




Digital Hardware Acceleration for Neural Networks: Energy Constraints and Performance

Frédéric Pétrot

Univ. Grenoble Alpes, CNRS, Grenoble INP*,

/Ensimag , F-38000 Grenoble, France

 tima.imag.fr/sls/people/petrot

 frederic.petrot@univ-grenoble-alpes.fr

*Institute of Engineering Univ. Grenoble Alpes



mi2i
Grenoble Alpes

Multidisciplinary Institute
In Artificial Intelligence



**GRENOBLE
INP
UGA**

Walking in the wild

Yesterday!

Peyresq 2022



Walking in the wild

What you escaped!

Sarcenas 2013



Breaking news: Computers actually need energy!

- ▶ Power consumption primer
- ▶ Orders of magnitude

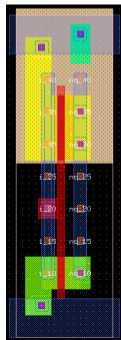
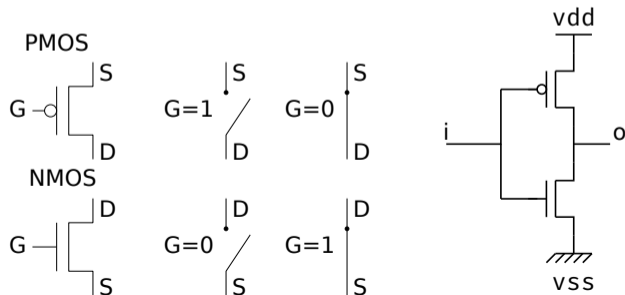
Hardware Accelerated AI^H ML^H NN

- ▶ DNN as seen by HW guys
- ▶ What is HW by the way and why use it?
- ▶ Ways to "enhance" NN computations
- ▶ Architecture Zoo
- ▶ SW frameworks

Electronic Devices and Power Consumption

Current Computer Technology

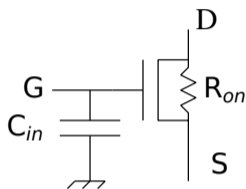
- ▶ Modern electronic: semi-conductor based devices
- ▶ Digital computations based on (nano-)electronic devices working in *base two* Binary digIT : $\{0, 1\}$ (a.k.a *bit*)
- ▶ Device \equiv CMOS transistor used as a switch



Switch Electrical Behavior

CMOS transistor \neq ideal switch

Second order model with parasitics:



- ▶ C_{in} : grid capacitance,
- ▶ R_{in} : input resistance $\rightsquigarrow \infty$
- ▶ R_{on} : resistance when closed
- ▶ R_{off} : resistance when open $\rightsquigarrow \infty$
- ▶ causes of non-instantaneous transitions

Charging a capacitance through a resistance:

$$V_{out} = V_{dd} \times (1 - e^{\frac{-t}{RC}})$$

Discharging a capacitance through a resistance:

$$V_{out} = V_{dd} \times (e^{\frac{-t}{RC}})$$

Power consumption: $P \propto CV^2f$

Why CMOS?

- ▶ Very small dimensions

Mass production smallest transistor width in 2021/2022 : 5 nm

- TSMC - Apple A14/A15, Huawei Mate 40, HiSilicon Kirin 9000 -: 173 MTr/mm²
Apple M1 (16 Mrds Trs): 134 MTrs/mm² actual transistor density
- Samsung - Exynos, Snapdragon 8xx, Nvidia Hopper -: 127 MTr/mm²
- Intel is arriving, ...

Si atomic radius is ≈ 0.11 nm !

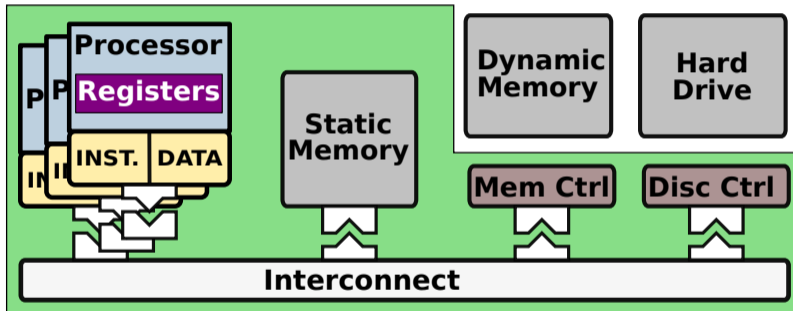
3 nm production announced for 2022 (TSMC) / 2023 (Samsung, Intel)

- ▶ Very high yield
- ▶ Boolean logic computation in $\leq 10 \times 10^{-12}$ seconds
- ▶ Power consumption $\leq 1 \times 10^{-15}$ joules / transition*
- ▶ Allows us to reason with zillions of 0 and 1

⇒ CMOS: hyper-hegemonic digital technology

*1 joule is the energy provided by 1 watt during 1 second.

Computer System Overview



Where to place data

Order of magnitude for size and access time

Registers

	1980		2020	2020 vs 1980
T_{acc} (ns)	300		0.25	÷1200
Typ. size (B)	64		256	×4

Static memory

	1980		2020	2020 vs 1980
\$/MB	19,200		5	÷3840
T_{acc} (ns)	300		1	÷300
Typ. size (KB)	32		8192	×256

Dynamic memory

	1980		2020	2020 vs 1980
\$/MB	8,800		0.003	÷2,930,000
T_{acc} (ns)	375		30	÷12.5
Typ. size (MB)	0.064		32,000	×500,000

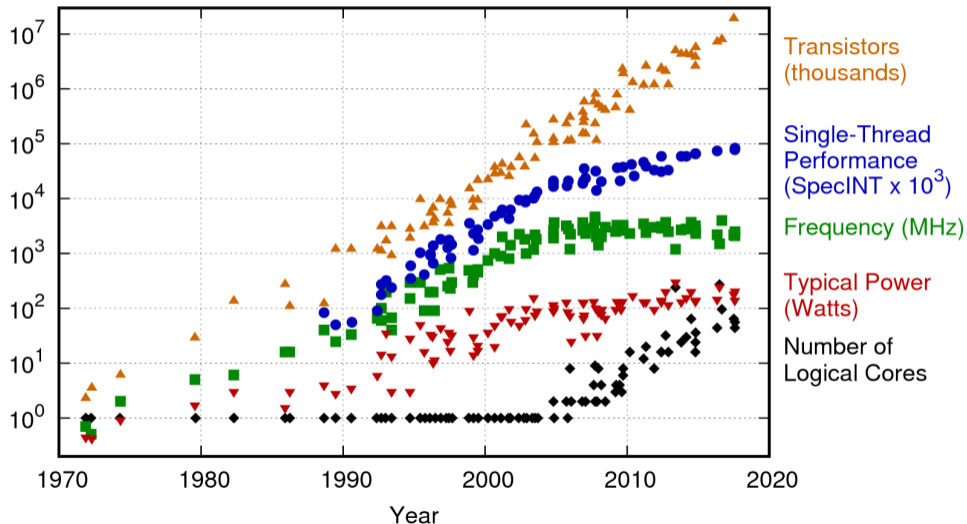
Hard drive

Metrique	1980		2020	2020 vs 1980
\$/MB	500		.000018	÷27,800,000
T_{acc} (ms)	87		12	÷7
Typ. size (GB)	0.001		8,000	×1,500,000

