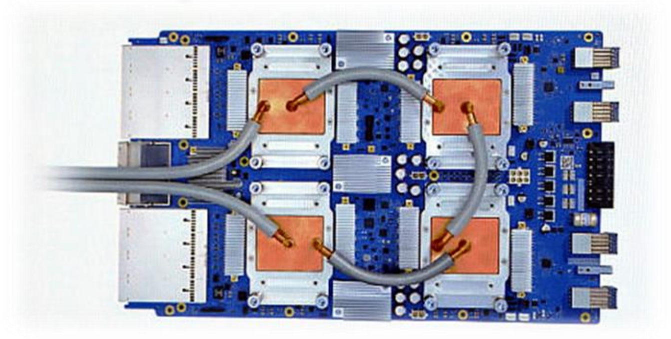
EITP25 2020

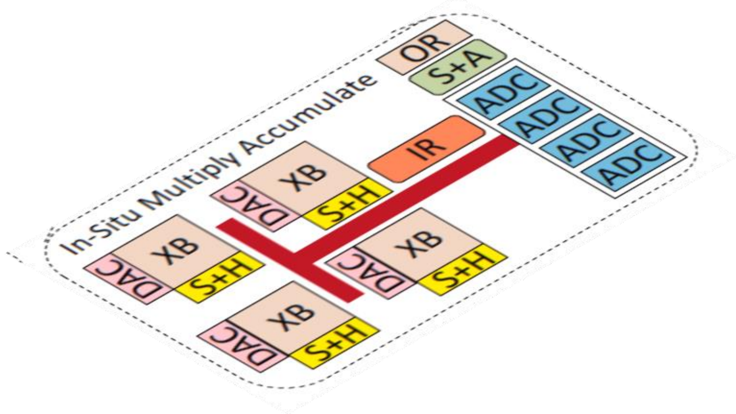
Lecture 5 –
Hardware for big data

Big data

Outline – Lecture 5

- The math of machine learning
- Graphics Processing Unit
- Tensor Processing Unit
- In-memory computation



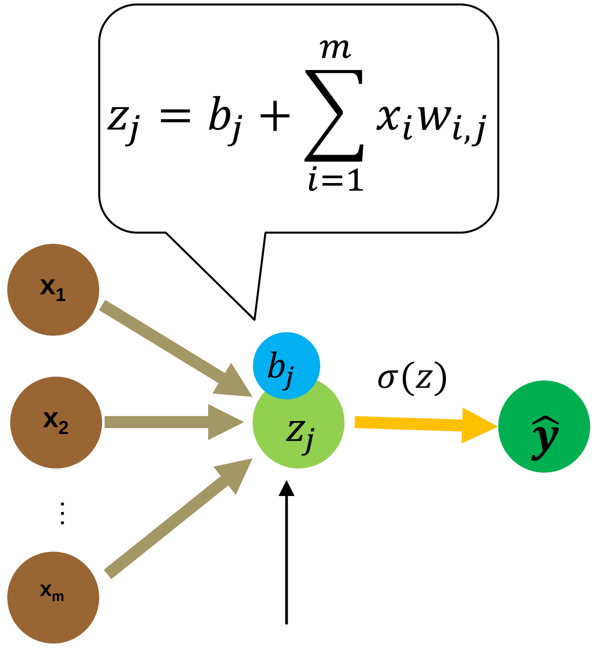


2 min exercise – Matrix multiplication

$$\begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 30 & 10 \\ 5 & 15 \end{bmatrix} = \boxed{?}$$

$$\underset{(m \times k)}{\begin{bmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mk} \end{bmatrix}} \underset{(k \times n)}{\begin{bmatrix} B_{11} & \cdots & B_{1n} \\ \vdots & \ddots & \vdots \\ B_{k1} & \cdots & B_{kn} \end{bmatrix}} = \underset{(m \times n)}{\begin{bmatrix} C_{11} & \cdots & C_{1n} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mn} \end{bmatrix}} \Rightarrow C_{ij} = \sum_{h=1}^{k} A_{ih} B_{hj}$$

Matrix multiply in deep learning



n nodes

In matrix form

$$[\boldsymbol{X}] = \begin{bmatrix} \boldsymbol{X_1} \\ \vdots \\ \boldsymbol{X^k} \\ \vdots \\ \boldsymbol{X^N} \end{bmatrix}$$ i.e. all input data sets 1..N in batch

$$\boldsymbol{Z} = \boldsymbol{B} + \boldsymbol{X^k W} = \left[b_1 + \sum_{i=1}^{m} x_{i,k} w_{i,1} \quad \dots \quad b_n + \sum_{i=1}^{m} x_{i,k} w_{i,n} \right]$$

The Graphical Processing Unit

- Originally developed for 3D graphics
- All shapes described by triangular polygons
  - 3 points (vertexes) in space
  - Describe a plane (face) that can be shaded

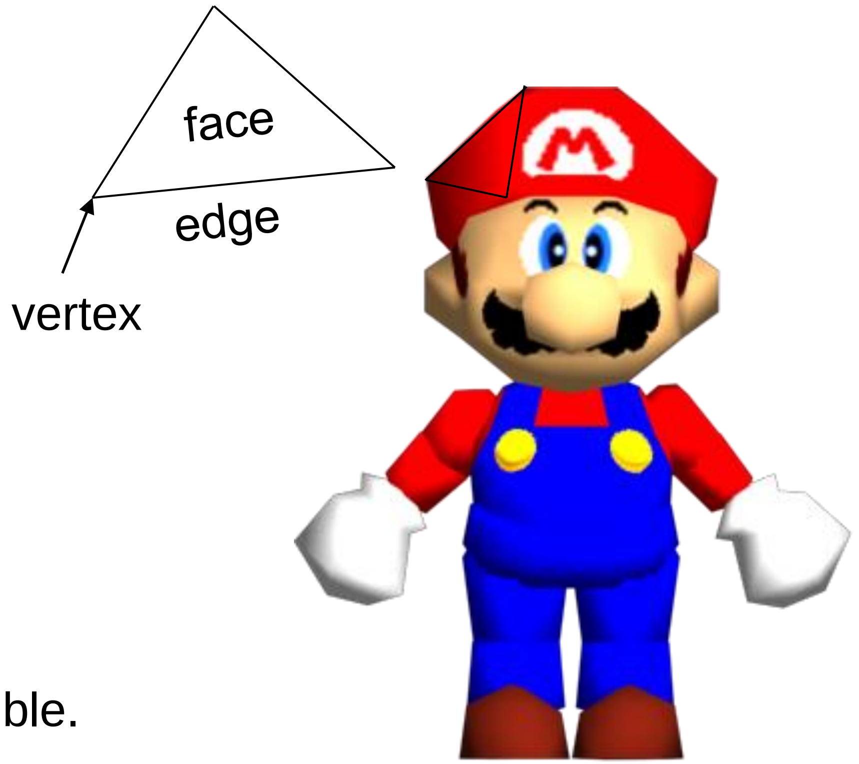
- Image generated by projection of polygons onto view plane
  - Requires A LOT of linear algebra

  i.e. matrix-vector multiplications…

  - The GPU is designed to perform these as fast as possible.

    → Useful for other things too!

