Hardware-friendly AI algorithms: Ternary Neural Networks

Frédéric Pétrot, Adrien Prost-Boucle, Alban Bourge, Liliana Andrade, Thomas Baumela, Ana Pinzari

TIMA Lab, Univ. Grenoble Alpes, CNRS, Grenoble INP

October 27th 2021

Hipeac Computing Systems Week

Biology vs Electronics

The Brain: the Ultimate Autonomous System

- 1,2 to 1,4 kg, 1260 cm^3
- Consumes between 15 and 30 Watts
- 86×10^9 neurons, $\approx 10^{12}$ synapses

Biology vs Electronics

The Brain: the Ultimate Autonomous System

- 1,2 to 1,4 kg, 1260 cm^3
- Consumes between 15 and 30 Watts
- 86×10^9 neurons, $\approx 10^{12}$ synapses

J. Vitti and D. Silverman, *"Bart the Genius"*, The Simpsons, 1990

Biology vs Electronics

The Brain: the Ultimate Autonomous System

- 1,2 to 1,4 kg, 1260 cm^3
- Consumes between 15 and 30 Watts
- $86{\times}10^9$ neurons, $\approx 10^{12}$ synapses

J. Vitti and D. Silverman, *"Bart the Genius"*, The Simpsons, 1990

Human Brain Project (EU Flagship)

- 16,000 neurons per 1 Watt chip
- 5.375 MW/brain
  (Bugey-1 Nuclear power plant: 540 MW, ≈100 brains)
- ($1.2{\times}10^9$ € from Europe: ≈ 1.4 cent of €/neuron)

Biology vs Electronics

The Brain: the Ultimate Autonomous System

- 1,2 to 1,4 kg, 1260 cm^3
- Consumes between 15 and 30 Watts
- $86{\times}10^9$ neurons, $\approx 10^{12}$ synapses
- (Energy comes from eggs, honey and mushrooms)

J. Vitti and D. Silverman, "*Bart the Genius*", The Simpsons, 1990

Human Brain Project (EU Flagship)

- 16,000 neurons per 1 Watt chip
- 5.375 MW/brain
  (Bugey-1 Nuclear power plant: 540 MW, ≈100 brains)
- ($1.2{\times}10^9$ € from Europe: ≈ 1.4 cent of €/neuron)

Biology vs Electronics

The Brain: the Ultimate Autonomous System

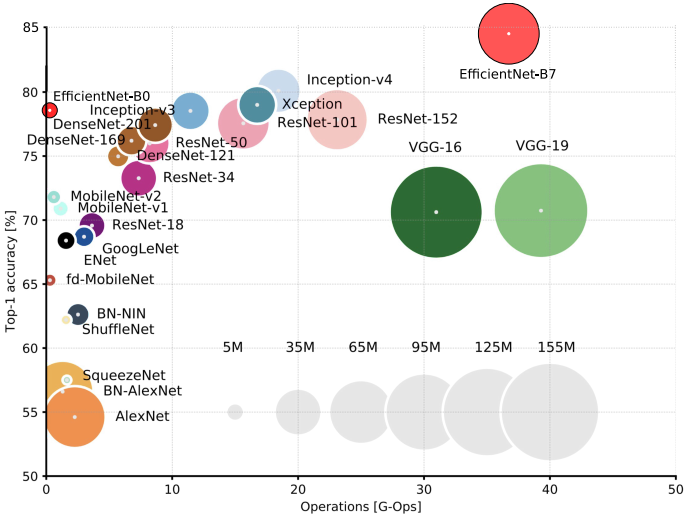
- 1,2 to 1,4 kg, 1260 cm^3
- Consumes between 15 and 30 Watts
- $86{\times}10^9$ neurons, $\approx 10^{12}$ synapses
- (Energy comes from eggs, honey and mushrooms)

J. Vitti and D. Silverman, "*Bart the Genius*", The Simpsons, 1990

Human Brain Project (EU Flagship)