Sources : <https://jcmmit.net/> and various (D/S)RAM vendors web sites

Orders of Magnitude

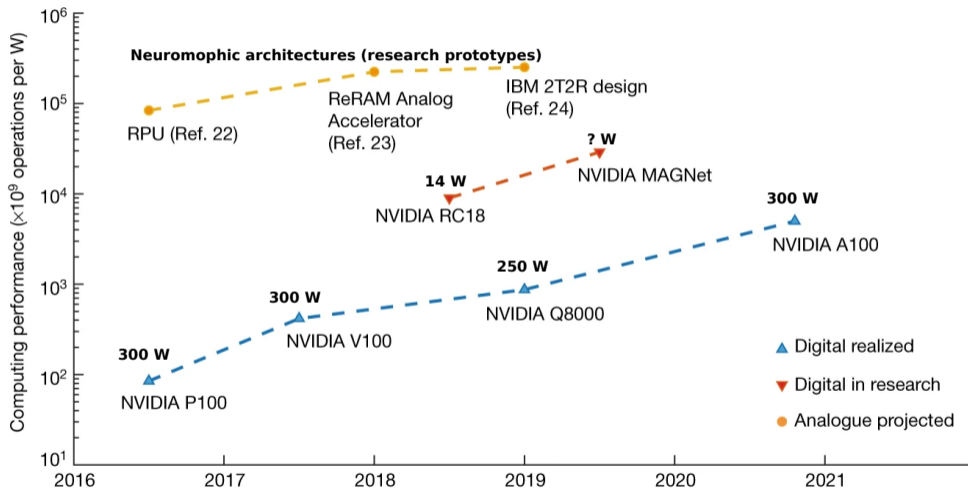
42 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Orders of Magnitude

b Hardware development



Mehonic, A., Kenyon, A.J. Brain-inspired computing needs a master plan. *Nature* 604, 255-260 (2022).

<https://doi.org/10.1038/s41586-021-04362-w>

Orders of Magnitude

Quick Summary

Raw Power Consumption:

- ▶ CPUs: 100/150 Watt
- ▶ GPUs: 250/300 Watt

Training Example

NVidia MegatronLM

- ▶ Used 45 Tera Bytes of data
- ▶ On 512 V100 NVIDIA GPUs during 9 days
- ▶ $512 \times 300 \times 9 \times 24 = 33177$ kWh
7× the energy an average French family uses per year! (4590 kWh)

⇒ We ought to do better!

Carbon Emissions and Large Neural Network Training, David Patterson et al,
<https://arxiv.org/pdf/2104.10350.pdf>

The Brain: the Ultimate Autonomous System

- ▶ 1,2 to 1,4 kg, 1260 cm³
- ▶ Consumes between 15 and 30 Watts
(max physical activity: 500 Watts)
- ▶ 86×10^9 neurons, $\approx 10^{12}$ synapses

The Brain: the Ultimate Autonomous System

- ▶ 1,2 to 1,4 kg, 1260 cm³
- ▶ Consumes between 15 and 30 Watts
(max physical activity: 500 Watts)
- ▶ 86×10^9 neurons, $\approx 10^{12}$ synapses



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

The Brain: the Ultimate Autonomous System

- ▶ 1,2 to 1,4 kg, 1260 cm³
- ▶ Consumes between 15 and 30 Watts
(max physical activity: 500 Watts)
- ▶ 86×10^9 neurons, $\approx 10^{12}$ synapses



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

Human Brain Project (EU Flagship)

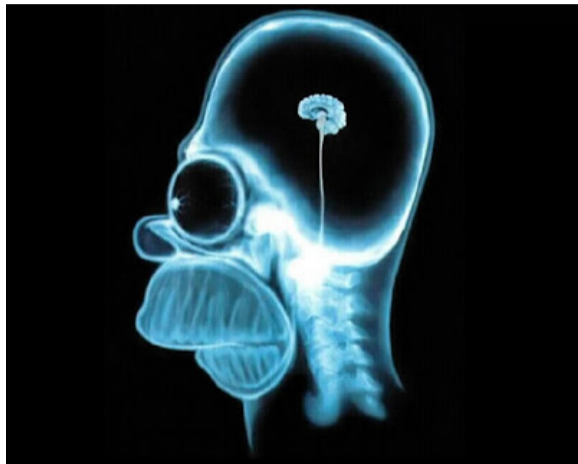
- ▶ 16,000 neurons per 1 Watt chip
- ▶ 5.375 MW/brain
(Bugey-1 Nuclear Power Plant: 540 MW, ≈ 100 brains)
- ▶ (1.2×10^9 € from Europe: ≈ 1.4 cent of €/neuron)



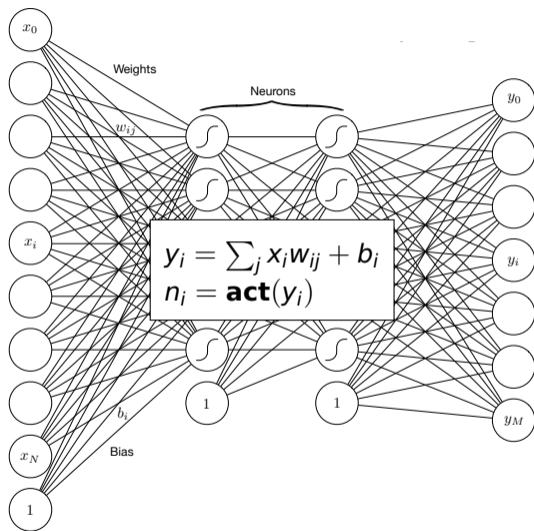
What AI do we really need?



What AI do we really need?