2007: Nvidia released CUDA – Platform for general purpose computing on GPU

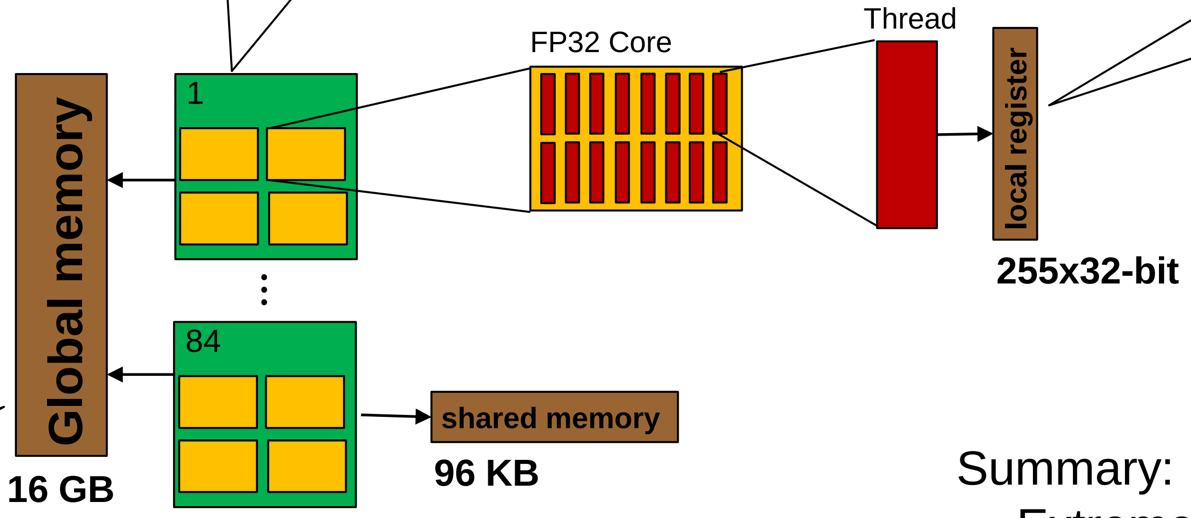


Architecture of the Nvidia Volta GPU

125 teraflops @ 300W
416 gigaflops/W

84 *multiprocessors* each with many processing *cores*

Max 32 threads/core
→ 175 056 threads in total

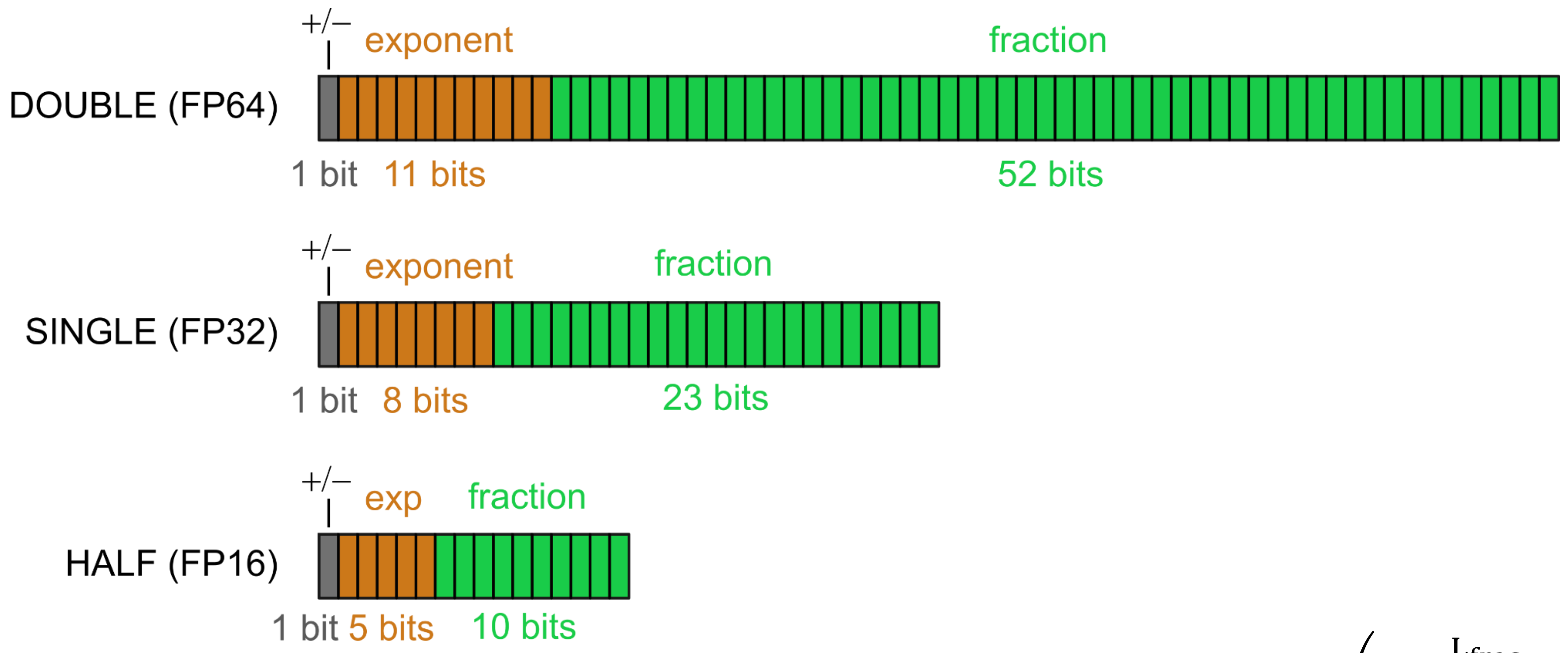


Globally shared high bandwidth memory (storage)

64 FP32
64 INT32
32 FP64
8 Tensor

Summary:
- Extremely parallelized hardware!
- Each core is very simple
- Focuses on single precision (FP32)

Data representation



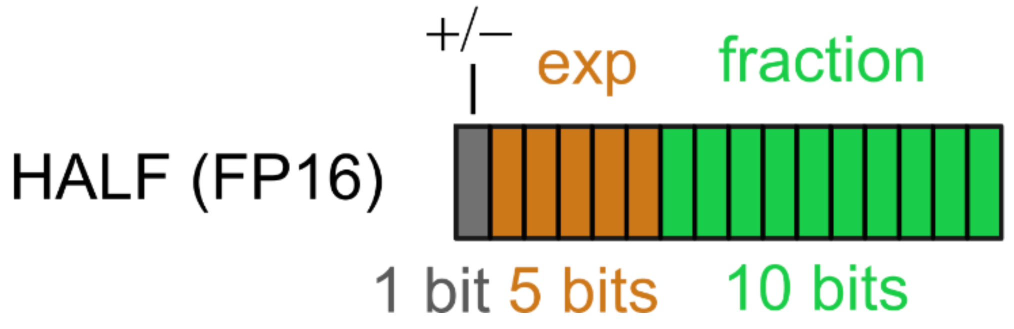
$$N = (-1)^{sign}\left(1 + \sum_{i=1}^{\mathrm{L_{frac}}} b_{\mathrm{L_{frac}}-i} 2^{-i}\right) * 2^{exponent}$$

So what precision do we actually need?

Exercise - What is actually 16 bit precision?

- What is the number of decimals it can represent?

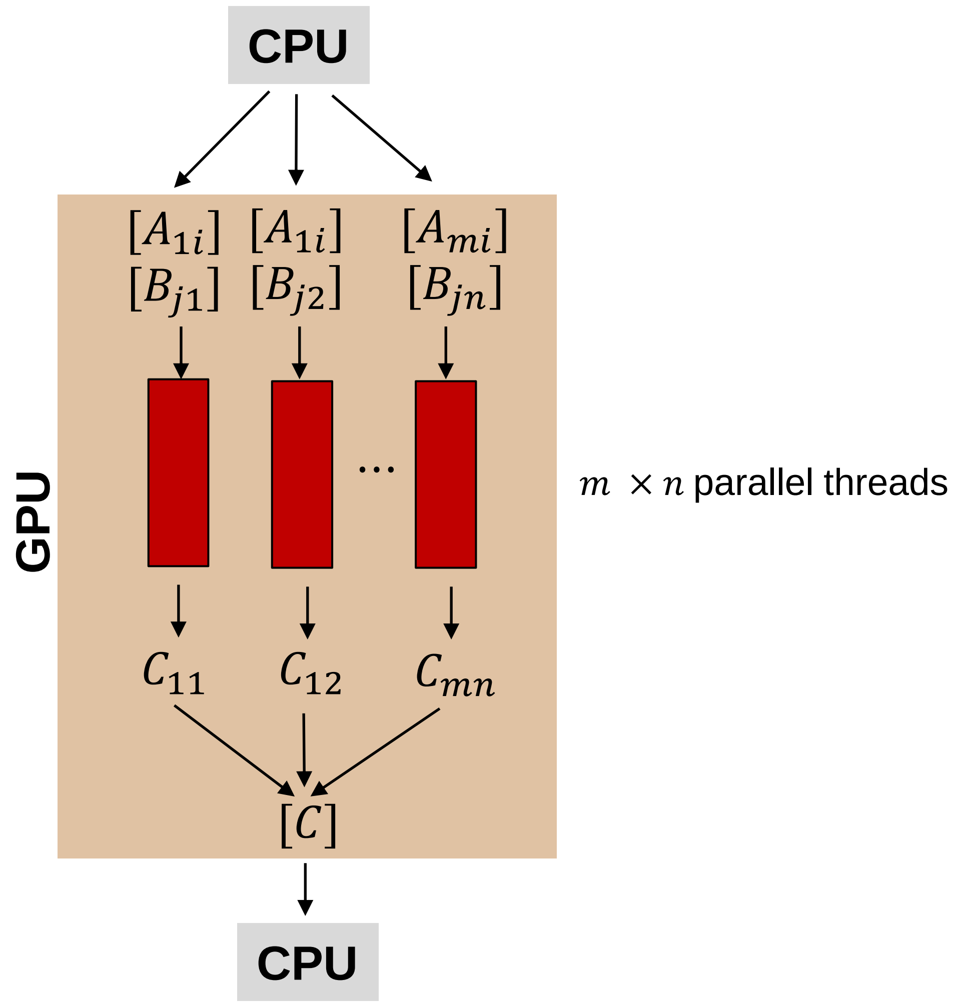
$$N = (-1)^{sign}\left(1 + \sum_{i=1}^{\mathrm{L_{frac}}} b_{\mathrm{L_{frac}}-i} 2^{-i}\right) * 2^{exponent}$$



Why GPU for matrix multiply?

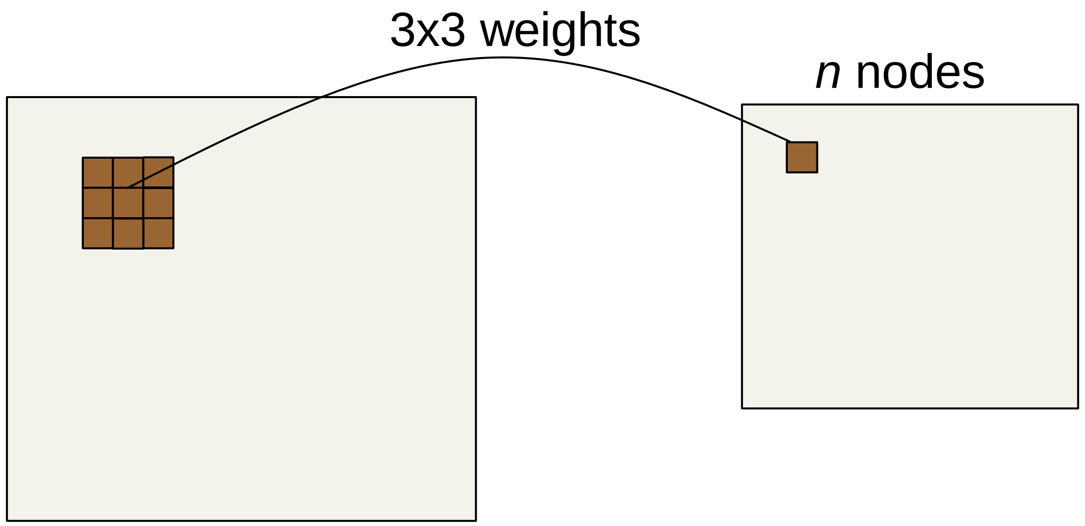
- $$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}\begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

- Every element in C needs only two vectors → can be parallelised.
- GPU offers extreme parallelized hardware
  - 10^6-10^9 threads
  - Local memory levels



Convolution on GPU

Convolution is an excellent use case for parallelization



$$\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_9 \end{bmatrix}, \boldsymbol{X_j} = \begin{bmatrix} x_1 \\ \vdots \\ x_9 \end{bmatrix}$$

$$z_j = b_j + \boldsymbol{X_j^T W}$$

x *n* such multiplications

x number of filters

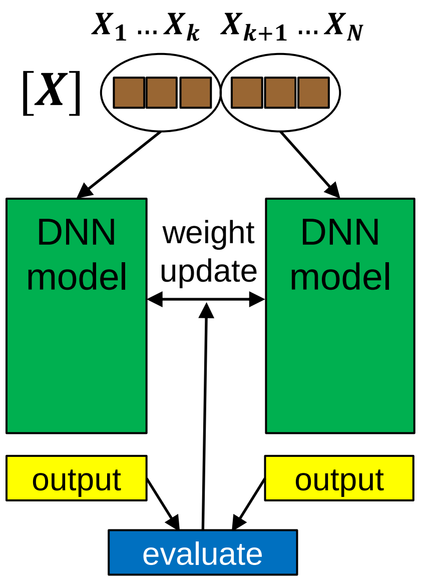
i.e. many smaller vector multiplications…