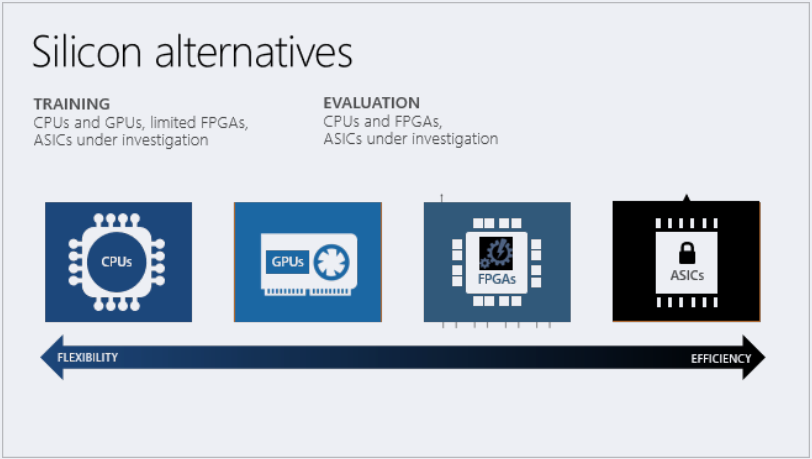
- 16,000 neurons per 1 Watt chip
- 5.375 MW/brain
  (Bugey-1 Nuclear power plant: 540 MW, ≈100 brains)
- ($1.2{\times}10^9$ € from Europe: ≈ 1.4 cent of €/neuron)

CNN Models: Accuracy, Operations and Weights



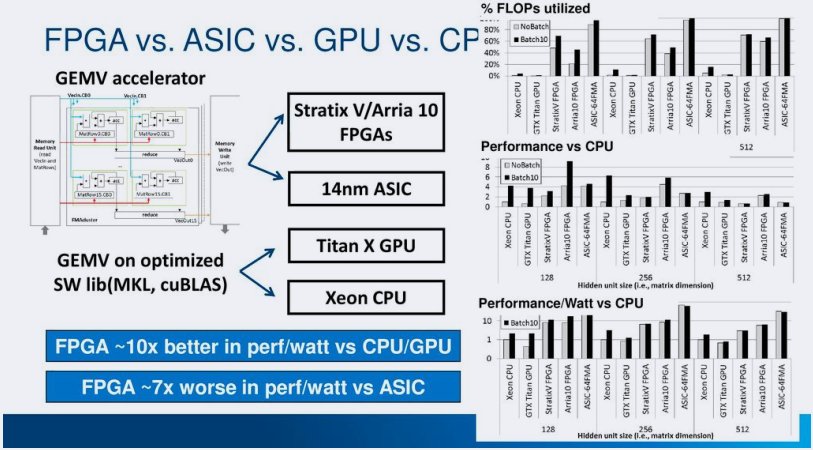
A. Canziani, E. Culurciello, A. Paszke, *"An Analysis of Deep Neural Network Models for Practical Applications"*, 2017 (EfficientNet-B0/B7 added by us)

What's the interest?



Microsoft Azure Machine Learning Documentation

What's the interest?

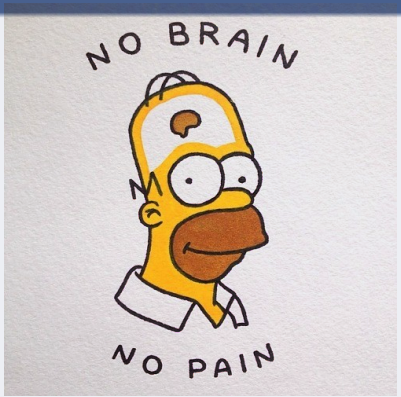


E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, D. Marr, Intel Labs

Hardware Accelerated Neural Network

What are the constraints?

- Accuracy needs depend on the application
- Silicon resources:
  ⇒ Computations to perform
  ⇒ Weights storage and access
- Energy efficiency
  Typical constraints :
  - 10-100 uW for wearables,
  - 10-100 mW for phones,
  - 1-10 W for plugged edge devices
  - 100-1000 W for plugged cloud devices



Computation Demanding

Inference involves a lot of computation...

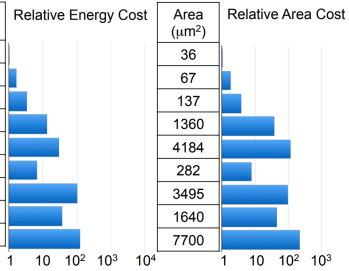
- Elevated number of floating point (FP) operations

  0.5G ≤ Nb of FLOPs ≤ 40G

- Floating point operations are energy and area costly

(My 4 core-i7 PC ~120 GFLOPs ⇒ 30 GFLOPs/core.)