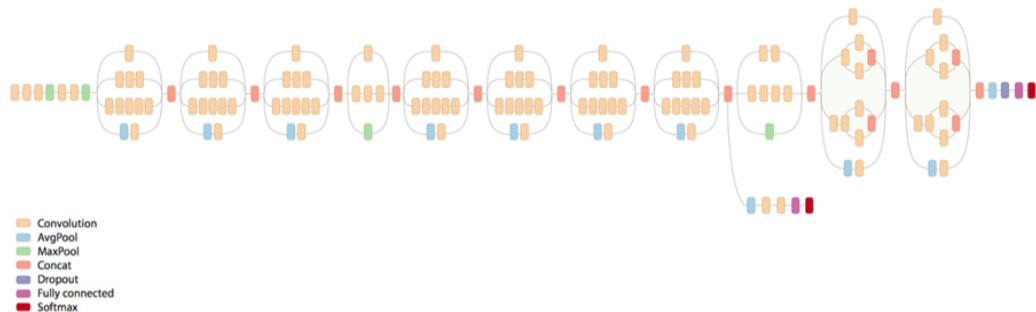
Artificial Neural Nets: Multi-Layer Perceptrons (late 1950's)



Two phases :
Training and Inference

- ▶ Elementary computations conceptually very simple
- ▶ Parameters: Weights and Biases
⇒ "Found" during training
⇒ "Used" during inference
- ▶ Available computing power limits training

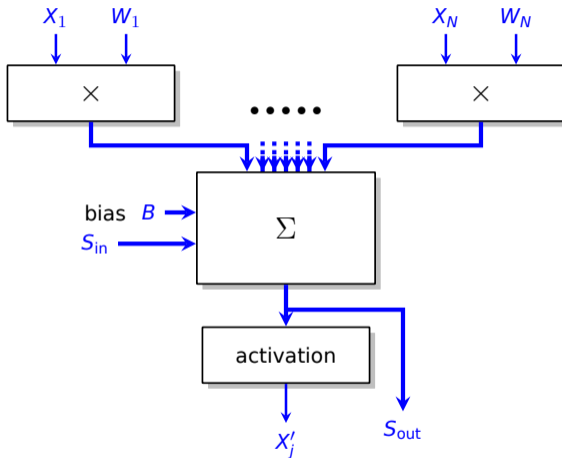
Artificial Neural Nets: Convolutional NN (Mid 2010's)



Another view of GoogLeNet's architecture.

- ▶ Same elementary operations, just repeated zillions times
- ▶ Additional inter-layer operations, still quite simple
- ▶ Training possible thanks to compute farms

Neuron Computation Sketched



Conv2D usage in Tensorflow

```
keras.layers.Conv2D(6, name='C1',  
                    kernel_size=5, strides=1, activation='relu',  
                    input_shape=train_i[0].shape, padding='valid')
```

Conv2D (naïve) software implementation in C

```
void conv2d(int in_channels, int out_channels,
            int img_size, int kernel_size,
            float input[img_size][img_size][in_channels],
            float kernel[out_channels][kernel_size][kernel_size][in_channels],
            float bias[out_channels],
            float output[img_size - 2 * (kernel_size / 2) + !(kernel_size & 1)]
                    [img_size - 2 * (kernel_size / 2) + !(kernel_size & 1)]
                    [out_channels])
{
    int fm_size = img_size - 2 * (kernel_size / 2) + !(kernel_size & 1);

    for (int k = 0; k < fm_size; k++)
        for (int l = 0; l < fm_size; l++)
            for (int o = 0; o < out_channels; o++) {
                float mac = 0;
                for (int m = 0; m < kernel_size; m++)
                    for (int n = 0; n < kernel_size; n++)
                        for (int i = 0; i < in_channels; i++)
                            mac += kernel[o][m][n][i] * input[k + m][l + n][i];
                output[k][l][o] = activation(mac + bias[o]);
            }
}
```

