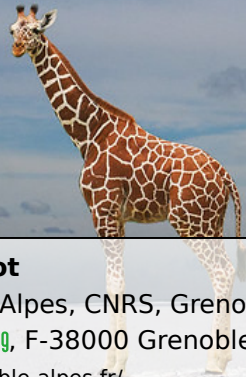
Hardware Acceleration for Neural Networks



Frédéric Pétrot

Univ. Grenoble Alpes, CNRS, Grenoble INP*,
TIMA/INP Ensimag, F-38000 Grenoble, France

tima.univ-grenoble-alpes.fr/
research/sls/members/frederic-petrot

frederic.petrot@univ-grenoble-alpes.fr

*Institute of Engineering Univ. Grenoble Alpes

MIAI Grenoble Alpes
Multidisciplinary Institute In Artificial Intelligence

EDGE AI

GRENOBLE INP UGA

cnrs

INVESTIR L'AVENIR

Co-funded by the European Union

Breaking news: Computers actually need energy!

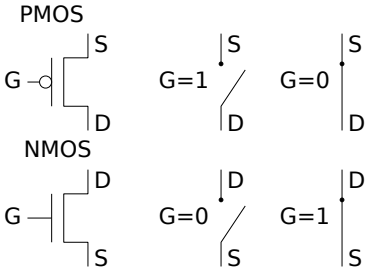
- Power consumption primer
- Orders of magnitude

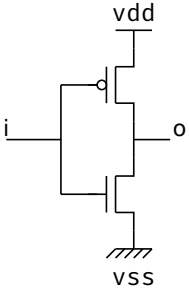
Hardware Accelerated AI^H^H ML^H^H **NN**

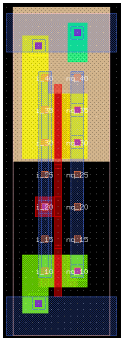
- DNN as seen by HW guys
- What is HW by the way and why use it?
- Ways to ”enhance” NN computations

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: CMOS transistor used as a switch

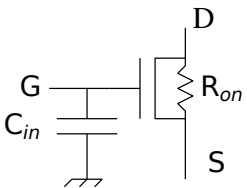






Switch Electrical Behavior

CMOS transistor ≠ 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 energy consuming 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: $P = \alpha C V_{dd}^2 f + I_{leak} V_{dd}$

Complementary Metal Oxyde Semiconductor

Why CMOS?

- Very small dimensions

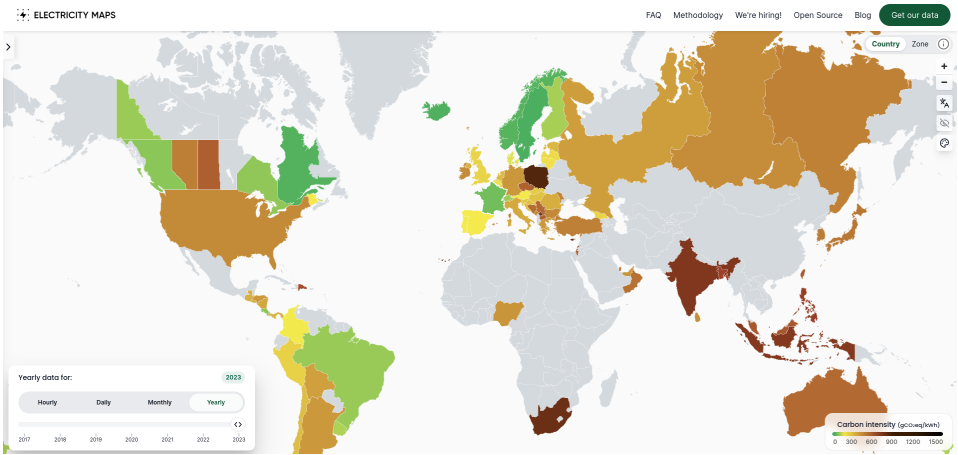
  Mass production smallest transistor width in 2024/2025 : 3 nm
  - TSMC – Apple A14/A15, Huawei Mate 40, HiSilicon Kirin 9000 –: 173 MTr/mm²
    Apple M1 (16 Mrds Trs): 134 MTrs/mm² actual transistor density
  - Samsung – Exynos, Snapdragon 8xx, Nvidia Hopper –: 127 MTr/mm²
  - Intel – 123.4 MTr/mm²

  Si atomic radius is ≈ 0.11 nm !

