **data size and type** to the simplest needed for the domain
- Use a **domain-specific** programming language

Deep Neural Networks

- Inpired by neuron of the brain
- Computes non-linear “activation” function of the weighted sum of input values
- Neurons arranged in layers
- Most practitioners will choose an existing design
  - Topology
  - Data type
- Training (learning):
  - Calculate weights using backpropagation algorithm
  - Supervised learning: stochastic graduate descent
- Inference: use neural network for classification

DNN Statistics

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

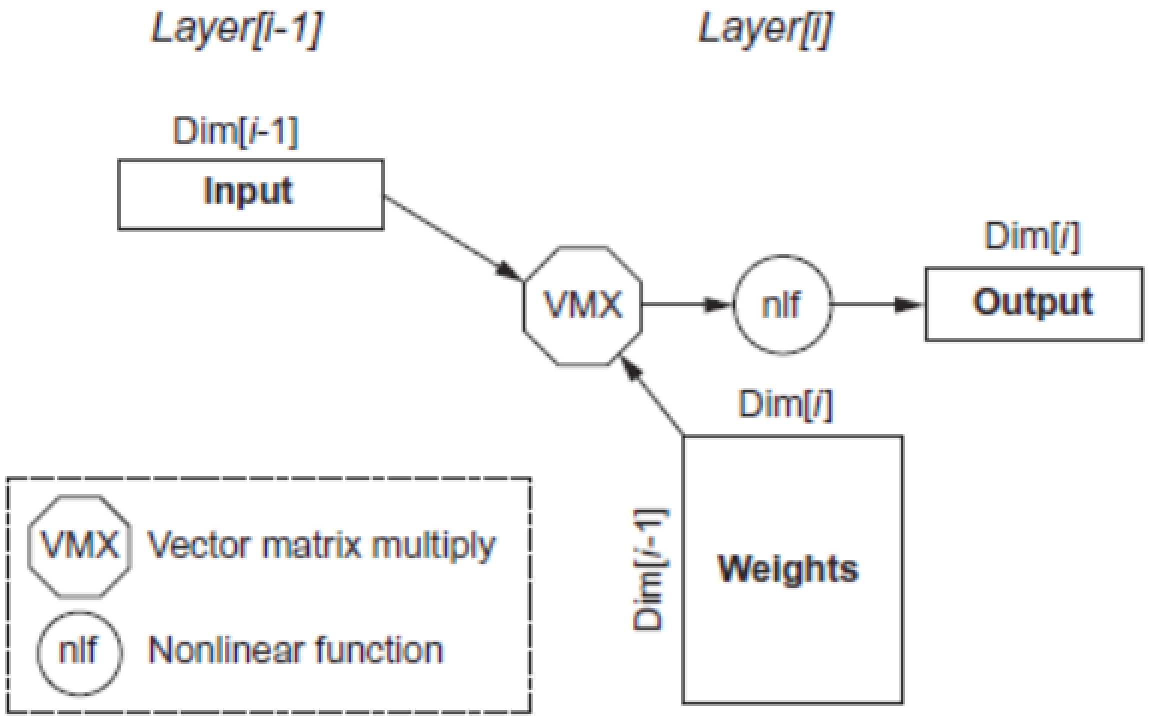
Type of data	Problem area	Size of benchmark's training set	DNN architecture	Hardware	Training time
text [1]	Word prediction (word2vec)	100 billion words (Wikipedia)	2-layer skip gram	1 NVIDIA Titan X GPU	6.2 hours
audio [2]	Speech recognition	2000 hours (Fisher Corpus)	11-layer RNN	1 NVIDIA K1200 GPU	3.5 days
images [3]	Image classification	1 million images (ImageNet)	22-layer CNN	1 NVIDIA K20 GPU	3 weeks
video [4]	activity recognition	1 million videos (Sports-1M)	8-layer CNN	10 NVIDIA GPUs	1 month