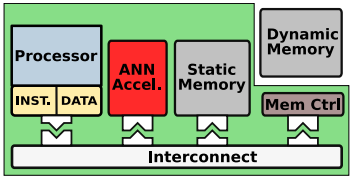
Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7



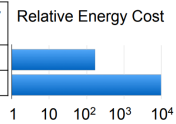
"Hardware Architectures for Deep Neural Networks", ISCA Tutorial, 2017

Memory demanding

Inference involves a lot of memory access...



Operation:	Energy (pJ)
32b SRAM Read (8KB)	5
32b DRAM Read	640



"Hardware Architectures for DNN", ISCA Tutorial, 2017

- Memory stores millions of (64-bit) weights
  $\Rightarrow$ 4M (GoogLeNet), 60M (AlexNet), 130M (VGG)
- Memory access becomes the bottleneck
  $\Rightarrow$ Each op needs 2 operands and produces a result
- Power consumption is high

Coping with GFLOPs and GBytes

Alternatives: trade FLOPs for (some) accuracy loss

Simplify the operations

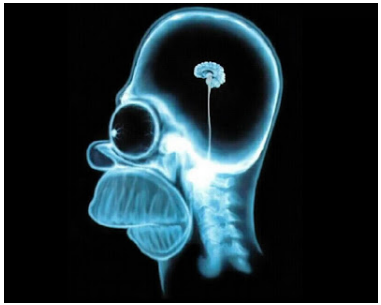
- Avoid sigmoid, average pooling and stuff
- FP arithmetic is not HW friendly

Alternatives: trade bytes for (some) accuracy loss

- Use “small” data types
  $\Rightarrow$ In any case no (16)/32/64-bit quantities

Alternatives: re-architect the “system”

- Integrate many memory cuts with processing elements and use them wisely
- Integrate computation into the memory itself



K. Usher, *”The Dwarf in the Dirt”*, Bones, 2009

CNN Models: Target Implementation

In Memory Computing: Emerging Technologies

Left to brigther guys!

ASIC vs FPGA: Business as usual

ASIC

- \- Hardwiring ANN too risky
  $\Rightarrow$ New and better NN every other day
- \+ VVM/MVM/MMM accelerators
- \+ Systolic arrays

FPGA

- \+ Great digital PIM
- \+ Huge internal memory throughput
- \- Limited number of ad-hoc resources
- \- No floating point operations

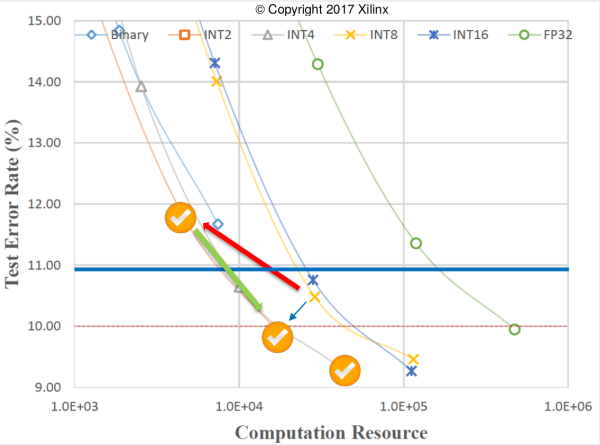
Learning for Quantized NN

Quantization levels and accuracy...

Just reducing precision, reduce hardware cost & increases error

Recuperate accuracy by **retraining** & **increasing network size**

1b, 2b and 4b provide pareto optimal solutions

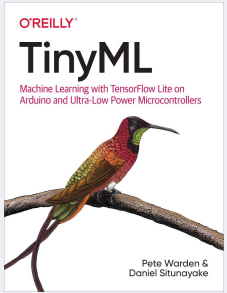


Kees Vissers, *"A Framework for Reduced Precision Neural Networks on FPGAs"*, MPSOC, 2017

Learning for Quantized NN

Tools are there

- Tensorflow/Keras
  - tflite: 8-bit Post-Training Quantization or Quantization Aware Training
    Targets micro-controllers, useful for hardware too!
  - qkeras: Quantization Aware Training for $1 \leq n \leq 8$ bits
    Implements (partially) SoA algorithms
- Pytorch
  - quantization API: 8-bit quantization
  - Brevitas: binary→byte quantization
- Larq: binary only
- ...