  2 nm production expected for 2025, 1 nm for 2027
- 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

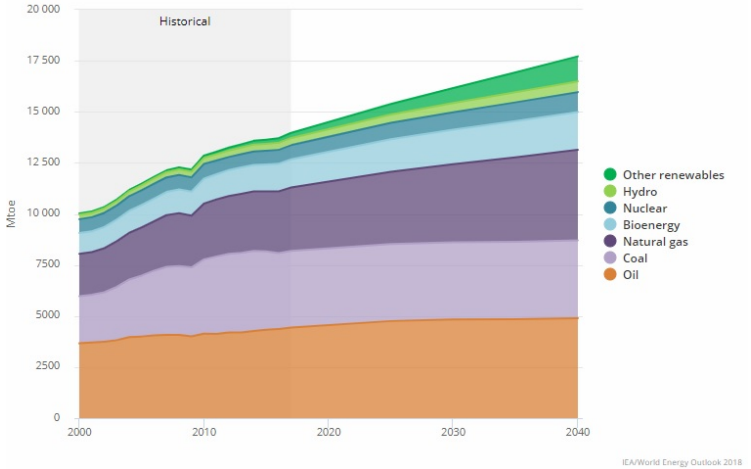
⇒ CMOS: hyper-hegemonic digital technology