Conv2D software implementation compiled with gcc -03

```

conv2d:      shrq    $2, %rsi      movq    208(%rsp), %rax      leaq    (Xrdi,%rax,4), %rsi      addq    %rdi, %r15      addss   (Xrbx), %xmm1
.LFB0:      shrq    $2, %r15      movq    %rax, 64(%rsp)      xorl    %eax, %eax      movss   (Xr10,%r15,4), %xmm0      addq    $4, %rbx
    pushq   %r15      movq    %rsi, 32(%rsp)      .L16:   movl    16(%rsp), %eax      mulss   (Xr11,%rsi,4), %xmm0      cvtts2sd %xmm1, %xmm0
    movl    %esi, %eax      testl   %eax, %eax      movl    16(%rsp), %eax      addss   %xmm0, %xmm1      call    relu@PLT
    movslq  %edx, %rsi      jle     .L1      testl   %eax, %eax      .L6:    movups  (Xrsi,%rax), %xmm0      cmpl    %eax, %r13d      pxor    %xmm0, %xmm0
    pushq   %r14      movslq  %ebx, %rdx      jle     .L19      movups  (Xrcx,%rax), %xmm3      jle     .L10      movq    104(%rsp), %rdi
    movl    %edi, %r14d      subl    $1, %eax      movq    136(%rsp), %rax      movups  %xmm0, %xmm2      cltq    %rax, %rdx      cvtsi2sdl %eax, %xmm0
    movslq  %ecx, %rdi      movq    216(%rsp), %rbx      movq    80(%rsp), %rbx      addq    %rax, %rdx      movq    72(%rsp), %rax
    pushq   %r13      movq    %r14d, %xmm1      pxor    %xmm3, %xmm0      addq    %r9, %rax      addq    %rdi, %rax      movq    80(%rsp), %rax
    movslq  %r14d, %rcx      imulq   %rdi, %rdx      movq    $0, 40(%rsp)      movss   (Xr11,%rdx,4), %xmm0      addq    $4, %rax      movss   %xmm0, -4(%rax)
    pushq   %r12      movl    $0, 56(%rsp)      movl    $0, 28(%rsp)      movss   (Xr10,%rax,4), %xmm0      movq    %rbx, 64(%rsp)
    imulq   %rcx, %rsi      andl    $-4, %r13d      leaq    (%rax,%rbx,4), %r10      mulss   %xmm0, %xmm2      addq    %xmm0, %xmm1      movss   %xmm0, -4(%rax)
    leaq    0(%rcx,4), %r15      movq    %rbx, 96(%rsp)      movl    28(%rsp), %eax      addss   %xmm1, %xmm2      addss   %xmm0, %xmm1      movq    %rax, 72(%rsp)
    pushq   %rbp      movq    208(%rsp), %rbx      .p2align 4,,10      movaps  %xmm0, %xmm1      .L10:   cmpq    120(%rsp), %rbx
    imulq   %rdi, %rcx      movl    %r13d, %ebp      .p2align 3      unpckhps %xmm0, %xmm1      leal    1(%r8), %eax      jne     .L16
    pushq   %rbx      leaq    4(%rbx,%rax,4), %rax .L14:   shufps  $255, %xmm0, %xmm0      addq    (%rsp), %rcx      addl    $1, 20(%rsp)
    movq    %rdi, %rbx      movq    %rdx, 128(%rsp)      movl    56(%rsp), %ebx      addss   %xmm2, %xmm1      cmpl    %eax, 16(%rsp)      movq    112(%rsp), %rdi
    subq    $152, %rsp      movq    %rax, 120(%rsp)      movq    8(%rsp), %rdi      addss   %xmm0, %xmm1      je      .L8      movl    20(%rsp), %eax
    movl    %edi, 16(%rsp)      leal    -1(%r14), %eax      movslq  %eax, %r9      cmpq    %r14, %rax      movl    %eax, %r8d      addq    %rdi, 88(%rsp)
    imulq   %rcx, %rdi      movl    %eax, 24(%rsp)      xorl    %r8d, %r8d      jne     .L6      cmpl    %eax, 60(%rsp)
    movq    %rsi, 48(%rsp)      movl    %r14d, %eax      imulq   %rcx, %r9      movl    %r12d, %eax      jmp     .L12      jne     .L17
    leaq    0(%rcx,4), %rsi      shrl    $2, %eax      addl    %eax, %ebx      cmpl    %r12d, %r13d      .p2align 3      movl    %r12d, %eax
    movslq  %eax, %rcx      subl    $1, %eax      movslq  %ebx, %rbx      je      .L10      .L8:    movl    28(%rsp), %ebx      addl    $1, 56(%rsp)
    movq    %r8, 8(%rsp)      addq    $1, %rax      imulq   48(%rsp), %rbx      movslq  %r8d, %rdi      movq    128(%rsp), %rdi
    movq    %rdi, 104(%rsp)      addq    $4, %rax      leaq    (Xrdi,%rbx,4), %r11      movq    32(%rsp), %rcx      movq    %rbp, %rdx      movq    %rbp, %r12
    leaq    0(%rcx,4), %rdi      movq    %rax, %r12      movq    40(%rsp), %rdi      addq    %rcx, 40(%rsp)      addq    %rdi, 96(%rsp)
    movl    %ebx, %ecx      movl    %r14d, %eax      leaq    (Xr10,%rdi,4), %rcx      leal    1(%rbx), %eax      movl    %eax, %ebp
    shrl    $31, %ecx      movl    %eax, %r13d      .p2align 4,,10      cmpl    %r8d, %ebx      movl    56(%rsp), %eax
    movq    %r9, 136(%rsp)      movq    %r12, %r14      .p2align 3      je      .L11      cmpl    %eax, 60(%rsp)
    movq    %ebx, %ecx      movq    %r15, %r12      .L12:   testl   %r13d, %r13d      addq    %rdx, %rsi      movl    %eax, 28(%rsp)      jne     .L4
    notl    %ebx      .L4:    testl   %r13d, %r13d      addq    %rdi, %r15      jmp     .L14      .L1:    addq    $152, %rsp
    movq    %r15, (%rsp)      movq    96(%rsp), %rax      jle     .L10      movss   (Xr10,%r15,4), %xmm0 .L18:   xorl    %eax, %eax
    andl    $-2, %ecx      movl    $0, 20(%rsp)      movl    20(%rsp), %eax      mulss   (Xr11,%rsi,4), %xmm0      popq    %rbx
    movq    %rdi, 112(%rsp)      movq    %rax, 88(%rsp)      cmpl    $2, 24(%rsp)      leal    1(%rax), %esi      movslq  %edx, %rdx      popq    %rbp
    subl    %ecx, %edx      movl    %ebp, %eax      leal    (%r8,%rax), %edx      addss   %xmm0, %xmm1      jmp     .L5      popq    %r12
    movl    %ebx, %ecx      movq    %r12, %rbp      jbe     .L18      cmpl    %esi, %r13d      .L19:   jmp     .L5      popq    %r13
    andl    $1, %ecx      movl    %eax, %r12d      movslq  %edx, %rdx      jle     .L10      .L11:   pxor    %xmm1, %xmm1      popq    %r14
    leal    (Xrdx,%rcx), %ebx .L17:   movq    8(%rsp), %rdi      movslq  %esi, %rsi      movq    64(%rsp), %rbx      popq    %r15
    movl    %ebx, 60(%rsp)      movq    $0, 80(%rsp)      movq    %rdx, %rax      addl    $2, %eax      movq    %xmm0, %xmm0
    testl   %ebx, %ebx      movq    88(%rsp), %rax      imulq   %rbp, %rax      leaq    (Xrsi,%r9), %r15      pxor    %xmm0, %xmm0
    jle     .L1      movq    %rax, 72(%rsp)      addq    %rbx, %rax      addq    %rdx, %rsi      movl    $1, %eax

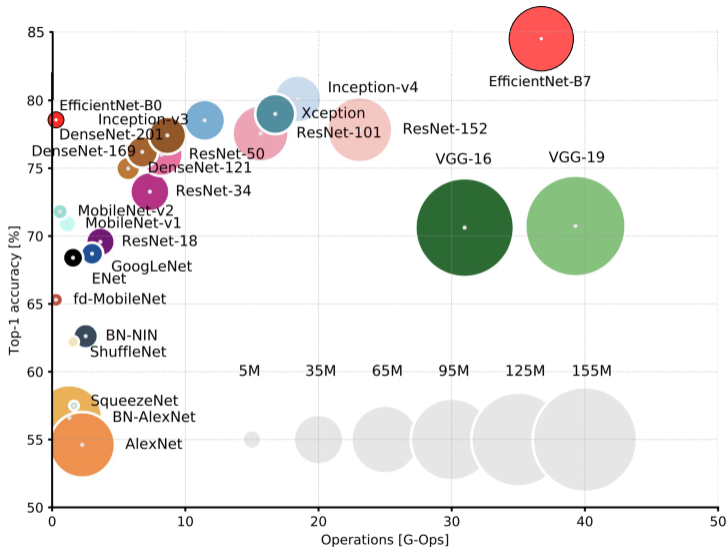
```

Implementation matters!

Table 4: Normalized global results for Energy, Time, and Memory

Total					
Energy (J)		Time (ms)		Mb	
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

CNN Models: Accuracy, Operations and Parameters



A. Canziani, E. Culurciello, A. Paszke, "An Analysis of Deep Neural Network Models for Practical Applications", 2018 (EfficientNet-B0/B7 added by myself)
<https://culurciello.medium.com/analysis-of-deep-neural-networks-dcf398e71aae>

Hardware Accelerated Neural Network

What's the interest?

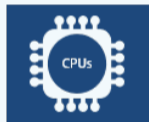
Silicon alternatives

TRAINING

CPUs and GPUs, limited FPGAs,
ASICs under investigation

EVALUATION

CPUs and FPGAs,
ASICs under investigation



FLEXIBILITY

EFFICIENCY

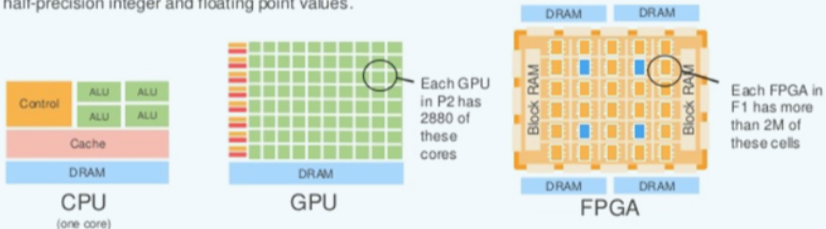
Microsoft Azure Machine Learning Documentation

Hardware Accelerated Neural Network

What's the interest?