Deep learning on GPU

DATA PARALLELISM

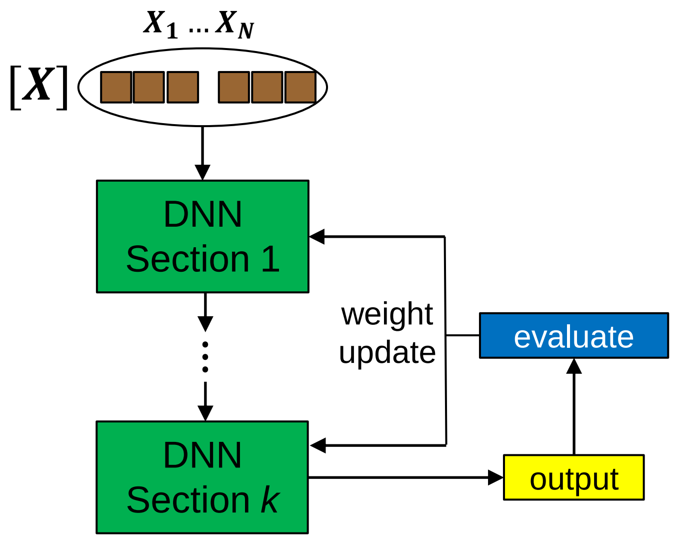
- Divide the data
- Run through model in parallel

→ Allows for bigger data sets



MODEL PARALLELISM

- Split the model
- Run model sections separately in parallel



The different parts can be computed on different parts of same GPU or in GPU network/cluster

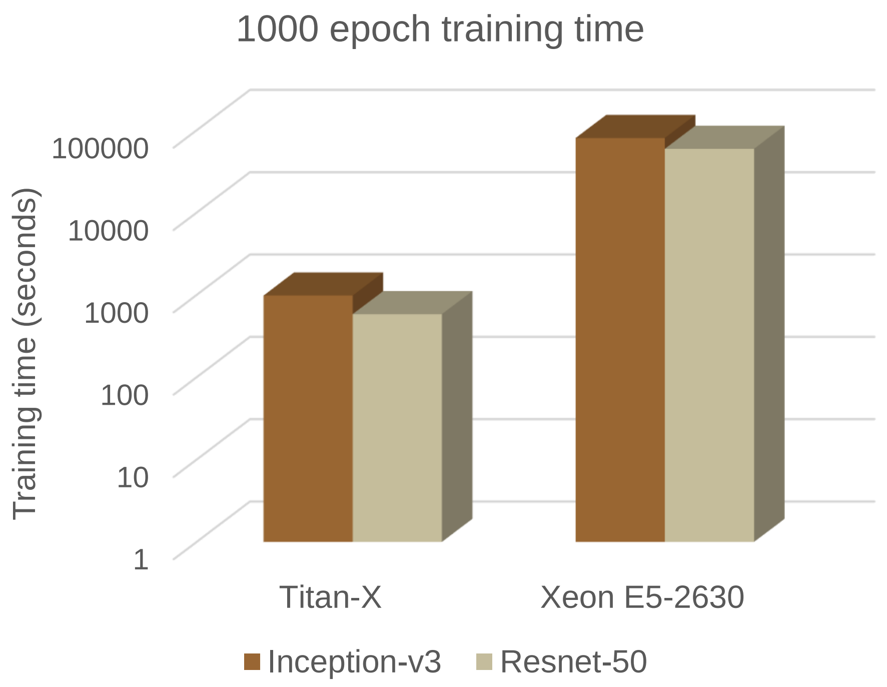
Cano et al. WIREs Data Mining Knowl Discov 2018, doi: 10.1002/widm.1232

GPU performance

- For training DNN's approximately 100x speed up compared to CPU

- Intel Xeon E5-2630:
  - 32 GB RAM with 6 cores and 12 threads @ 2.33 GHz.

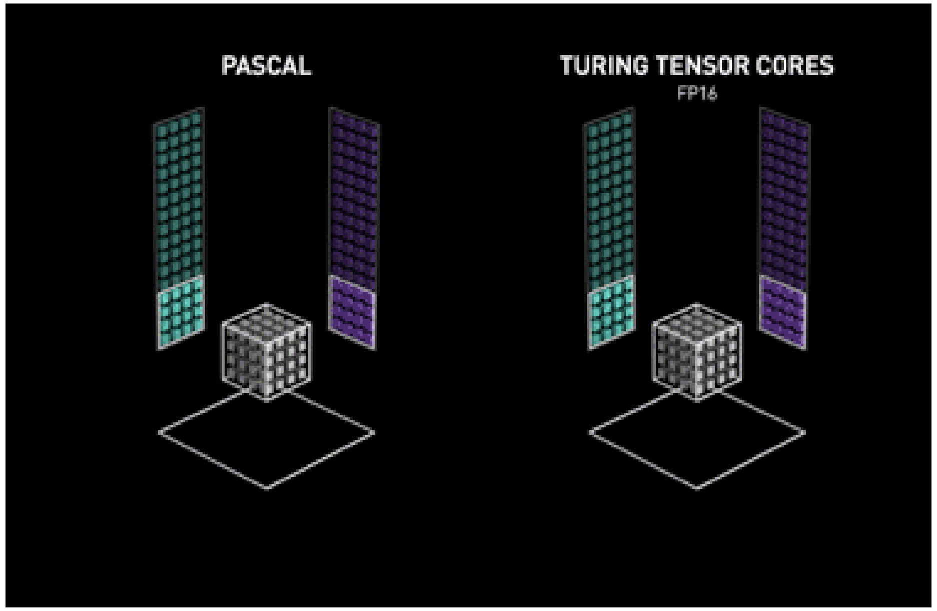
- NVIDIA TITAN X (No tensor cores):
  - 12 GB RAM and roughly 3072 cores



Sundar et al. ICECA 2018

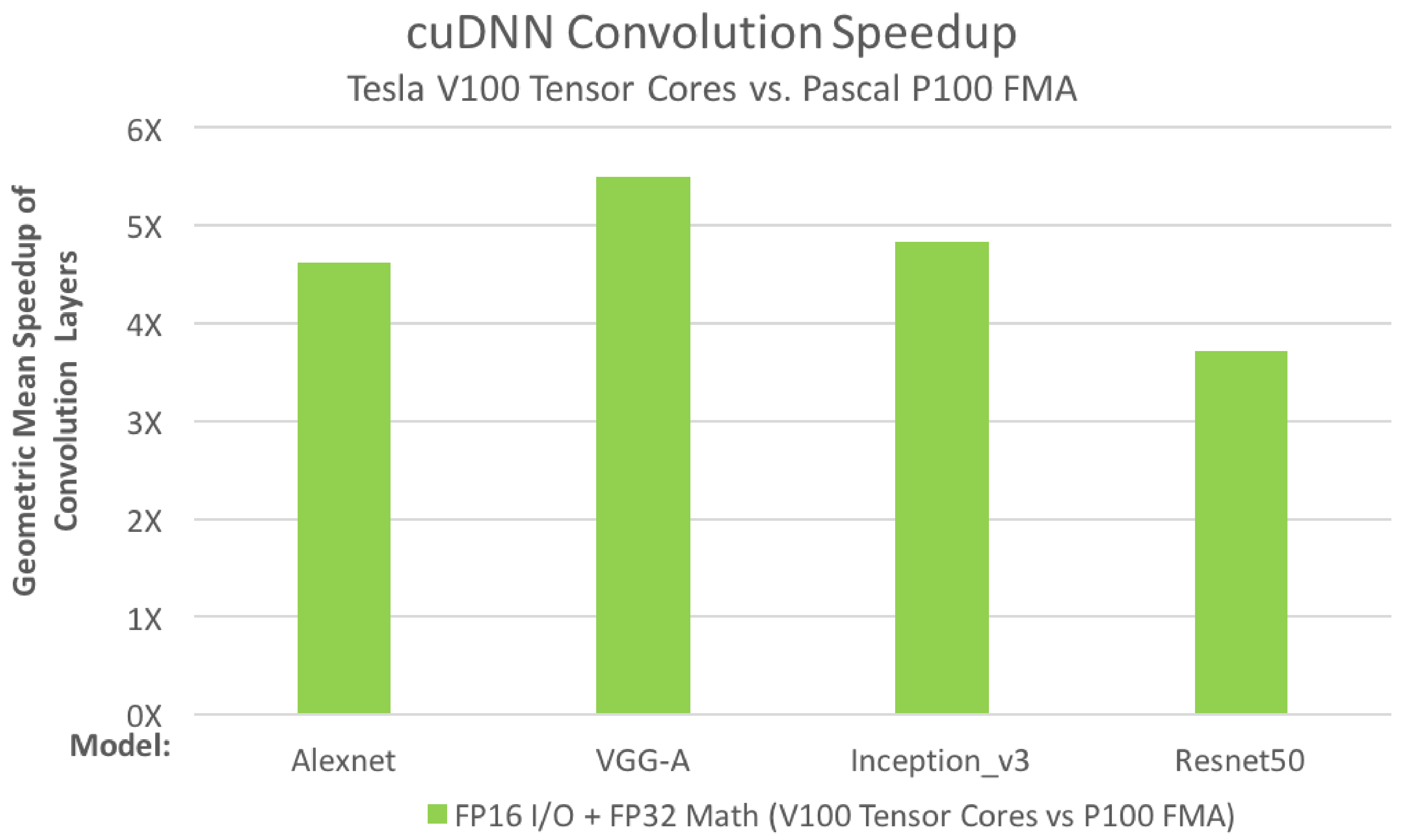
Tensor cores

- Nvidia Volta adds ”Tensor cores” to speed up matrix multiplication
- 4x4 simultaneous matrix mult. + accumulation
- Mixed precision, i.e. result and accumulation can be FP16 or FP32, multiplication is FP16.



$$\underbrace{D}_{\text{FP16 or FP32}} = \underbrace{\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}_{\text{FP16}} \underbrace{\begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}}_{\text{FP16}} + \underbrace{\begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}}_{\text{FP16 or FP32}}$$

GPU performance with tensor cores



Google's Tensor Processing Units

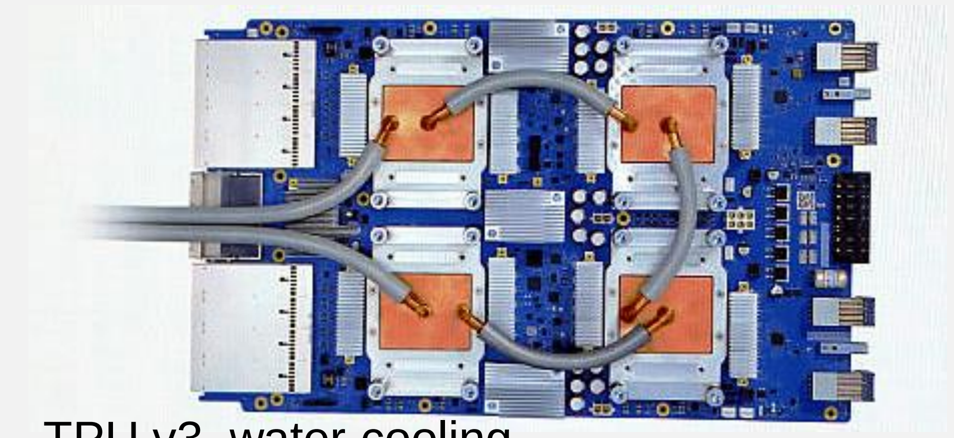
- Highly specialized hardware by Google → designed ONLY for deep learning in mind
- Optimized for large batches and CNNs.
- Version 3 available since 2018 in Google Cloud.