Energy production



https://app.electricitymaps.com/map

Energy production

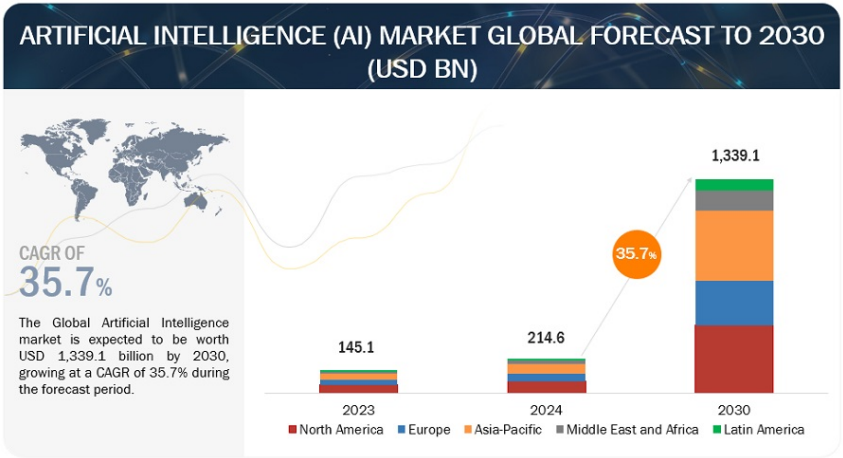


in MegaTonne of Oil Equivalent

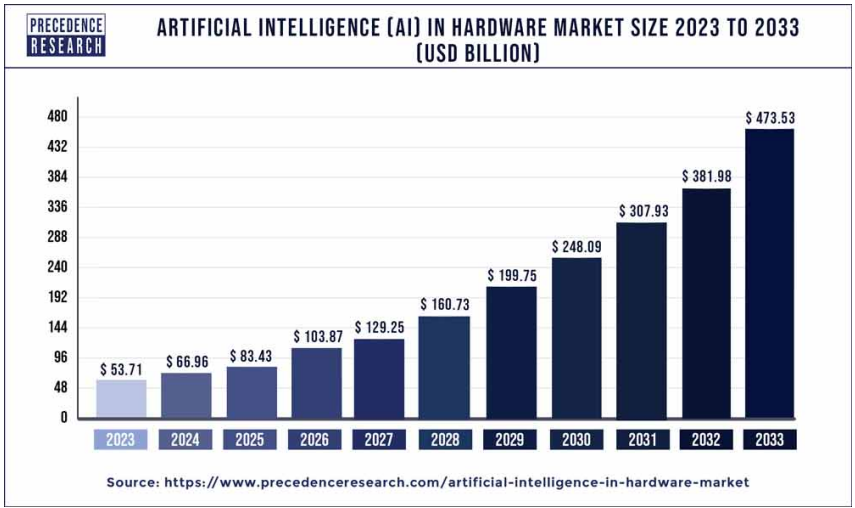
U.S. Energy Information Admisistration

`https://www.eia.gov/outlooks/ieo/pdf/0484(2017).pdf`

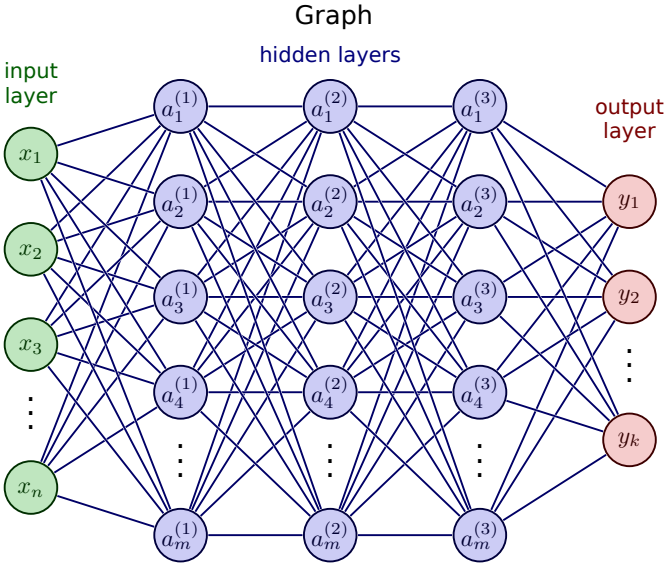
AI Market Forecast



AI Market Forecast

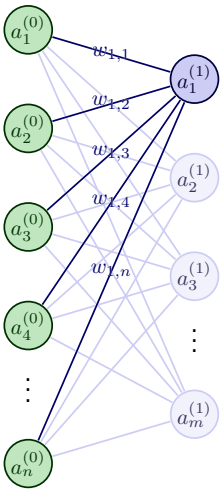


Neural Network 101



Neural Network 101, cont.

Math



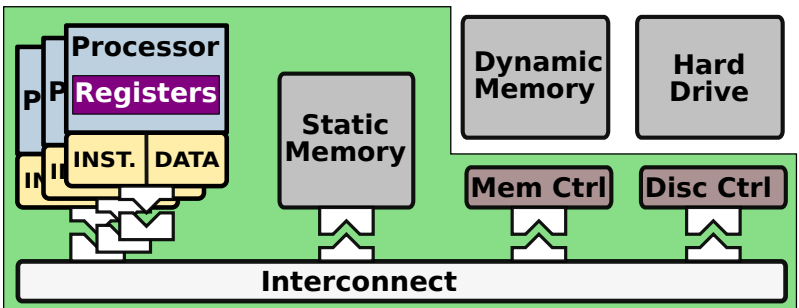
$$= \sigma\left(w_{1,0}a_0^{(0)} + w_{1,1}a_1^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} & w_{1,1} & \ldots & w_{1,n} \\ w_{2,0} & w_{2,1} & \ldots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \ldots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$a^{(1)} = \sigma\left(\mathbf{W}^{(0)}a^{(0)} + \mathbf{b}^{(0)}\right)$$

Overview


Processor
Registers
INST.
DATA
Static Memory
Dynamic Memory
Hard Drive
Mem Ctrl
Disc Ctrl
Interconnect

Computer System 101, cont.

Storage order of magnitude for size, access time, and cost

Registers

	1980		2023	2023 vs 1980
T_{acc} (ns)	300		0.20	÷1500
Typ. size (B)	64		512	×8

Static memory

	1980		2023	2023 vs 1980
$/MB	19,200		5	÷3840
T_{acc} (ns)	300		1	÷300
Typ. size (KB)	32		8192	×256

Dynamic memory

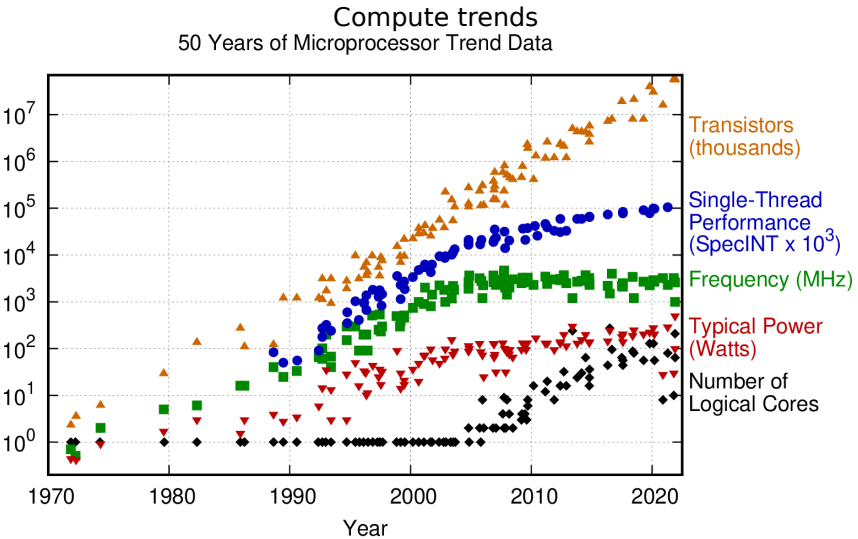
	1980		2023	2023 vs 1980
$/MB	8,800		0.003	÷2,930,000
T_{acc} (ns)	375		30	÷12.5
Typ. size (MB)	0.064		32,000	×500,000