Parallel Processing in GPUs and FPGAs

A **GPU** is effective at processing the same set of operations in parallel – single instruction, multiple data (SIMD). A GPU has a well-defined instruction-set, and fixed word sizes – for example single, double, or half-precision integer and floating point values.

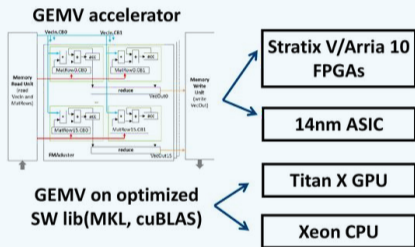


An **FPGA** is effective at processing the same or different operations in parallel – multiple instructions, multiple data (MIMD). An FPGA does not have a predefined instruction-set, or a fixed data width.

Hardware Accelerated Neural Network

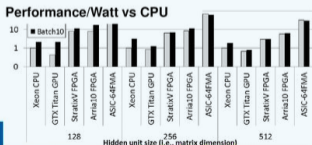
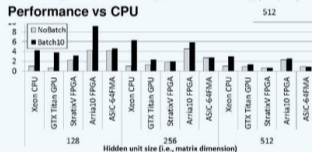
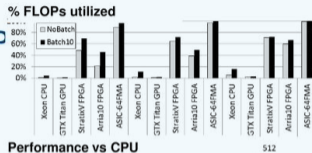
What's the interest?

FPGA vs. ASIC vs. GPU vs. CPU



FPGA ~10x better in perf/watt vs CPU/GPU

FPGA ~7x worse in perf/watt vs ASIC



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

What's the interest?

Optimize server side AI

- ▶ Energy
Minimize TCO for AI workloads
Greener AI for social acceptance
- ▶ Throughput
Enhance job throughput at constant energy budget

Local computation possible!

- ▶ Energy
No router, cloud server, ...
⇒ Huge constraint in Edge Computing
⇒ Worse in IoT
⇒ Transmitting data costs energy
- ▶ Latency
Immediate response, no dead zone, no network reliability issue, ...
- ▶ Privacy/security
No storage in someone else's servers
Neither wire nor wireless sniffing possible

Hardware Accelerated Neural Network

What are the constraints?

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

Ad-hoc hardware means:

- ▶ ANN HW/SW partitioning
- ▶ Clever SW scheduling
- ▶ Clever SW data handling

Burden on SW implementation:

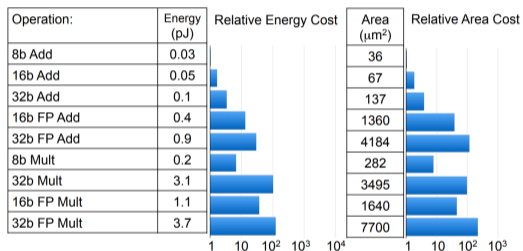
- ▶ Compilation Frameworks needed!
- ▶ No "one-size fits all" network

Computation Demanding

Inference involves a lot of computations, ...

- ▶ High number of floating point (FP) operations
 $0.5G \leq \text{Nb of FLOPs} \leq 40G$
- ▶ Floating point operations are energy and area costly

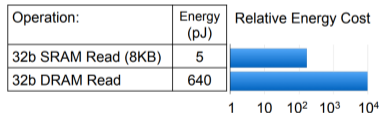
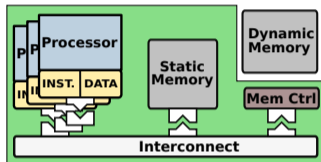
(My 4 core-i7 PC ~ 120 GFLOPs $\Rightarrow 30$ GFLOPs/core)



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

Memory demanding

Inference involves a lot of memory accesses, ...



"Hardware Architectures for DNN", ISCA Tutorial, 2017

- ▶ Memory stores millions of (64 or 32-bit) weights
⇒ 4M (GoogLeNet), 60M (AlexNet), 130M (VGG)
- ▶ Memory access becomes the bottleneck
⇒ Each op needs 2 operands and produces a result
- ▶ An "elevated" power consumption is involved

Coping with GFLOPs and GBytes

Alternatives: trade FLOPs for (some) accuracy loss

Simplify the operations

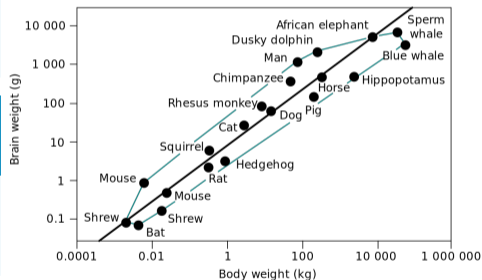
- ▶ Avoid sigmoid, tanh, sqrt and stuff
- ▶ FP arithmetic is not really HW friendly

Alternatives: trade bytes for (some) accuracy loss

- ▶ Use “small” data types, not 32/64-bit floats or ints

Alternatives: re-architect the “system”

- ▶ Integrate many memory cuts with processing elements and use them wisely



Coping with GFLOPs and GBytes

Alternatives: trade FLOPs for (some) accuracy loss

Simplify the operations

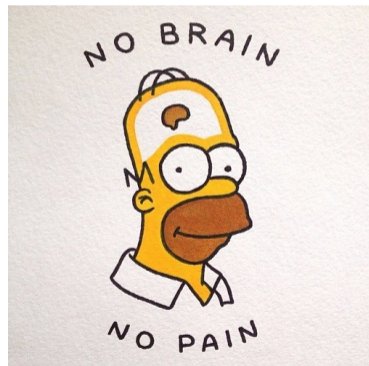
- ▶ Avoid sigmoid, tanh, sqrt and stuff
- ▶ FP arithmetic is not really HW friendly

Alternatives: trade bytes for (some) accuracy loss

- ▶ Use “small” data types, not 32/64-bit floats or ints

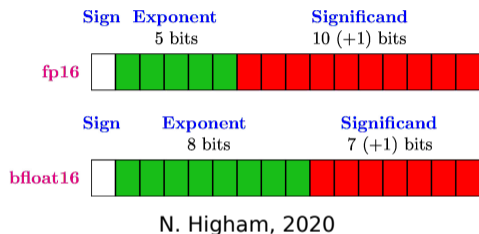
Alternatives: re-architect the “system”

- ▶ Integrate many memory cuts with processing elements and use them wisely



Using small floats

Introduction of a new floating point representation (mainly needed for *training*):
Google bfloat16 (“b” for brain)



Used for both weights and activations

- ▶ Large dynamic range, still small differences close to zero
- ▶ Reduction of multiplier power and footprint
- ▶ Optimized storage and bandwidth

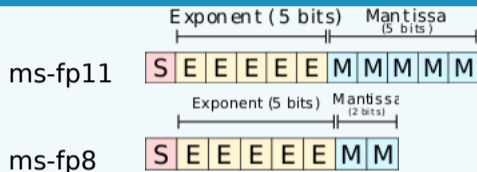
Quickly adopted and implemented in HW and SW

- ▶ Google TPU v2/v3, TensorFlow
- ▶ Intel Nervana, Intel Quartus FPGAs
- ▶ CPUs: Intel Xeon (AVX-512), ARMv8.6-A, IBM Power10
Supported from gcc 10.1 on

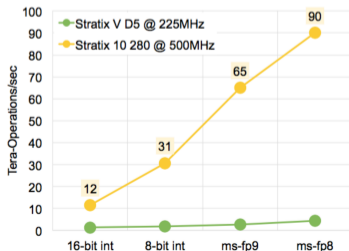
Using smaller floats!