- Provided as package with 4 chips
- Each chip: 2 cores with matrix multiplier units (MXU)
  - V2: 1 MXU/core → 8 for package
  - V3: 2 MXU/core → 16 for package

- TPU v2: 180 teraflops
  @ > 4x200W → ~225 gigaflops/W
- TPU v3: 420 teraflops
  (power usage unknown, but needs water cooling!! >300W x 4)
- ~350 gigaflops/W
- Memory bandwidth: 2400 GB/s (x1.5 in v3)

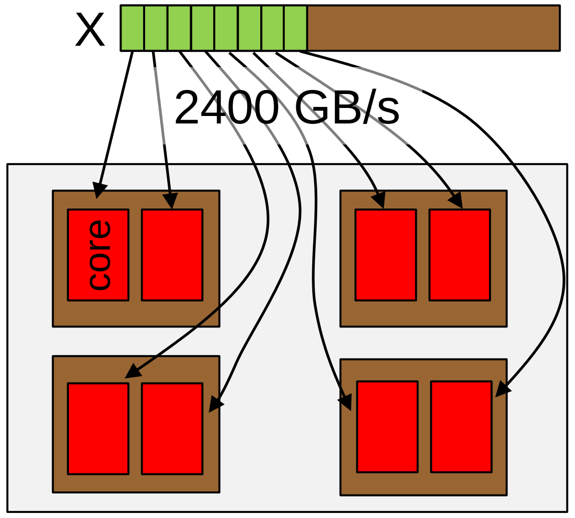
The size of these heat sinks screams "over 200W each."
https://www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/

TPU v2



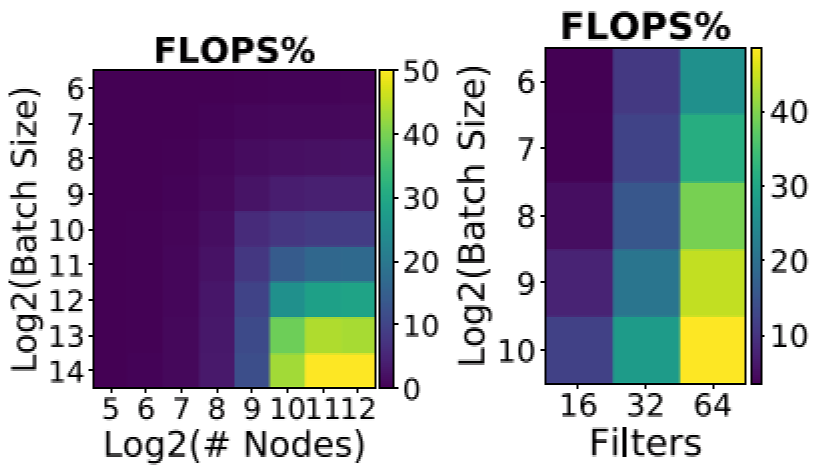
TPU v3. water cooling

TPU compute model



A full batch of training data is sent to each core
i.e. the core memory (8 GB v2, 16GB in v3) limits batch size!

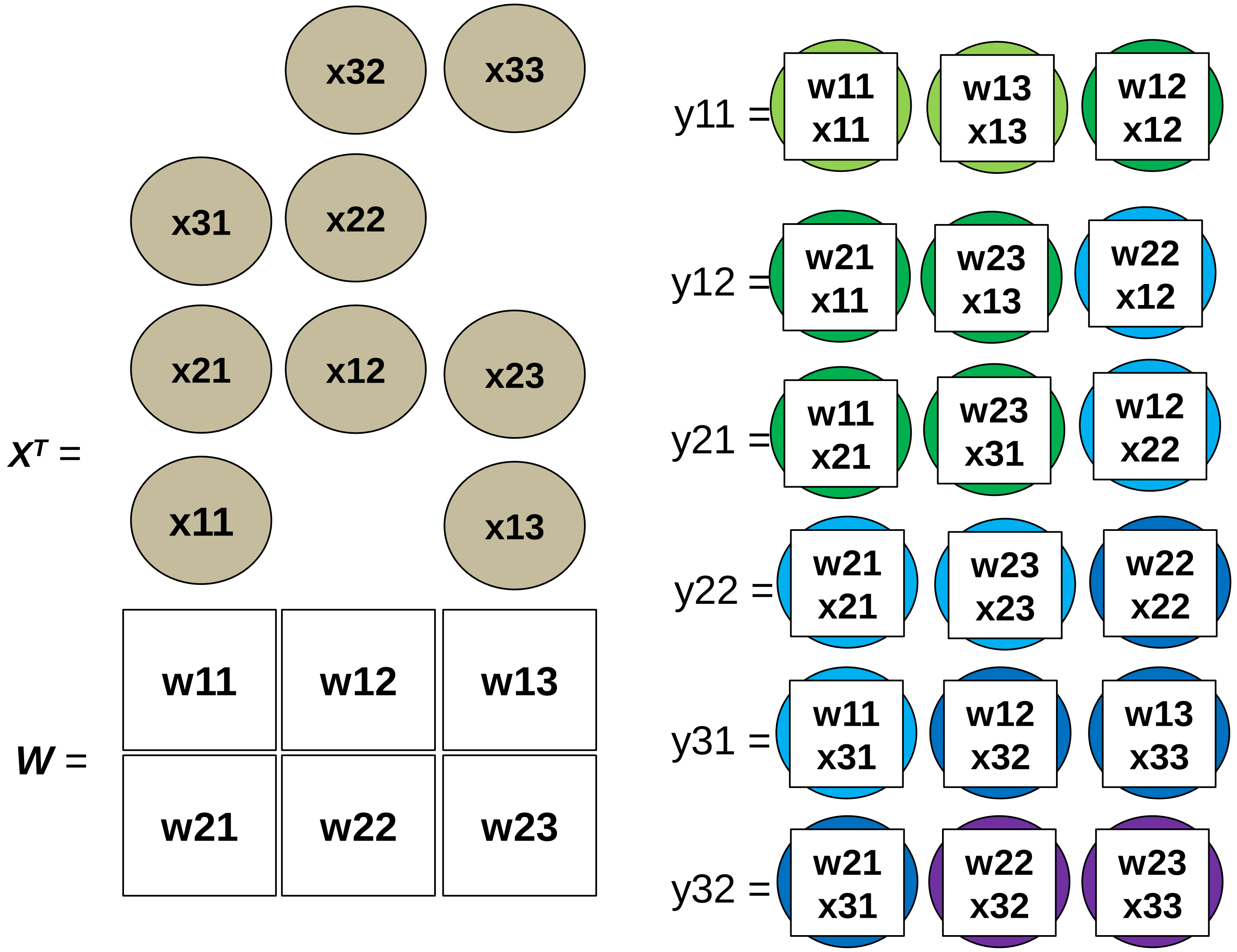
Memory bandwidth often limits performance. Best FLOP utilization with
- large batches
- many filters (CNN)



(a) FC (b) CNN

Matrix multiplier unit

- **Systolic array**
  - calculates matrix multiply ”on the fly”
  - Does not wait for all data to arrive

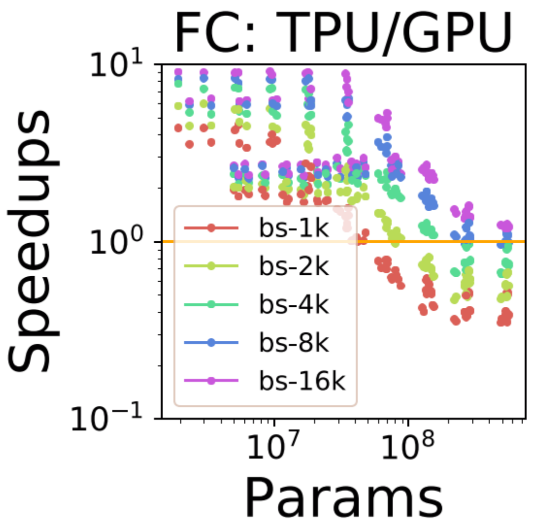


https://medium.com/sciforce/understanding-tensor-processing-units-10ff41f50e78

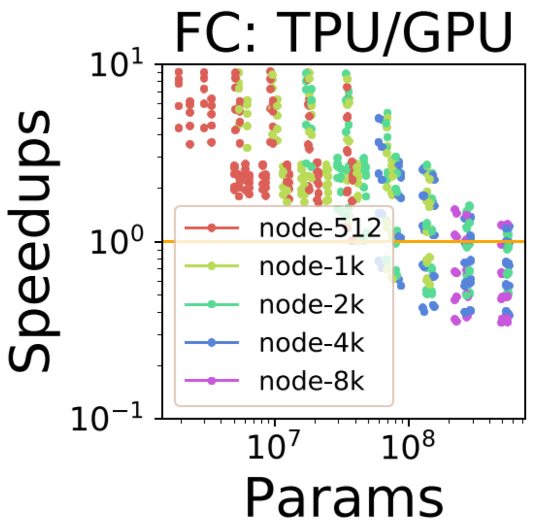
TPU vs GPU

- For **fully connected layers**:
  - Large batches → TPU wins
  - Large number of parameters (weights) → GPU wins
  - Size of each data (node) → GPU wins
- For **CNNs:**
  - Generally TPU wins!
  - Max benefit for large batch sizes (6x)
  - Max benefit for large filters

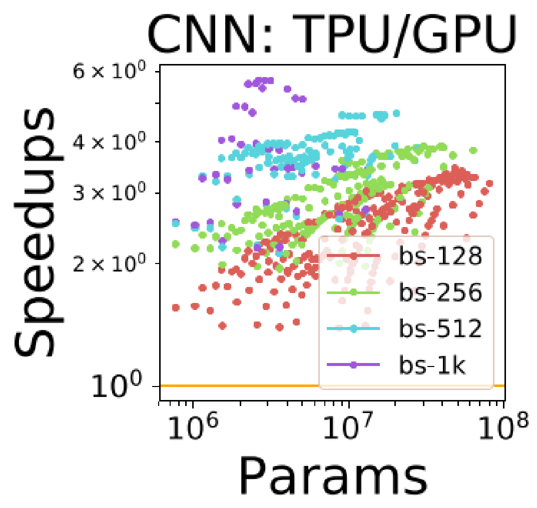
Q: Why different results for FCLs and CNNs?