Hard drive

	1980		2023	2023 vs 1980
$/MB	500		.000018	÷27,800,000
T_{acc} (ms)	87		12	÷7
Typ. size (GB)	0.001		8,000	×1,500,000

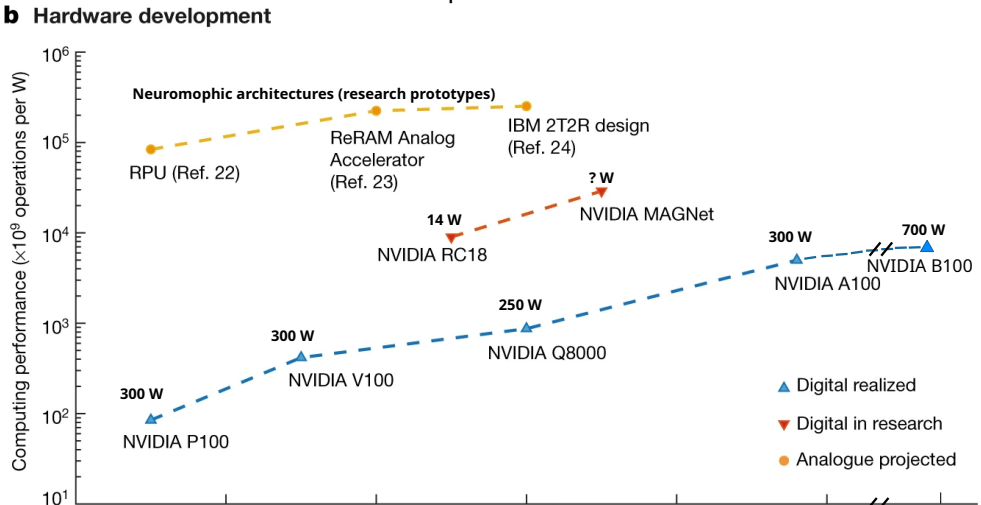
Sources : John C. McCallum (`https://jcmit.net/`) and various (D/S)RAM vendors web sites

Computer System 101, cont.



Computer System 101, cont.

Compute trends



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

Computer System 101 + Neural Networks 101 ⇒

Quick Summary

Raw Power Consumption:

- CPUs: 100/150/300 Watt
- GPUs: 250/300/700 Watt

Training Example

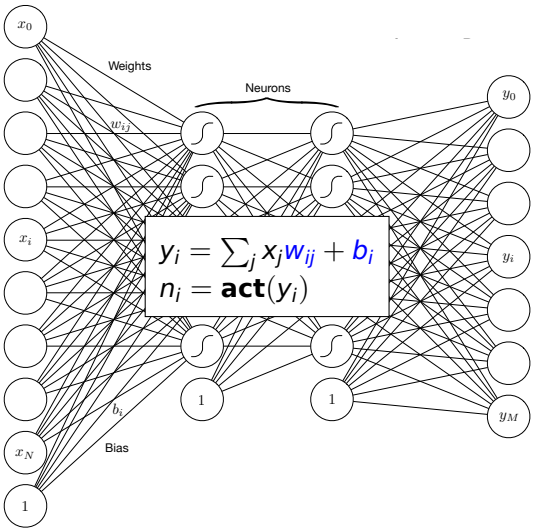
NVidia MegatronLM

- Used 45 Tera Bytes of data
- On 512 V100 NVIDIA GPUs during 9 days
- $512 \times 300 \times 9 \times 24 = 33177$ kWh
  $7\times$ 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