Quantization aware training: a bit of literature

"Less bit per weight/bias and activation"

- "Deep compression", 2016, NVIDIA and Stanford
  Quantize only weights, to 5-bit, > 5000 cites
- "XNOR-net", 2016, Allen AI and U. Washington
  Binary CNNs, 1-bit, > 3000 cites
- "Binarized neural networks", 2016, Univ. Montréal
  Binary weights and activations, 1-bit, ~ 3000 cites
- "DoReFa-net", 2018, Megvii
  Framework for "Quantization Aware Training", *n*-bit, > 1000 cites
- ...

Example: Ternary Convolutional Neural Networks

Principle

- Ternarize $\{-1, 0, 1\}$ parameters and activations[a]
- Sweet spot between resource usage and accuracy

[a] *"Perhaps the prettiest number system of all is the balanced ternary notation."* Donald Knuth, The Art of Computer Programming, Volume 2: Seminumerical algorithms.

Ternarization Performance - Error Rates (%)

	CIFAR10	SVHN	GTRSB
NN-64			
Float	13.11	2.40	0.96
Ternary	13.29	2.40	1.05

	CIFAR10	SVHN	GTRSB
NN-128			
Float	10.48	2.27	0.76
Ternary	10.61	2.30	0.80

Experiments with Ternary Networks

Multiple networks

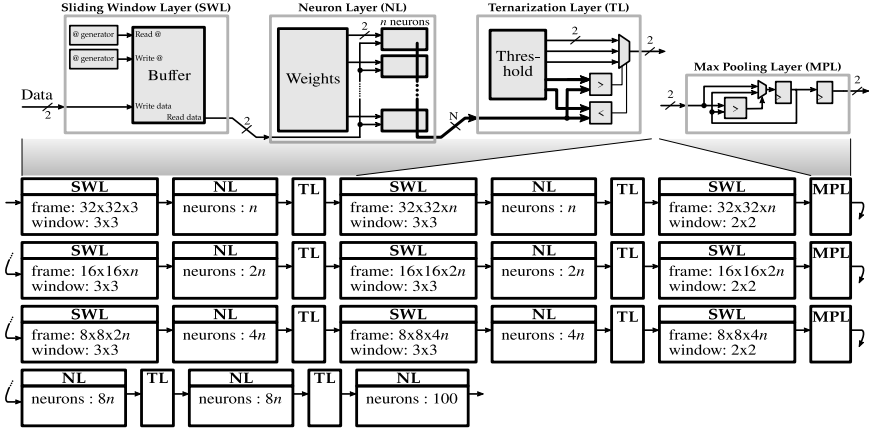
- VGG-like networks
- two geometries : NN-64 and NN-128
- multiple “acceleration factors” inside network (ranging 1 to 256)
- tradeoff area/throughput/energy

Automation (kind of ...)

- *handmade* generic hardware building blocks (`vhdl`)
- automatically generated networks
- customizable home-made tools (old school `C` and `tcl`)

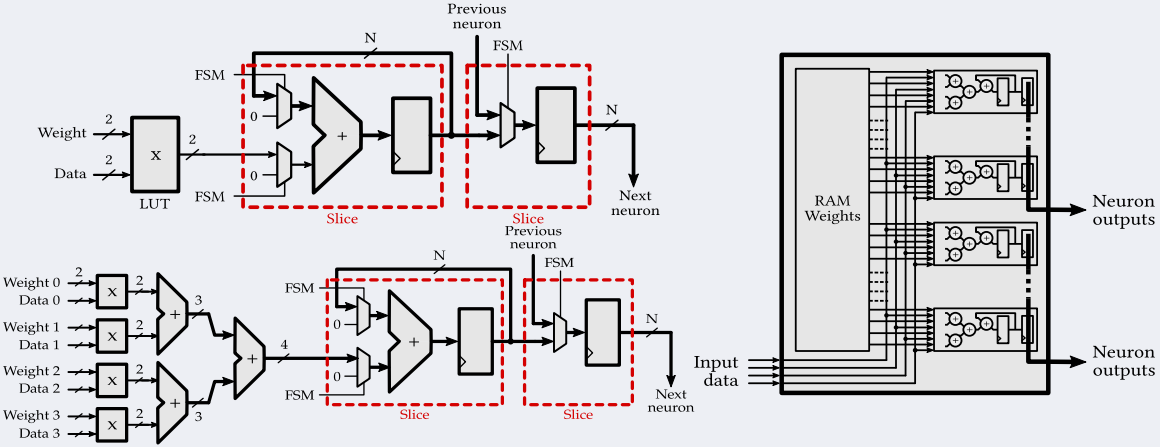
Overview

- 12 theoretical layers, 29 physical layers (+30 "glue" fifos)
- NN64: 1930 neurons, 3.5 M parameters, NN128: 3850 neurons, 14 M parameters