Most popular DNNs

- Multilayer perceptron (MLP)
- Recurrent neural network (RNN)
  - Speech recognition
  - Computer translation
- Convolutional neural network (CNN)
  - Computer vision

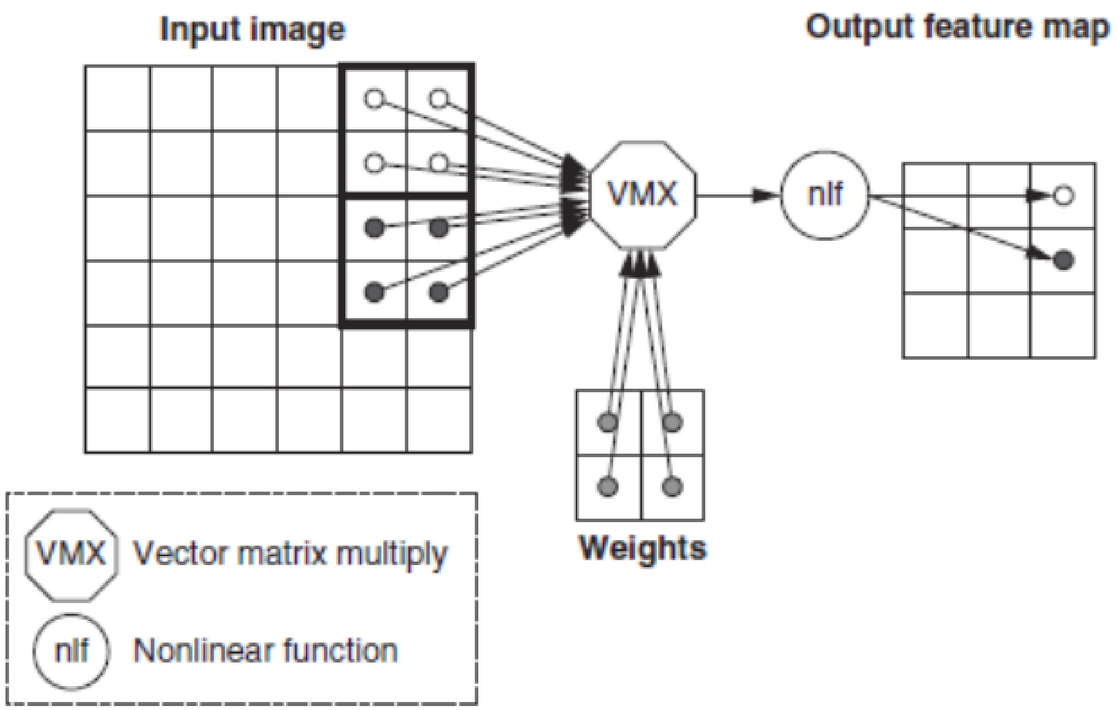
Multi-Layer Perceptron

- 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

- Computer vision
- Each layer raises the level of abstraction
  - First layer recognizes horizontal and vertical lines
  - Second layer recognizes corners
  - Third layer recognizes shapes
  - Fourth layer recognizes features, such as ears of a dog
  - Higher layers recognizes different breeds of dogs