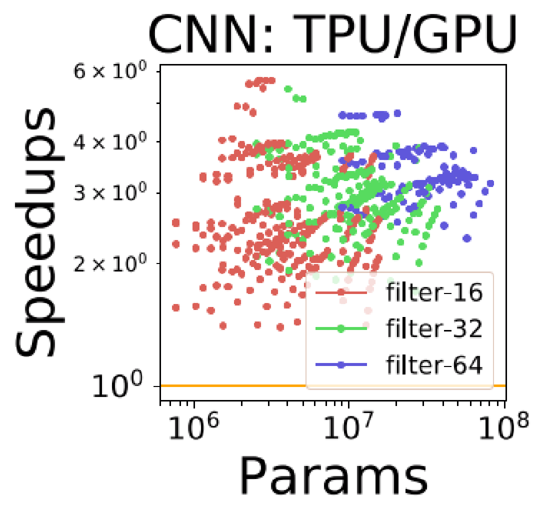
(b) Batch Size



(c) Node



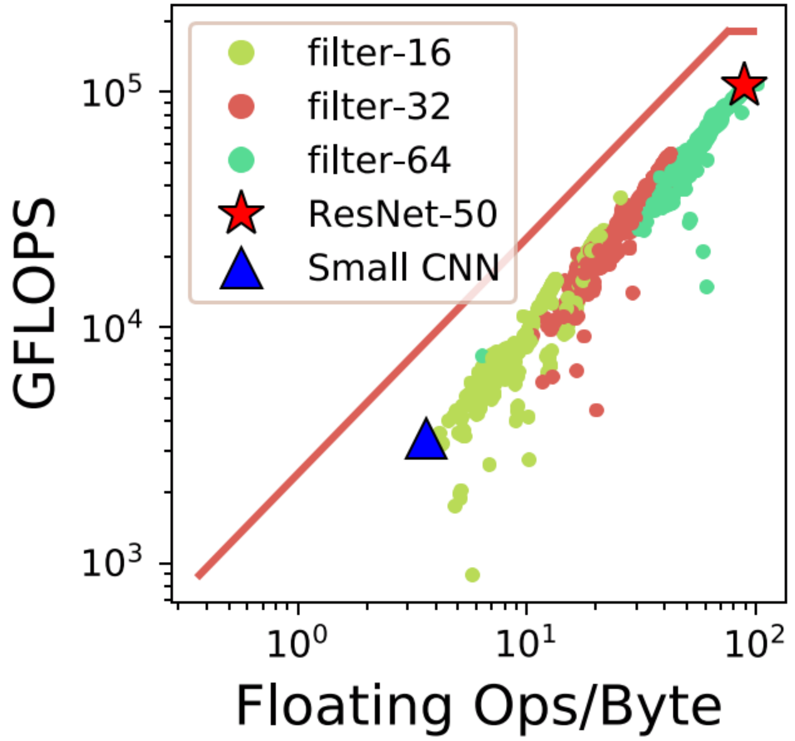
(b) Batch Size



(c) Filter Size

Memory limitation of TPU

- For CNNs the TPU is mostly memory transfer limited
- For compute heavy things (ResNet-50) → compute limited
- Does not often achieve FLOPs near theoretical limit!
  - Only for very large CNNs…



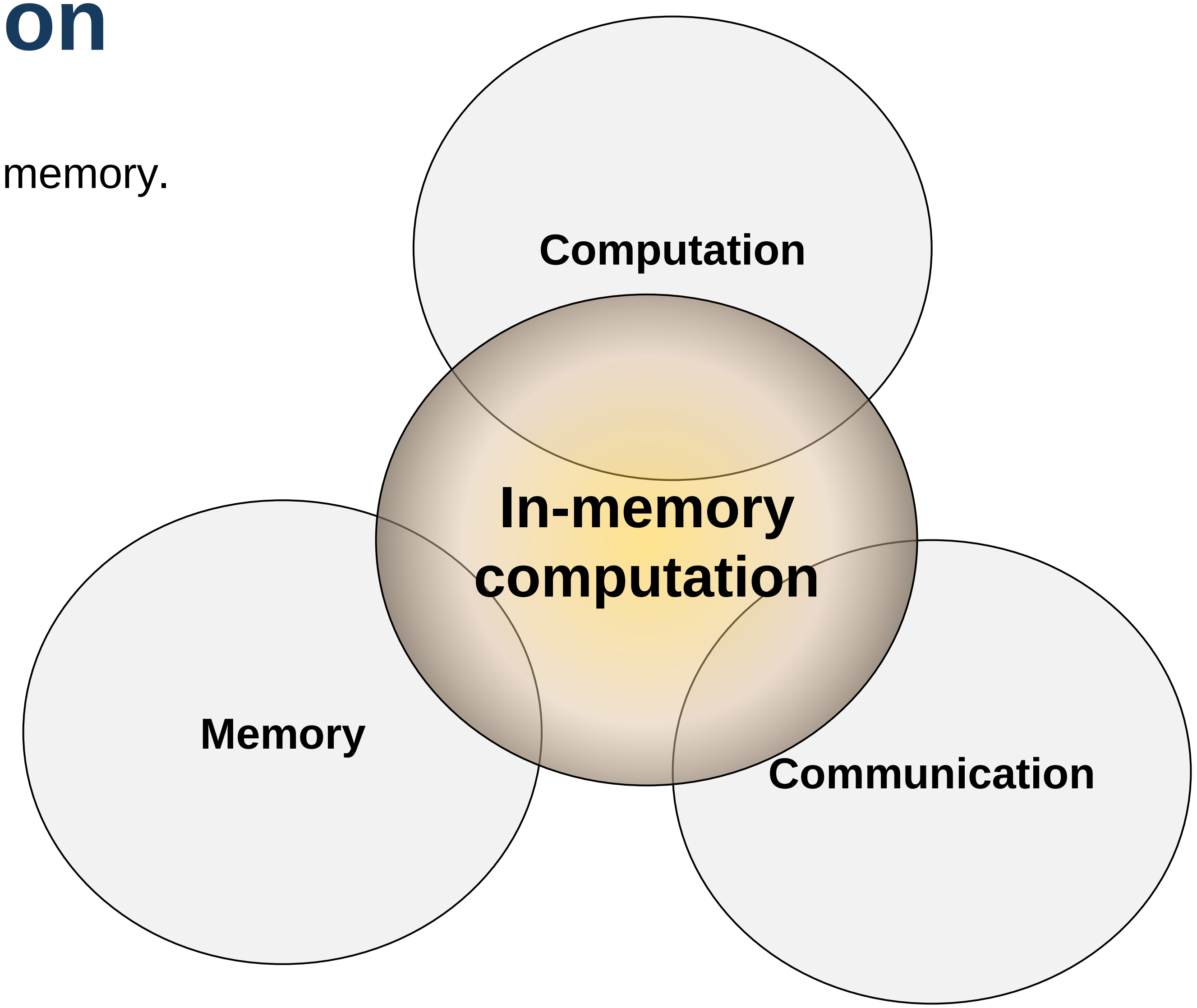
Can we avoid memory transfer limitation?