Goal of ternarization: *have parameters fit in FPGA distributed memories*

Neurones and Parallelism



Max NN-64 “acceleration factor” that fits on a VC709: 128

Acceleration factors

- Base implementation: at most 1 activation transfered between layers in 1 cycle
  But some layers take more time than others to compute ⇒ stalls
- Use parallelism at layer level:
  Transfering "several" activations in 1 cycle in/out of bootleneck layers

NN size	Acc. factor	Parallelism per layer (in/out)											
		NL1	*NL2*	*MPL1*	*NL3*	*NL4*	*MPL2*	*NL5*	*NL6*	*MPL3*	*NL7*	*NL8*	*NL9*
	1	-	-	-	-	-	-	-	-	-	-	-	-
	2	-	2 / 1	-	-	-	-	-	-	-	-	-	-
	4	-	4 / 1	-	-	2 / 1	-	-	-	-	-	-	-
	8	-	8 / 1	-	2 / 1	4 / 1	-	-	2 / 1	-	-	-	-
64	16	1 / 2	16 / 2	2 / 1	4 / 1	8 / 1	-	2 / 1	4 / 1	-	-	-	-
	32	2 / 4	32 / 4	4 / 1	8 / 2	16 / 2	2 / 1	4 / 1	8 / 1	-	-	-	-
	64	3 / 8	64 / 8	8 / 2	16 / 4	32 / 4	4 / 1	8 / 2	16 / 2	2 / 1	-	-	-
	128	9 / 16	192 / 16	16 / 4	32 / 8	64 / 8	8 / 2	16 / 4	32 / 4	4 / 1	-	-	-
	256	27 / 32	576 / 32	32 / 8	64 / 16	128 / 16	16 / 4	32 / 8	64 / 8	8 / 2	2/1	-	-

Squeezing High-Efficiency TCNN in FPGA: Adder Trees

Ternary Adders

- Sum of *trits* ≠ Sum of bits
  With $(x, y) \in \{-1, 0, 1\}^2, x + y \in \{-2, -1, 0, 1, 2\}$

LUT savings with optimized ternary adder tree

Number of inputs	4	8	16	32	64	128	192	256
Generic 2-bit radix-2 adder tree (LUT)	6	21	44	90	181	380	568	757
Optimized ternary adder (LUT)	4	9	21	44	90	184	274	371
Savings	33.3%	57.1%	52.3%	51.1%	50.3%	51.6%	51.8%	51.0%

Overall LUT savings when using optimized ternary adder trees

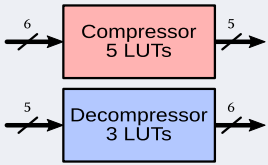
Acc. factor	4	8	16	32	64	128	256
Savings for NN-64	0.18%	1.38%	4.61%	10.9%	17.3%	24.1%	32.0%
Savings for NN-128	0.22%	1.71%	5.63%	12.9%	19.6%	25.7%	

Squeezing High-Efficiency TCNN in FPGA: Weight Compression

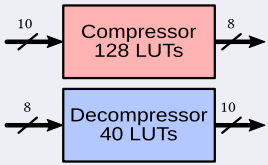
Trits encoding

- Naïve encoding: 2 bits to encode 1 trit ⇒ suboptimal
  Ex.: 3 trits encoded on 6 bits while $3^3 = 27$ combinations ⇒ encodable on 5 bits
- Optimal number of bits per trits: $b = \lceil \log_2 (3^T) \rceil = \lceil T \times \log_2(3) \rceil$
- Minimal number of bits per trits: $b/T \approx \log_2(3) \approx 1.585$ bits
  ⇒ Maximum saving ≈ 20.75 % (Shannon limit)

Interesting cases



3 trits / 5 bits ⇒ 16 % saving



5 trits / 8 bits ⇒ 20 % saving

[Interlude] Compression challenge!