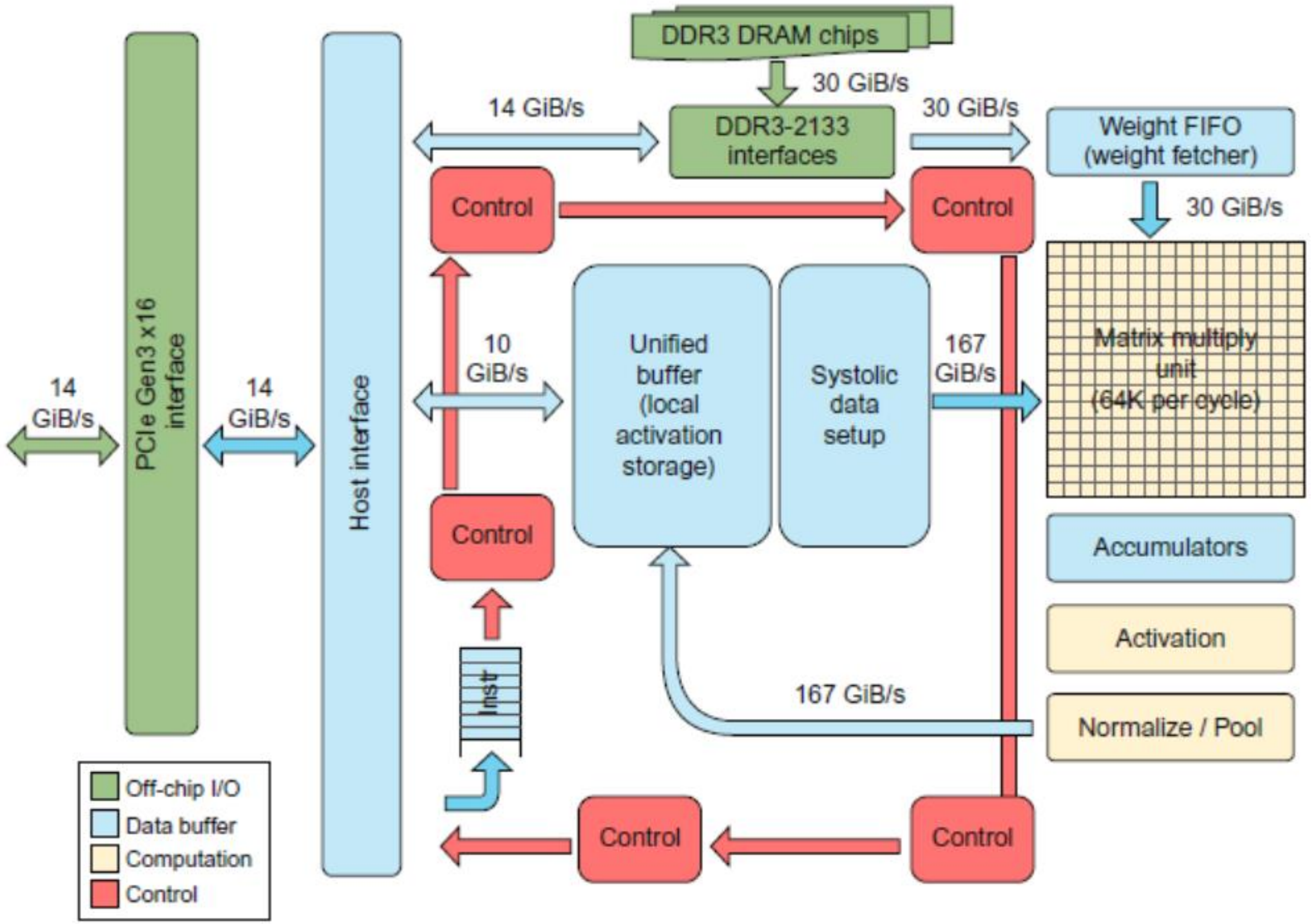
DNN Summary

- Batches
  - Reuse weights once fetched from memory across multiple inputs
  - Increases operational intensity
- Quantization
  - Use 8- or 16-bit fixed point
- Operations
  - Matrix-vector multiply
  - Matrix-matrix multiply
  - Stencil
  - ReLU
  - Sigmoid
  - Hyperbolic tangeant

Tensor Processing Unit

- Google's DNN ASIC
- 256 x 256 8-bit matrix multiply unit
- Large software-managed scratchpad
- Coprocessor on the PCIe bus