In-memory computation

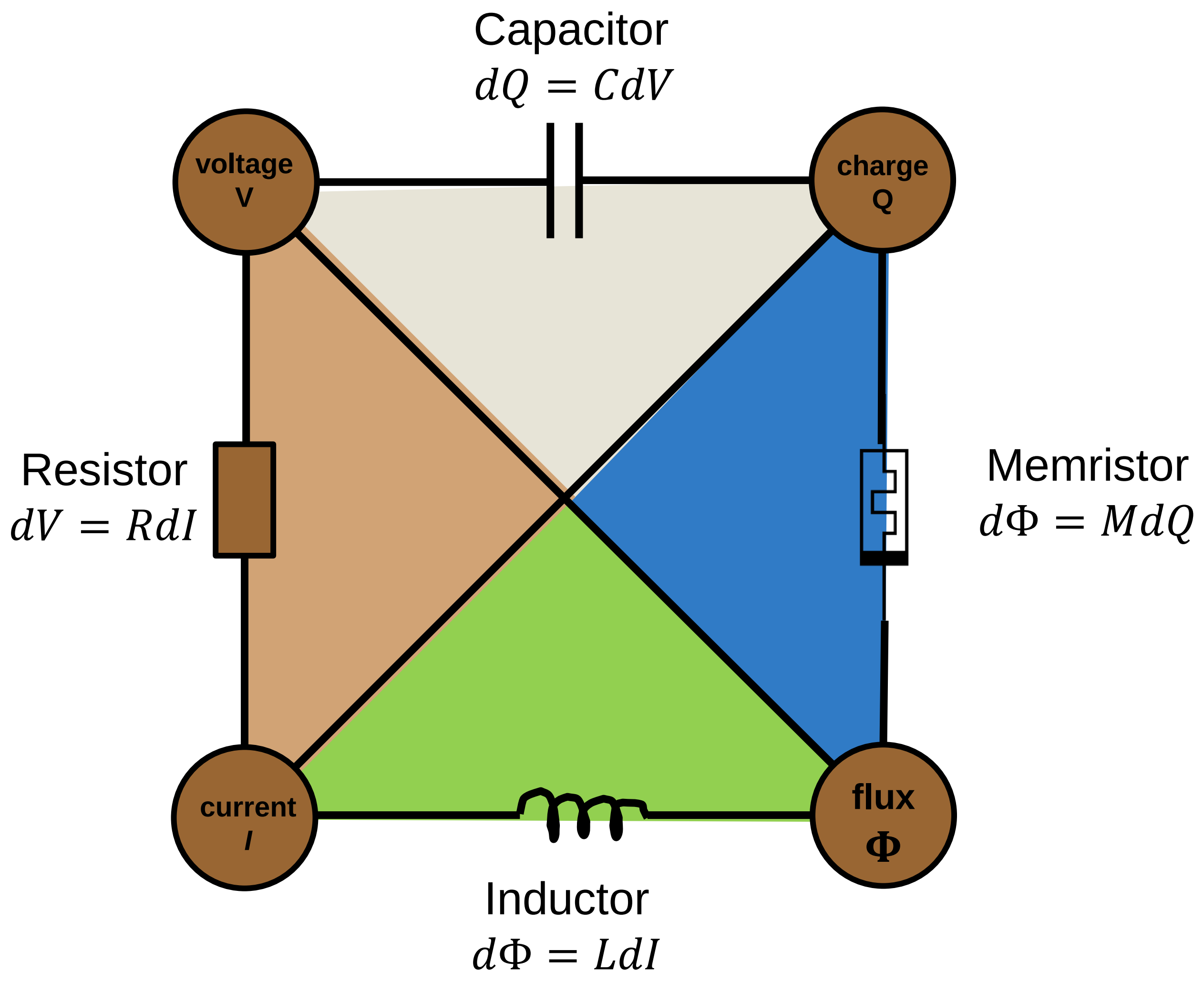
- **Concept:** Perform computation directly in memory.
- **Benefits:**
  1. Do not move memory around
     → reduced memory access latency
  2. Extreme parallelism!

- **Q: For what applications is this useful?**

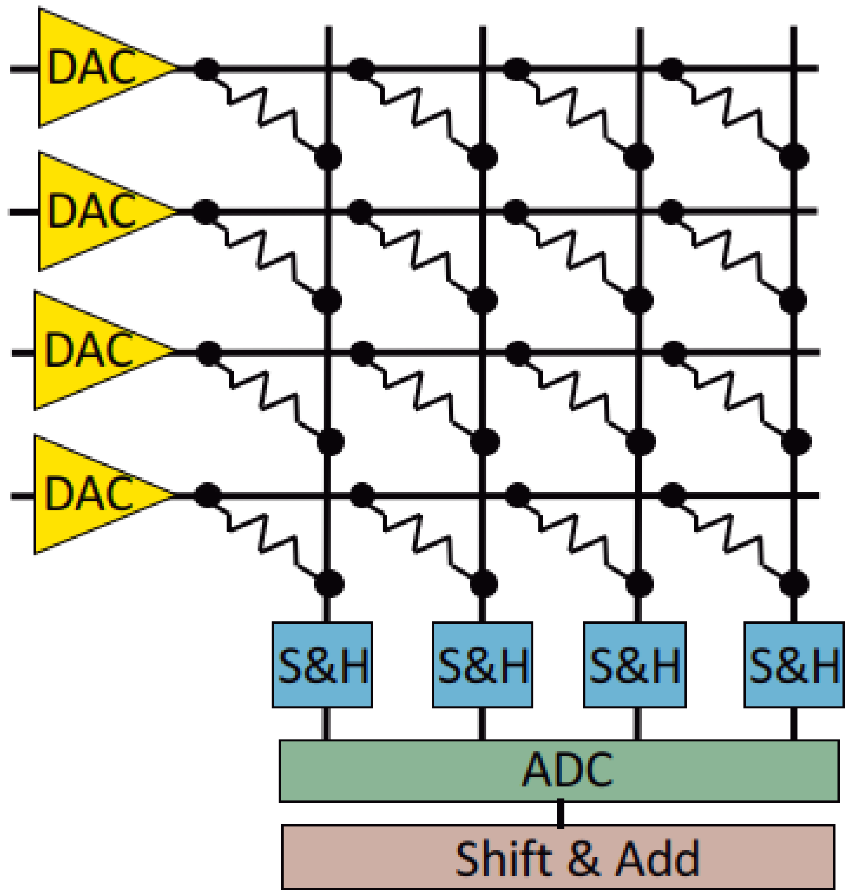
- How to achieve it?



The memristor



Memristor memory array



- Crossbar array: Memristor at every crossing
- Each row (word line) → input
  - Digital-analog converter (DAC) digital → analog signal
- Each column (bit line) → output
  - Sample and Hold circuit on each output (measures *I*)
- Common analog to digital converter (ADC) on output transforms to digital again
- (Shift & Add circuit collects bits together again)

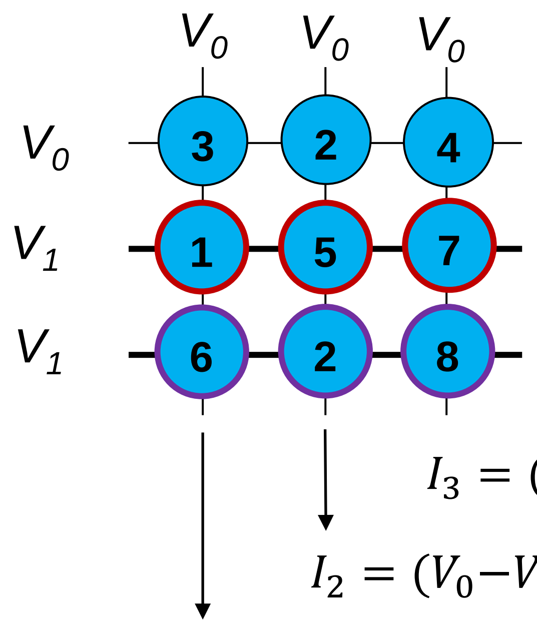
But how to use it?

Shafiee et al. ACM SIGARCH 2016

Vector addition

- **What is the sum of two (or more) vectors in the array?**

- Row bias high (V_1): select which vectors to add
- Row bias low (V_0): unselected vectors
- All column biases low (V_0)
- Current on each column gives the element sum



= [1 5 7] + [6 2 8]

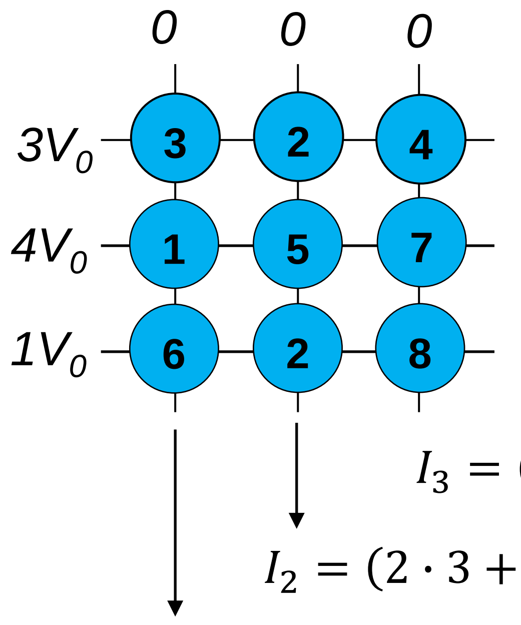
$$I_3 = (V_0 - V_0) \cdot 4 + (V_1 - V_0)(7 + 8) \rightarrow 15$$

$$I_2 = (V_0 - V_0) \cdot 2 + (V_1 - V_0)(5 + 2) \rightarrow 7$$

$$I_1 = (V_0 - V_0) \cdot 3 + (V_1 - V_0)(1 + 6) \rightarrow 7$$

Vector-Matrix multiplication

- $\boldsymbol{Ax = b?}$
- Represent the memory array as A^T
- x is applied as voltage to rows
- b is read as the currents on the output.



$$I_3 = (4 \cdot 3 + 7 \cdot 4 + 8 \cdot 1)V_0 \rightarrow 48V_0$$

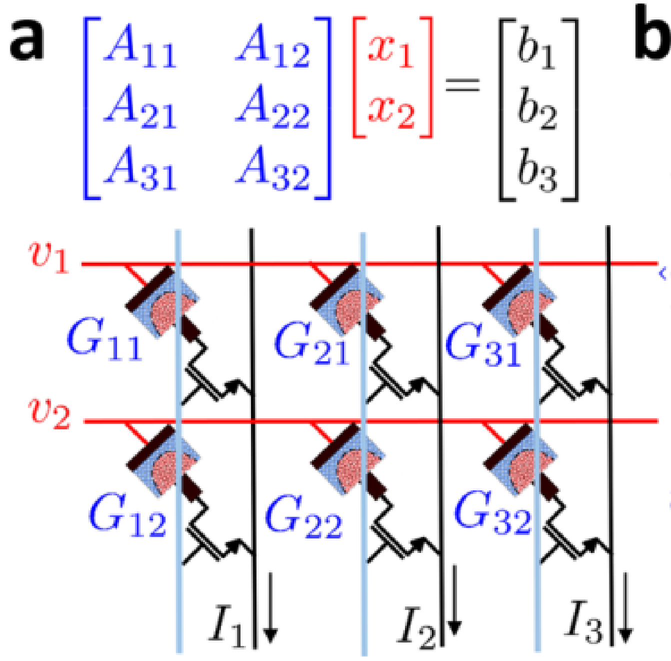
$$I_2 = (2 \cdot 3 + 5 \cdot 4 + 2 \cdot 1)V_0 \rightarrow 28V_0$$