Find the "best" mapping of 3 trits on 5 bits:
⇒ any order will do, just find the one that minimizes the bits→trits equations

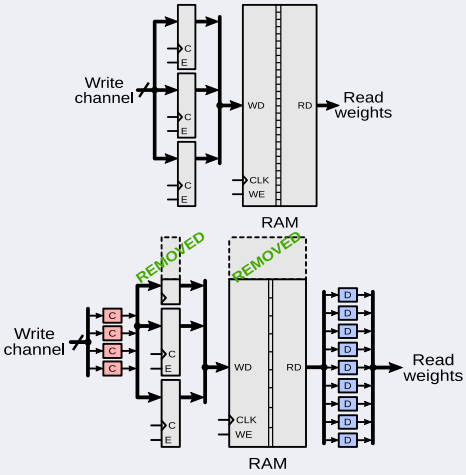
t_5 ... t_0	$b_4 \dots b_0$
000000	-0001
000001	-0010
000011	-0110
000100	00-00
001100	01-00
010101	00011
010100	00101
010111	00111
010000	01001
010001	01010
011101	01011
011100	01101
010011	01110
011111	01111

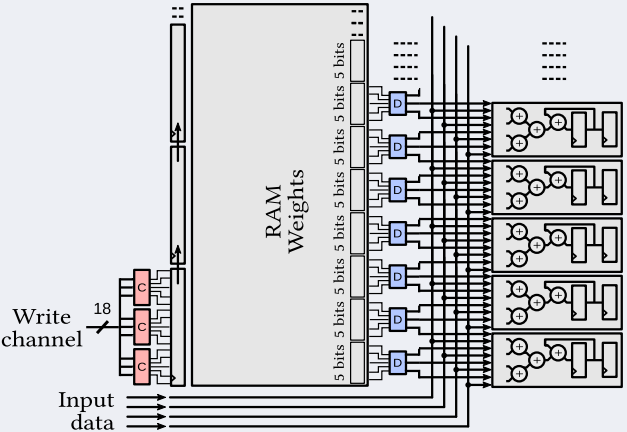
t_5 ... t_0	$b_4 \dots b_0$
000101	10000
110101	10011
000111	10100
110100	10101
110111	10111
001101	11000
110000	11001
110001	11010
111101	11011
001111	11100
111100	11101
110011	11110
111111	11111

$$x_0 = b_0 \wedge (b_1 \vee b_2)$$
$$x_1 = b_0 \vee b_1$$
$$t_0 = b_1 \vee \overline{b_0} \wedge b_4$$
$$t_1 = t_0 \wedge b_2$$
$$t_2 = x_0 \vee \overline{x_1}$$
$$t_3 = t_2 \wedge b_3$$
$$t_4 = x_0 \vee b_3 \wedge x_1,\ t_5 = t_4 \wedge b_4$$

Squeezing High-Efficiency TCNN in FPGA: Weight Compression

Compression of weights





Measured Results for NN-64 on Xilinx XC7VX690T@250 MHz (VC709)

Acc. factor	Resource usage LUT (logic)	LUTRAM	BRAM 18k	FF
128	170555 (39.4%)	37402 (21.47%)	1410 (48.0%)	321352 (37.1%)
256*	302748 (69.9%)	106168 (60.9%)	2920 (96.7%)	640849 (74.0%)

NN-64 with parallelism degree 128

- Uses half of the FPGA resources, reaches max throughput of 60.2k fps (32×32)
- (LUT+B)RAM throughput of 18.7 Tb/s (290 Gb/s for FC layers only[a])
- End to end latency including PCIe + RIFFA: 135 µs
- Max performance: 18.7 T(T)OP/s (9.33 T(T)MAC/s)
- Power @ max performance: 11.5 W (Idle FPGA ≈ 2 W)
- Peak efficiency of 5226 fps per Watt $\Rightarrow$ 1.62 T(T)OP/s/W or 810 G(T)MAC/s/W

[a] VC709 DRAM throughput ≈ 204 Gb/s

Take away

- **Learning for low bit-size**
  It's coming and it's useful, for both HW and SW
  ⇒ tensorflow lite works out of the box for 8-bit
  ⇒ 2/4-bit SIMD in some micro-controllers targeting lower energy footprint
- **FPGAs are a very good fit for ANN if parameters fit in internal memory**
  ⇒ Extreme quantization needed, but unbeatable throughput
  ⇒ Specific access shemes if network doesn't fit-in, ASIC-like solutions
- **ASICs follow a very different (but equally useful) architectural path**
  ⇒ Quanization limited to bytes for now
  ⇒ Hardwired networks for high-volume ultra-low power applications
  ⇒ Tensor Processor Units otherwise, need controller to fetch and schedule