Microsoft BrainWave project:

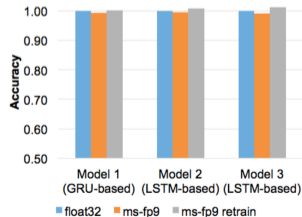
ms-fp Microsoft Floating-Points



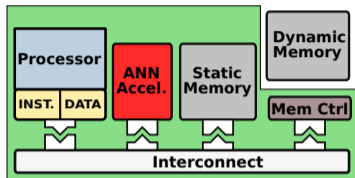
FPGA Performance vs. Data Type



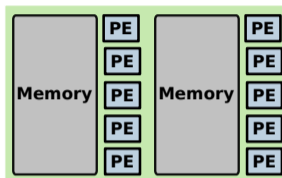
Impact of Narrow Precision on Accuracy



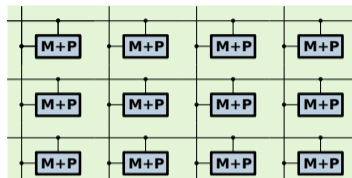
Typical Architectures for HW ANN



Exploit weight sparsity to optimize memory usage and weight placement

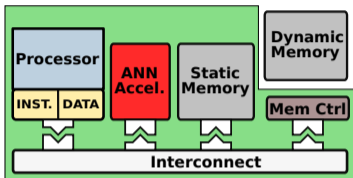


Use low precision/high efficiency computation along with on-chip memory storage of the weights

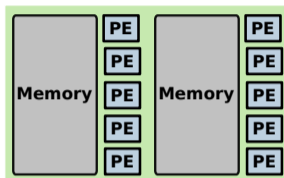


Integrate computation inside the memory itself, directly where the data is stored

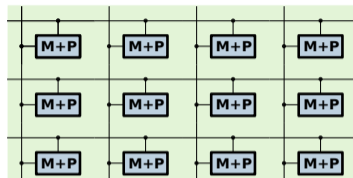
Typical Architectures for HW ANN



Exploit weight sparsity to optimize memory usage and weight placement



Use low precision/high efficiency computation along with on-chip memory storage of the weights



Integrate computation inside the memory itself, directly where the data is stored

Not within the scope of this presentation

Quantization

Quantization levels and accuracy...

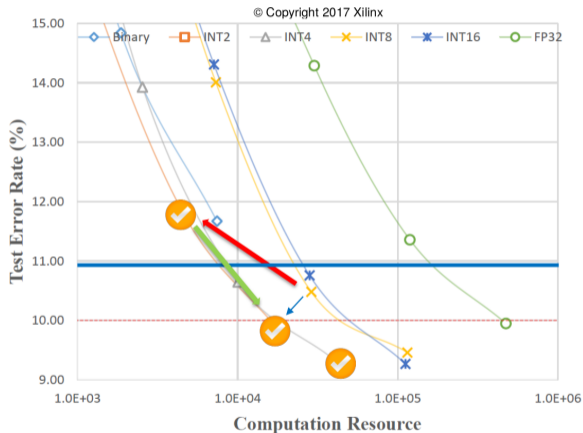
Just reducing precision,
reduce hardware cost &
increases error



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

Quantization aware training: a bit of literature

“Less bit per weights and activations”

- ▶ “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
- ▶ ...

Frameworks

- ▶ Tensorflow
tflite, qkeras
 - ▶ Pytorch
quantization API
 - ▶ Larq
binary only
 - ▶ ...
- ≥ 8 -bit \Rightarrow Post-Training
Quantization Ok
 < 8 -bit \Rightarrow Quantization
Aware Training required

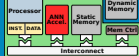
Post-training quantization example:

```
# https://www.tensorflow.org/lite/performance/post_training_integer_quant
def representative_data_gen():
    for input_value in tf.data.Dataset.from_tensor_slices(train_i).batch(1).take(100):
        yield [input_value]

converter = tf.lite.TFLiteConverter.from_keras_model(nn_model)
# Get float model for later (comparison, mainly)
tflite_model = converter.convert()
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
# Ensure that if any ops can't be quantized, the converter throws an error
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Set the input and output tensors to uint8 (APIs added in r2.3)
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
tflite_model_quant = converter.convert()

tf.lite.Interpreter(model_content=tflite_model_quant)
```

Exploit Sparsity and Quantization



Custom hardware for sparse matrix-vector multiplication

Deep Compression Technique

Reduces storage requirements

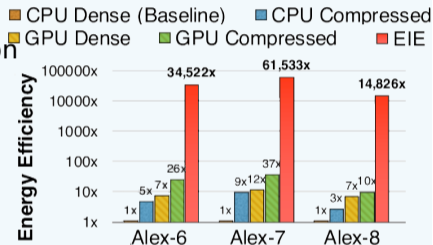
- ▶ Dedicated sparse matrix/vector representation
⇒ Eliminates redundant connections
- ▶ Quantizes weights down to 5 bits

Quantization of AlexNet weights

- ▶ 256 shared weights (Conv layers) ⇒ 4 bits
- ▶ 35x of reduction (240MB ⇒ 6.9MB)

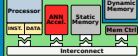
Weights stored into on-chip SRAM

⇒ 5 pJ/access (vs. 640 pJ/access off-chip DRAM)



Power efficiency

600 mW for Alexnet Fully-Connected layers



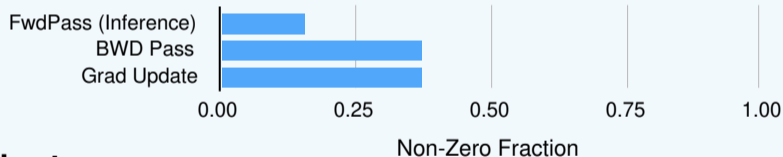
Acceleration using Low-Precision (ternary) weights

Only balanced **ternary weight** are used $\{-1, 0, +1\}$

- ▶ Floating point accumulations are kept
- ▶ Multipliers are not needed

Most of the FP operations operate on zero values

- ▶ Zero-skipping



Demonstrated highest accuracy

- ⇒ 93% on the ImageNet object classification challenge
- ⇒ Divide by 3 the number of FP operations

YodaNN: VLSI Implementation of binary-weights CNN Accelerator

Based on BinaryConnect [Courbariaux, NeurIPS 2015]