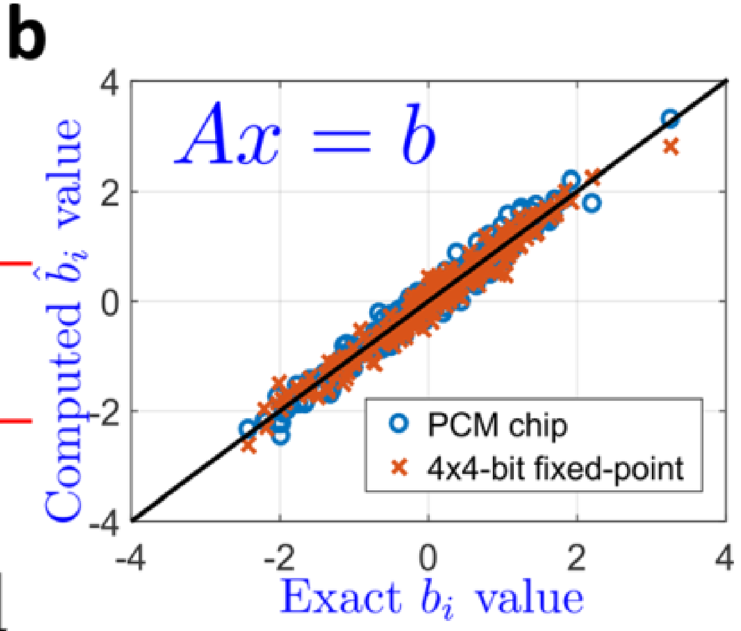
$$I_1 = (3 \cdot 3 + 1 \cdot 4 + 6 \cdot 1)V_0 \rightarrow 19V_0$$

Example: $\begin{bmatrix} 3 & 1 & 6 \\ 2 & 5 & 2 \\ 4 & 7 & 8 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix} = ?$

$\rightarrow [19 \quad 28 \quad 48]$

Matrix-Vector multiplication in practice


a
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
v_1
v_2
G_{11}
G_{21}
G_{31}
G_{12}
G_{22}
G_{32}
I_1
I_2
I_3

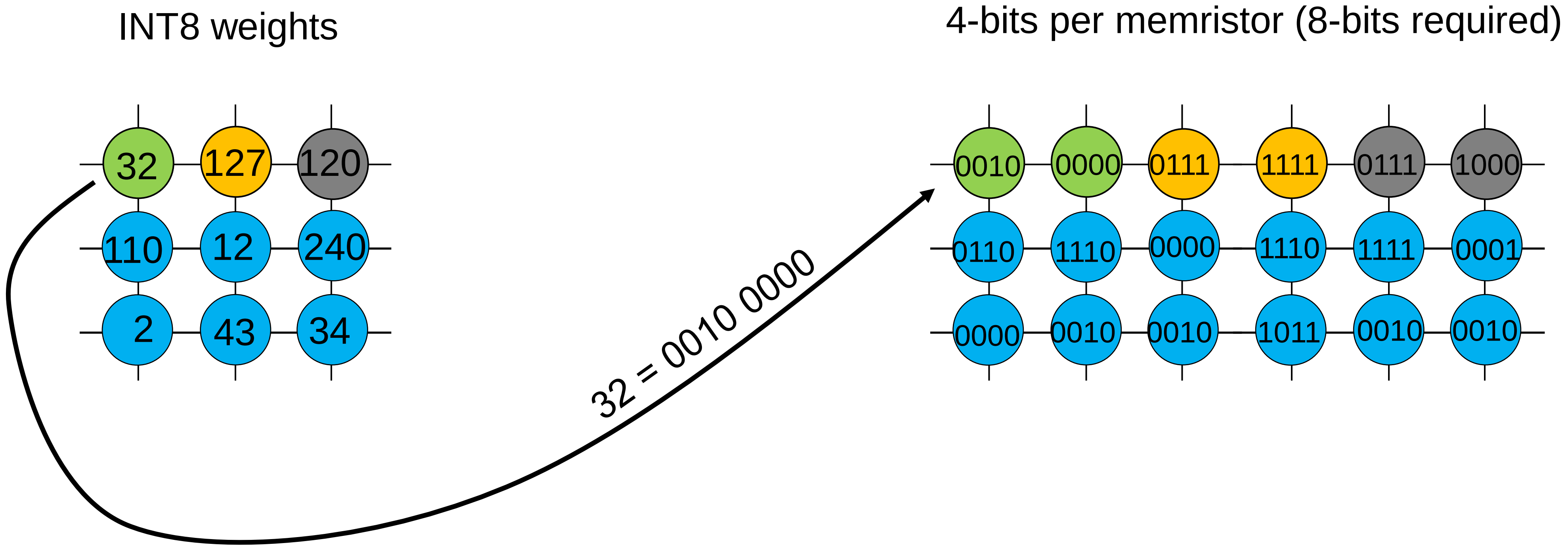


- 256x256 memory array (PCM)
  - memory cell conductance → memory state (A_{ij})
  - x_i given by applied voltages v_i to devices.
  - I_i → b_i
- Matches precision of 4-bit fixed point arithmetic

Sebastian et al. JAP 2018

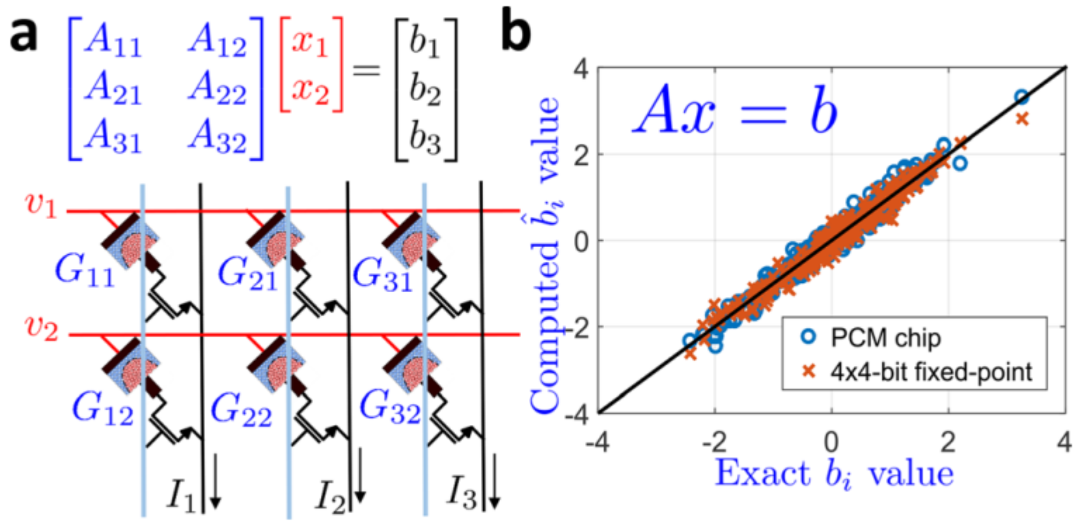
Few-bit representations

- If memristors cannot represent full precision → separate out bits
- Requires shift-and-adder circuit on output to collect numbers again



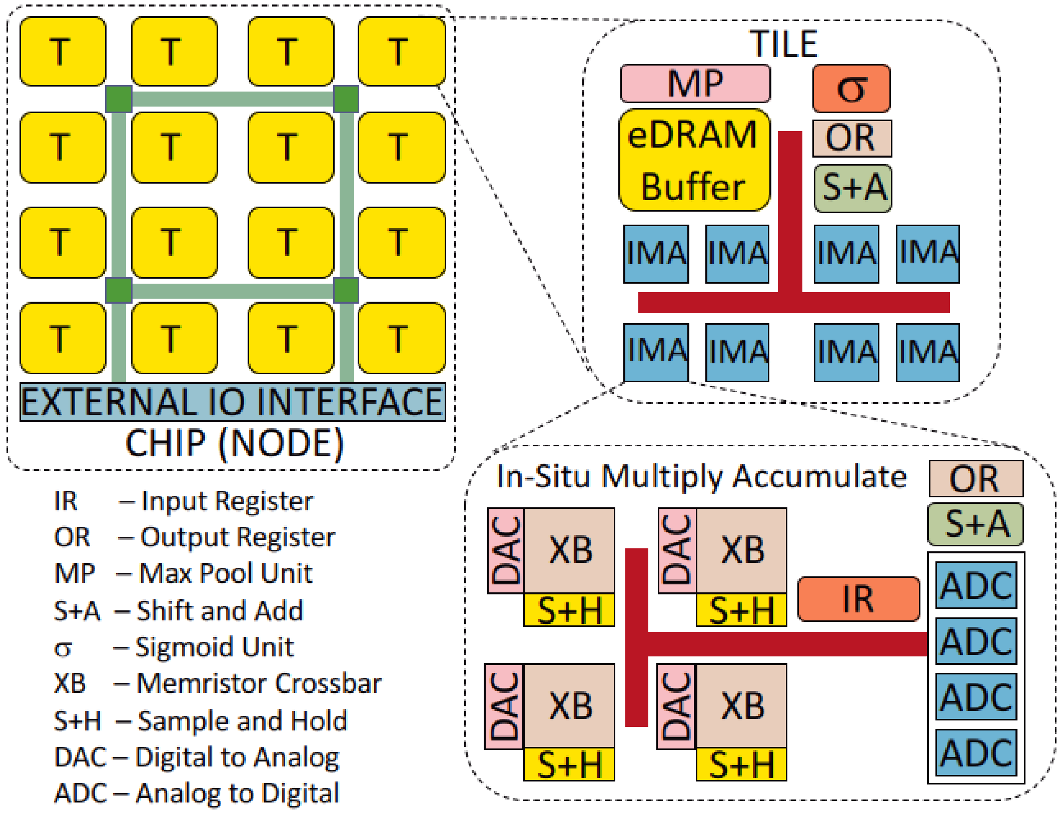
2 min exercise – bit representation in array

- Real-life memristor devices match 4 bit precision.
- If one needs 16 bit precision, how large array is needed to calculate *b* vector if size of *A* = [32, 32]?


a
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
v_1
v_2
G_{11}
G_{21}
G_{31}
G_{12}
G_{22}
G_{32}
I_1
I_2
I_3
b
$Ax = b$
Computed $\hat{b}_i$ value
Exact b_i value
PCM chip
4x4-bit fixed-point

Example of real system - ISAAC

- Developed in 2016
- Organized in *Tiles*, each with:
  - 8 In-situ Multiply-Accumulate (IMA) units
  - Activation (σ), maxpool (MP) units
  - eDRAM buffer for input
- Each *IMA* unit:
  - 4 memristor arrays (128 x128)
  - 2 bits/memristor
- Applied to CNN inferencing
  - Different Tiles mapped to different CNN layers