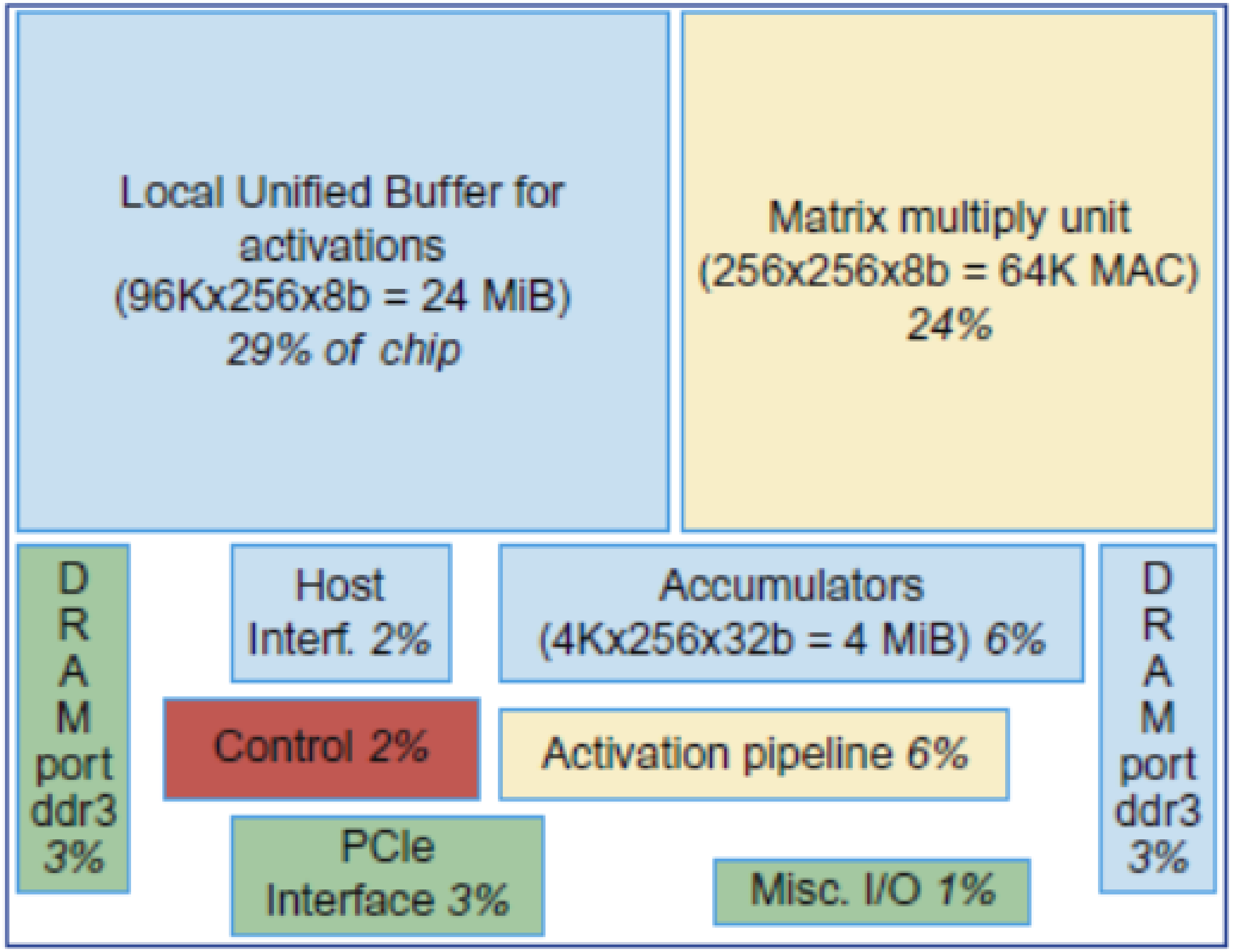
Tensor Processing Unit

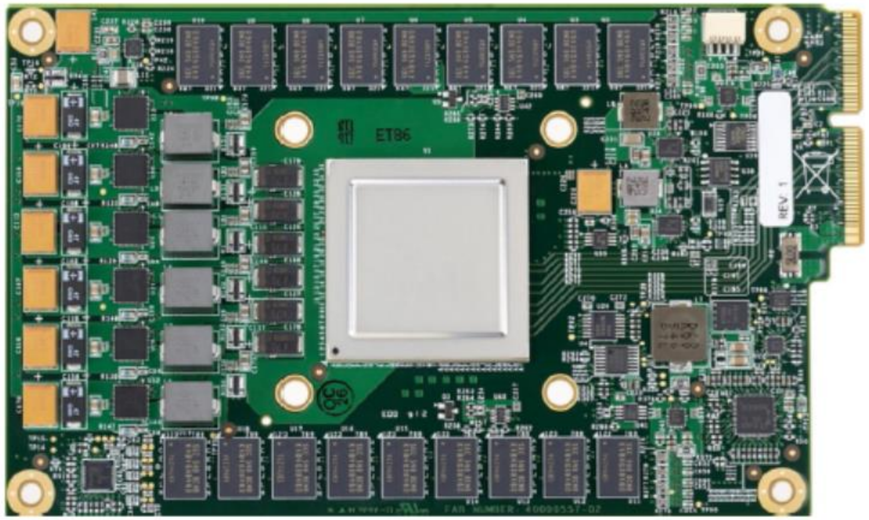


TPU ISA

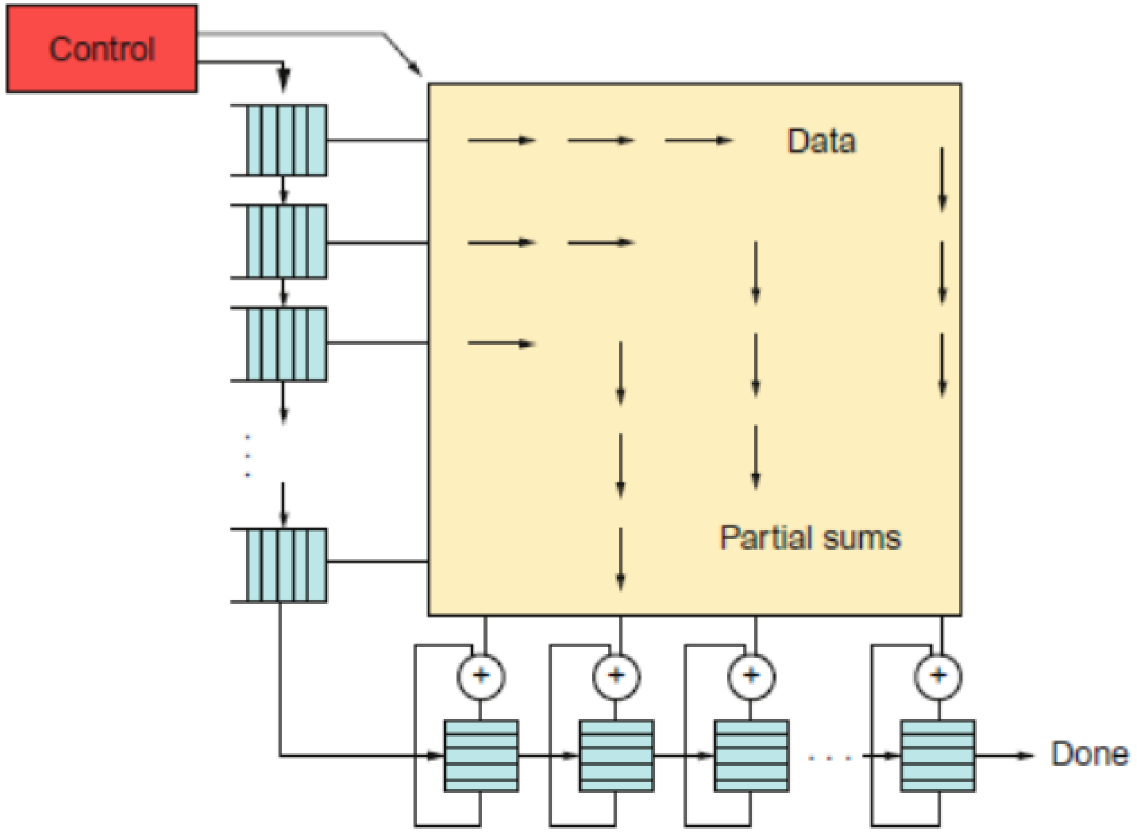
- **Read_Host_Memory**
  - Reads memory from the CPU memory into the unified buffer
- **Read_Weights**
  - Reads weights from the Weight Memory into the Weight FIFO as input to the Matrix Unit
- **MatrixMatrixMultiply/Convolve**
  - Perform a matrix-matrix multiply, a vector-matrix multiply, an element-wise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