- ▶ Binary weights $\in \{-1, +1\}$
- ▶ 2's complement and multiplexers instead of multipliers
- ▶ Still full fledge adders: 12-bit activations

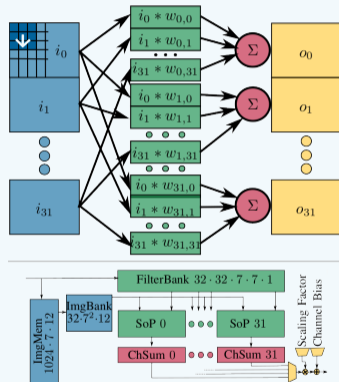
Large on-chip weights storage thanks to their size

- ▶ Latch-based standard cell memory

Flexible accelerator

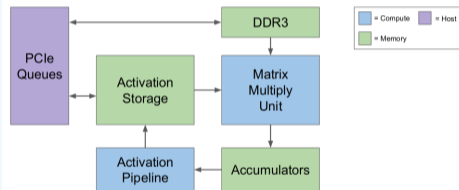
- ▶ 7 kernel sizes supported

⇒ 61.2 TOP/s/W at 0.6V



Google Edge Tensor Processor Unit

TPUv1 Recap

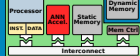


	V1	V2	V3
Clock Frequency (MHz)	800	1066	1066
# of (X, Y)-PEs	(4, 4)	(4, 4)	(4, 1)
PE Memory	2 MB	384 KB	2 MB
# of Cores per PE	4	1	8
Core Memory	32 KB	32 KB	8 KB
# of Compute Lanes	64	64	32
Instruction Memory	16384	16384	16384
Parameter Memory	16384	8192	8192
Activation Memory	1024	1024	1024
I/O Bandwidth (GB/s)	17	32	32
Peak TOPS	26.2	8.73	8.73

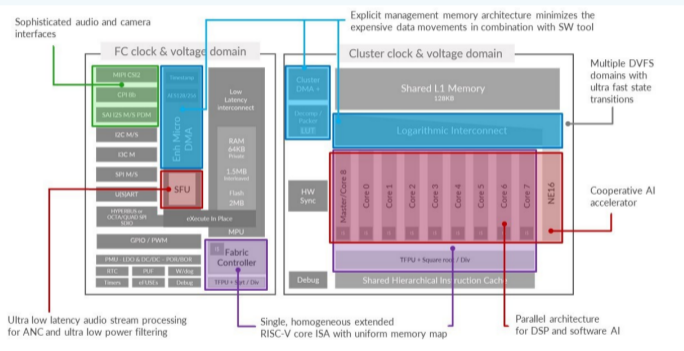
Comes for free in Tensorflow lite

Yazdanbakhsh et al., "An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks", Google, 2021

8, 4, 2-bit computations



Greenwaves GAP9

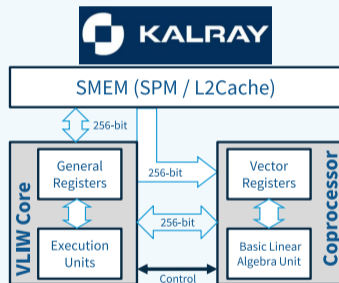


- ▶ 330 μ W/GOp
- ▶ Up to 15.6 GOPs and 32.2 GMACs
- ▶ 8, 4, 2-bit SIMD computations
- ▶ Support from ML frameworks
- ▶ (RISC-V based)

<https://greenwaves-technologies.com/>, 2021

Kalray MPPA3 Tensor Coprocessor

- ▶ Extend VLIW core ISA with extra issue lanes
 - ⇒ Separate 48x 256-bit wide vector register file
 - ⇒ Matrix-oriented arithmetic operations
- ▶ Full integration into core instruction pipeline
 - ⇒ Move instructions supporting matrix-transpose
 - ⇒ Proper dependency / cancel management
- ▶ Leverage MPPA memory hierarchy
 - ⇒ SMEM directly accessible from coprocessor
 - ⇒ Memory load stream alignment operations
- ▶ Arithmetic performances
 - ⇒ 128x INT8→INT32 MAC/cycle
 - ⇒ 64x INT16→INT64 MAC/cycle
 - ⇒ 16x FP16→FP32 FMA/cycle



- ▶ Integration within learning frameworks?

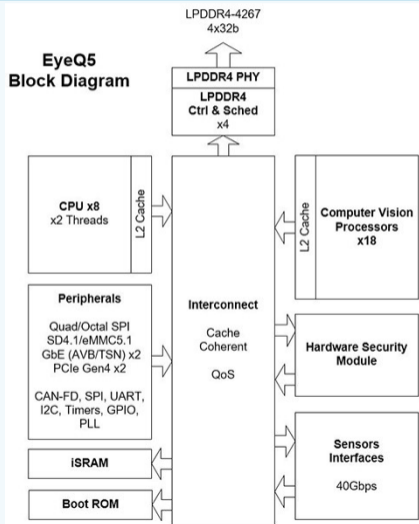
Mobileye EyeQ5

- ▶ Vector Microcode Processors:
CV dedicated VLIW SIMD engines
- ▶ Multithreaded Processing Cluster:
in between CPU and GPU
- ▶ Programmable Macro Array:
probably some sort of CGRA
- ▶ Full cache coherency!

10+ Watt, 24 Tops

Very ad-hoc programming approach (AFAIU)

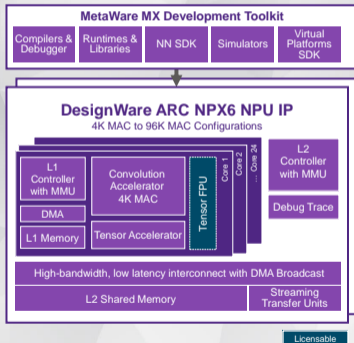
www.mobileye.com/our-technology/evolution-eyeq-chip/



Synopsys ARC NPX6

new

DesignWare ARC NPX6 w/ 440 TOPS* Performance



- **Scalable NPX6 architecture**

- 1 to 24 core NPU up to 96K MACS (440 TOPS*)
- Multi-NPU support (up to eight for 3500 TOPS*)

- Trusted **software tools** scale with the architecture

- **Convolution accelerator** – MAC utilization improvements with emphasis on modern network structures, including Transformers

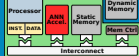
- **Generic Tensor accelerator** – Flexible Activation & support of Tensor Operator Set Architecture (TOSA)

- **Memory Hierarchy** – high bandwidth L1 and L2 memories

- **DMA broadcast lowers external memory bandwidth requirements and improves latency**

* 1.3 GHz, 5nm FFC worst case conditions using sparse EDSR model

30 TOPs/W in 5 nm



HW accelerated AI for less than \$100

- ▶ Google Coral:
byte based matrix \times matrix TPU,
2 Watt, 4 TOPs
- ▶ NVIDIA Jetson Nano:
float and int GPU (128-cores),
10 Watt, 472 GFlops
- ▶ Intel Neural Computing Stick 2:
float VPU (128-bit VLIW vector (?) procs),
2 Watt, 4 TOPs

Software support out-of-the-box
by major "generic" frameworks

- ▶ Tensorflow[lite]
- ▶ Pytorch

Or ad-hoc ones

- ▶ OpenVino
- ▶ ...