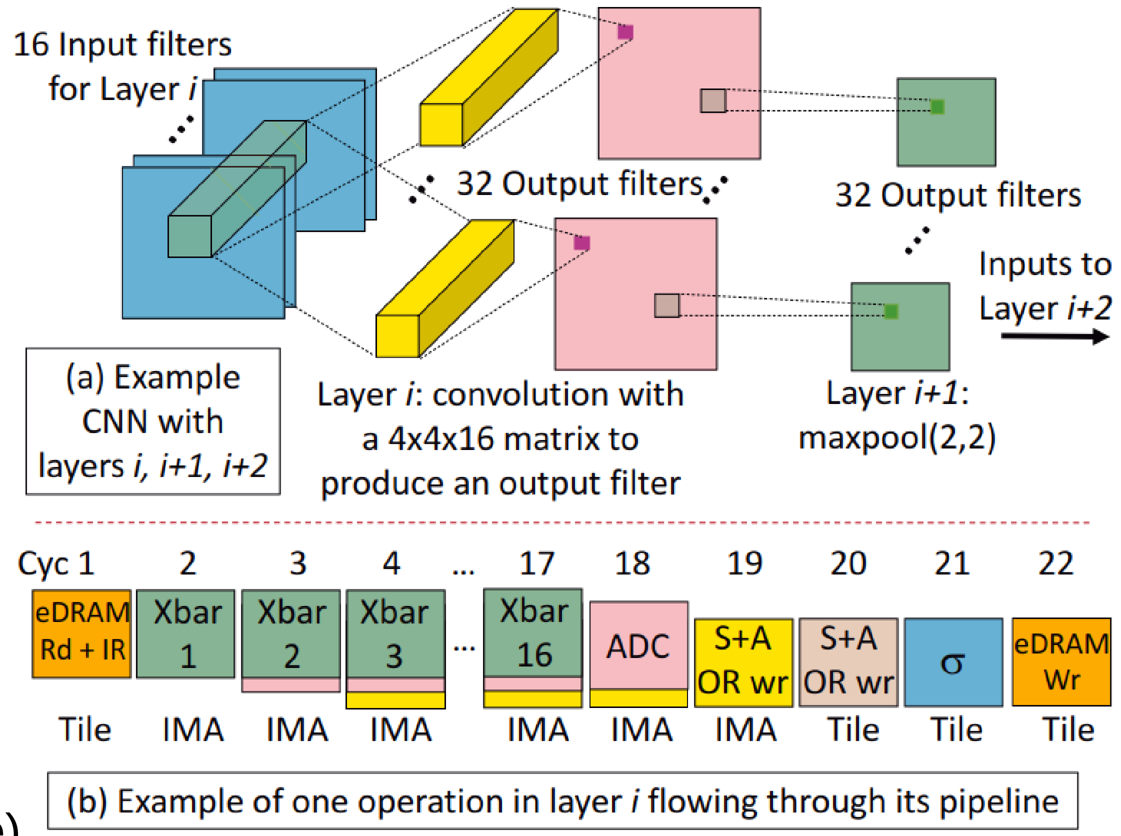


IR	– Input Register
OR	– Output Register
MP	– Max Pool Unit
S+A	– Shift and Add
σ	– Sigmoid Unit
XB	– Memristor Crossbar
S+H	– Sample and Hold
DAC	– Digital to Analog
ADC	– Analog to Digital

Shafiee et al. ACM SIGARCH 2016

Convolution in memory

- A 4x4 kernel on 16 filters
  → 4x4x16=256 multiply-add op's.
  → 256 matrix rows
- 32 output filters → 32 parallel calculations
- 16 bit precision → eight 2 bit memristors
- → 32*8 = 256 columns calculated in parallel
- Each crossbar = 128x128
  → four arrays needed, i.e. 1 IMA.



(b) Example of one operation in layer *i* flowing through its pipeline

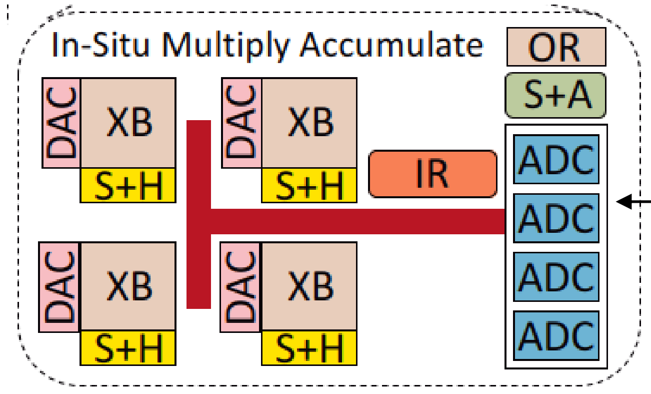
1. 256 16bit inputs read from eDRAM → IMA IR (1 cycle)
2. Send 1 bit x 256 inputs at a time through array (16 cycles)
3. Digitize (ADC) and merge output (S+A)
4. …

22 cycles (a' 100 ns) to complete convolution of one layer (2.2 µs)
Q: Why is this better than on GPU?

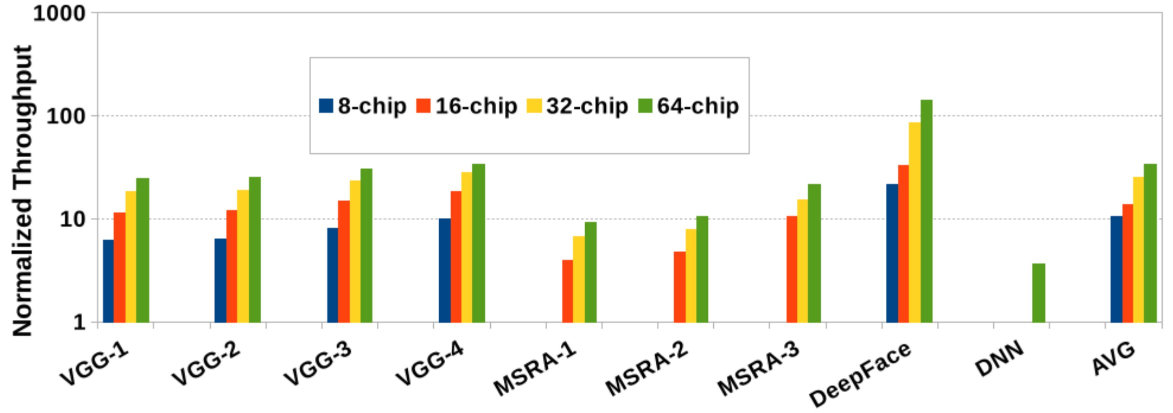
Performance benchmark

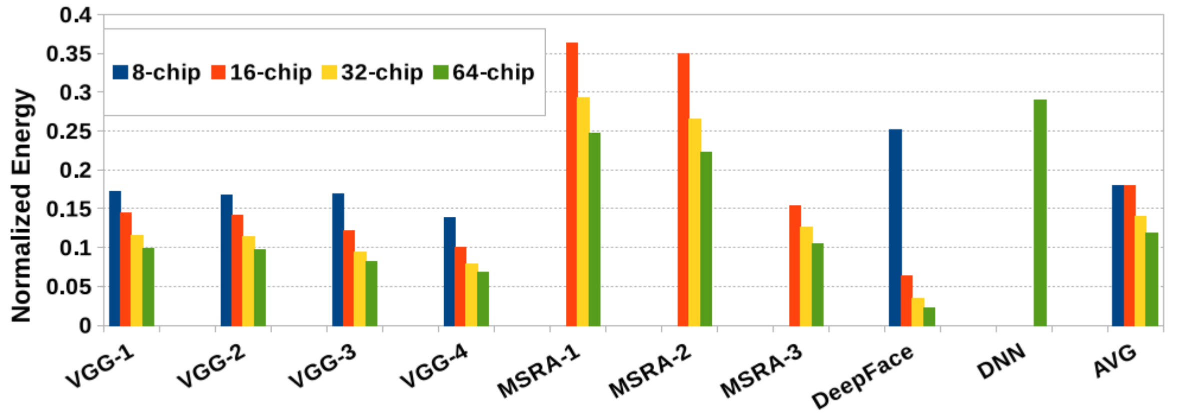
- Beats state-of-the-art GPU and TPU on power efficiency (x2)

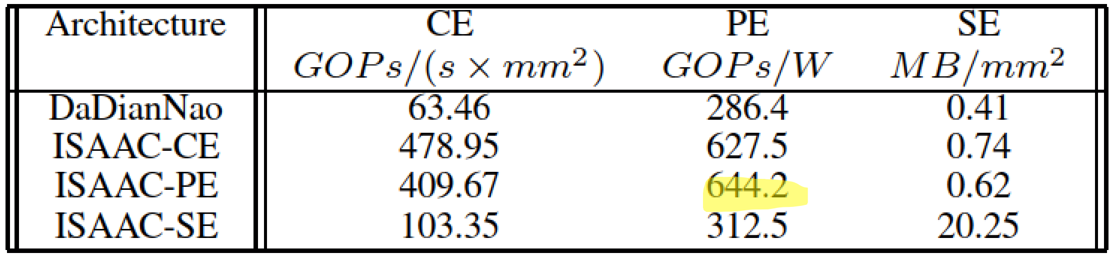
- The ADC, DRAM buffer etc reduces speed by 4x.
- ADC is power hungry too.
- But design is from 2016.
  → Potential for even better performance!



ADC limits power consumption!







Architecture	CE $GOPs/(s \times mm^2)$	PE $GOPs/W$	SE MB/mm^2
DaDianNao	63.46	286.4	0.41
ISAAC-CE	478.95	627.5	0.74
ISAAC-PE	409.67	644.2	0.62
ISAAC-SE	103.35	312.5	20.25

Nvidia Tesla V100 416 gigaflops/W
TPU 3.0 350 gigaflops/W (est.)

FOR 2021:

- Ref: https://ieeexplore.ieee.org/document/8662395 demonstrates 53 Tops/W efficiency!
- In addition: https://arxiv.org/ftp/arxiv/papers/1909/1909.07514.pdf
- Prototype chip measurements show that the proposed design achieves high binary DNN accuracy of 98.5% for MNIST and 83.5% for CIFAR-10 datasets, respectively, with energy efficiency of **24 TOPS/W** and **158 GOPS throughput**. This represents 5.6X, 3.2X, 14.1X improvements in throughput, energydelay product (EDP), and energy-delay-squared product (ED2 P), respectively, compared to the state-of-the-art literature