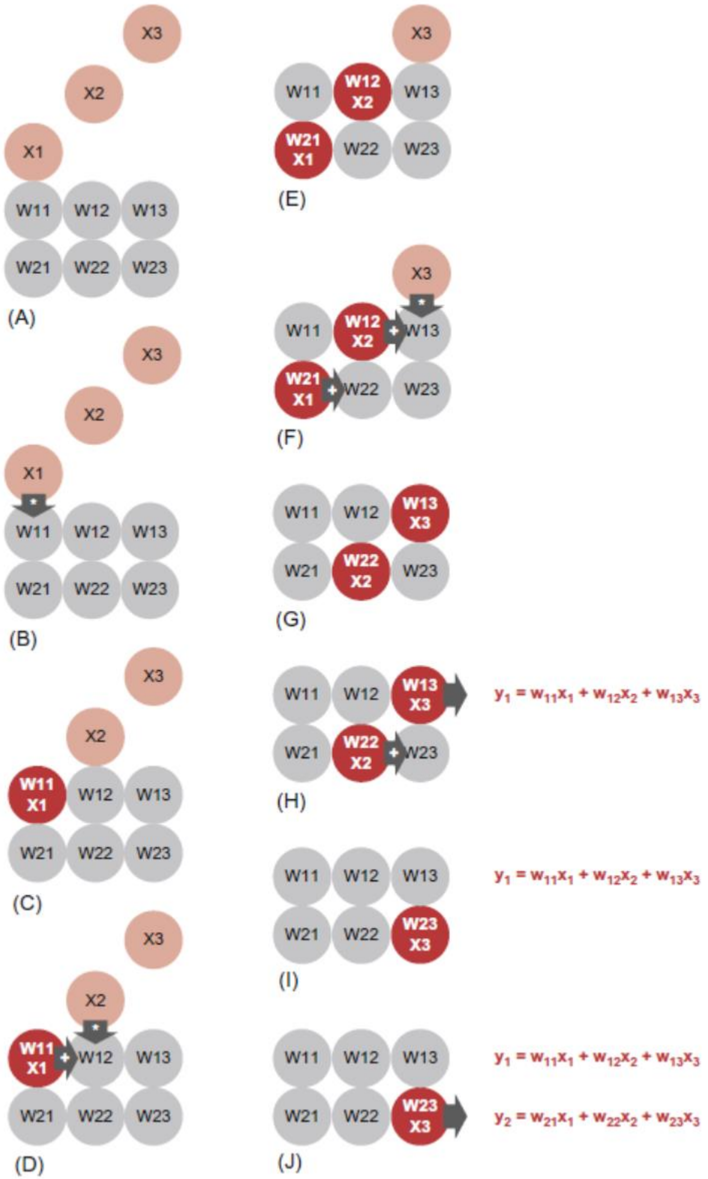
Tensor Processing Unit





TPU ISA


Control
Data
Partial sums
Done


(A)
(B)
(C)
(D)
(E)
(F)
(G)
(H)
(I)
(J)
$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$
$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$

The TPU and the Guidelines

- Use dedicated memories
  - 24 MiB dedicated buffer, 4 MiB 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

Any Questions?

```
            .text
__start:    addi t1, zero, 0x18
            addi t2, zero, 0x21
cycle:      beq t1, t2, done
            slt t0, t1, t2
            bne t0, zero, if_less
            nop
            sub t1, t1, t2
            j cycle
            nop
if_less:    sub t2, t2, t1
            j cycle
done:       add t3, t1, zero
```