FINN: Framework for building FPGA[†] accelerators

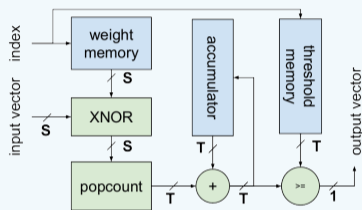
Mapping binarized neural networks to hardware All values $\in \{-1, +1\}$

- ▶ Binary input activation
- ▶ Binary synapse weights
- ▶ Binary output activation

Weights kept in on-chip memory

- ⇒ Zynq-7000 FPGA technology
- ⇒ 80.1% accuracy for CIFAR-10
- ⇒ Total system power 25W

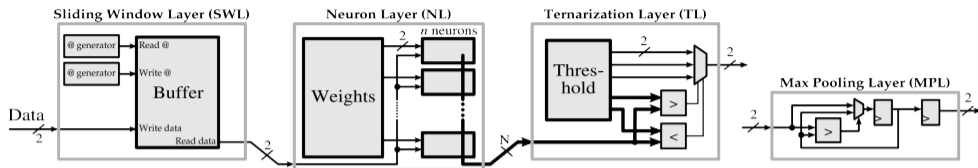
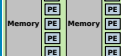
Convolution layer



- ▶ Dot-product between input vector and row of synaptic weight matrix
- ▶ Compares result to a threshold
- ▶ Produces single-bit output

[†]Field-Programmable Gate-Array: fine-grain reconfigurable hardware technology.

Ternary weights *and* ternary activations



FPGA Architecture for Ternary Neural Networks (TNN)

- ▶ Large-scale ternary CNN pipeline, VGG-like
- ▶ Neuron layer → memory (ternary weights) + neurons
- ▶ Ternarization layer → ternary activations $\in \{-1, 0, +1\}$

⇒ Error rate 13.29% for CIFAR-10 (vs. 19.9% in FINN)

⇒ Virtex-7 FPGA technology (VC709, Laaaaaaarge FPGA)

⇒ Peak power 12 W, 1.62 TOP/s/W (vs 0.69 TOP/s/W in FINN)

⇒ Throughput > 60k fps

A. Prost-Boucle et al., "High-Efficiency Convolutional Ternary Neural Networks with Custom Adder Trees and Weight Compression", ACM TRET, 2018

Sept 2021 Overview

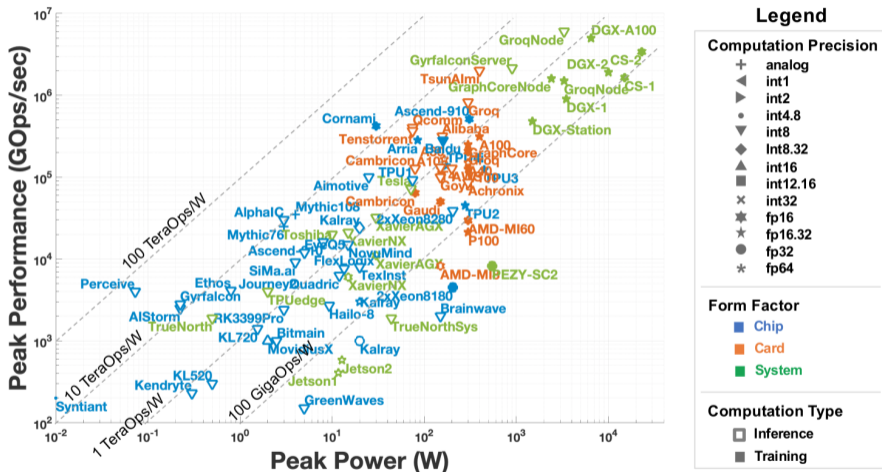


Fig. 2. Peak performance vs. power scatter plot of publicly announced AI accelerators and processors.

A. Reuther et al., "AI Accelerator Survey and Trends", arXiv, 2021

End-to-end Flows

From Python to boosted HW/SW execution

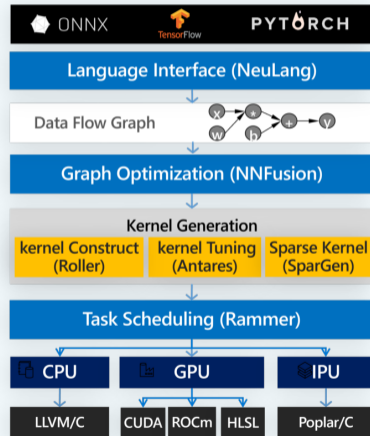
Motto: "Machine learning compiler frameworks for CPUs, GPUs, and machine learning accelerators aim to enable machine learning engineers to optimize and run computations efficiently on any hardware backend"

Exchange format: ONNX

Frameworks:

- ▶ Apache TVM
- ▶ Apple Core ML
- ▶ Facebook Glow
- ▶ Google XLA
- ▶ Microsoft DL Compiler and Optimizer

Operational today, but backends not simple to get into!



Recent (bad) news

Mobile ViT (Apple, March 2022), <https://arxiv.org/pdf/2110.02178.pdf>

Model	# Params ↓	FLOPs ↓	Top-1 ↑	Inference Time (ms)		
				iPhone12 - CPU	iPhone12 - Neural Engine	
MobileNetv2	3.5 M	0.3 G	73.3	7.50 ms	0.92 ms	→ CPU/NNE = 8.1X
DeiT	5.7 M	1.3 G	72.2	28.15 ms	10.99 ms	
PiT	4.9 M	0.7 G	73.0	24.03 ms	10.56 ms	
MobileViT (Ours)	2.3 M	0.7 G	74.8	17.86 ms	7.28 ms	→ CPU/NNE = 2.5X
	0.7X Model Size	2.3X FLOPs	+1.5% Accuracy	2.4X Time	7.9X Time	

(Courtesy of Pierre Paulin, Synopsys)

Apple AI NPU not well suited to support Transformers, ...
As all others, I'd say!

Classical digital (CMOS) architectures are here to stay

Design of more efficient HW for inference and training

- ▶ Ad-hoc circuits necessary for high-performance, low energy solutions
- ▶ Quantization
 - ⇒ Simpler arithmetic circuits
 - ⇒ Lower memory requirements
 - ⇒ Lower bandwidth requirements

But HW guys cannot do that alone!

- ▶ New quantization aware training needed
- ▶ Better understanding of learning needs
- ▶ HW accelerated low bit-width learning
- ▶ Frameworks to facilitate HW accelerator usage