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



Two phases :
Training and Inference

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

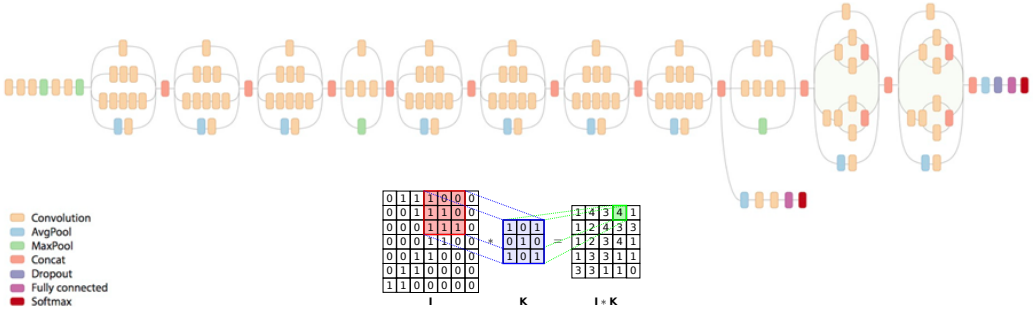
NaÃ¨ve FC in C

```
void dense(int inputs,
           int outputs,
           float input[inputs],
           float weight[outputs][inputs],
           float bias[outputs],
           float output[outputs])
{
    for (int j = 0; j < outputs; j ++) {
        for (int i = 0; i < inputs; i ++) {
            output[j] += input[i] * weight[j][i];
        }
        output[j] = activation(output[j] + bias[j]);
    }
}
```

Boils down to matrix-vector multiplication

- Input data can be large: small 32x32 pixels images ⇒ 1024 NN inputs

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, fork and merge of connections
- Training possible thanks to GPU based compute farms

NaÃ¯ve conv2D software implementation in C

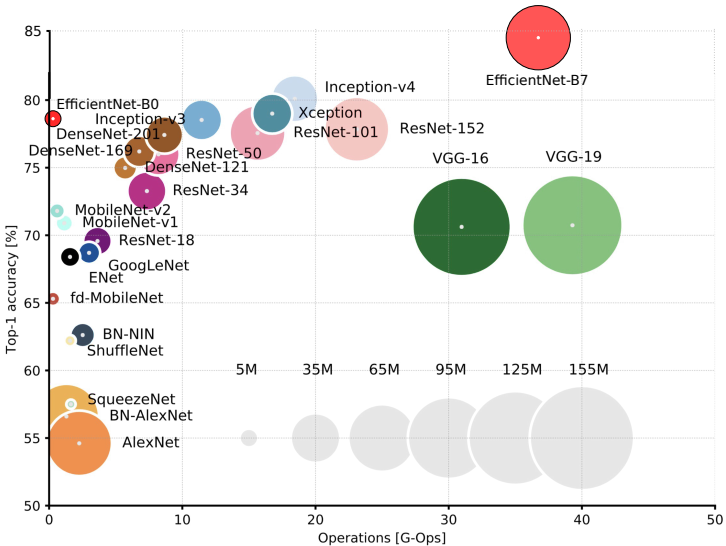
```
void conv2d(int in_channels, int out_channels, int img_size, int kernel_size,
            float input[img_size][img_size][in_channels],
            float kernel[out_channels][kernel_size][kernel_size][in_channels],
            float bias[out_channels],
            float output[img_size - 2 * (kernel_size / 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 o = 0; o < out_channels; o++)
        for (int k = 0; k < fm_size; k++)
            for (int l = 0; l < fm_size; l++) {
                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] = relu(mac + bias[o]);
            }
}
```

Conv2D software implementation compiled with `gcc -O3`

```
conv2d:
    pushq   %r15
    movl    %esi, %eax
    movl    %edx, %esi
    movl    %ecx, %r15d
    pushq   %r14
    movslq  %edi, %rdx
    pushq   %r13
    leaq    0(,%rdx,4), %r10
    pushq   %r12
    pushq   %rbp
    pushq   %rbx
    subq    $32, %rsp
    movl    %ecx, -96(%rsp)
    movslq  %esi, %rcx
    imulq   %rdx, %rcx
    movq    %r8, -104(%rsp)
    movq    %r10, -112(%rsp)
    movq    %rcx, -64(%rsp)
    movslq  %r15d, %rcx
    imulq   %rcx, %rdx
    movq    %rcx, %rbx
    movl    %r15d, %ecx
    shrl    $31, %ecx
    addl    %r15d, %ecx
    andl    $-2, %ecx
    imulq   %rdx, %rbx
    subl    %ecx, %esi
    movl    %r15d, %ecx
    notl    %ecx
    andl    $1, %ecx
    leal    (%rsi,%rcx), %r8d
    testl   %eax, %eax
    jle     .L4
    movq    %r9, %r11
    movslq  %eax, %r9
    movl    %edi, %r13d
    shrq    $2, %r10
    leaq    0(,%rdx,4), %rdi
    leaq    0(,%r9,4), %rdx
    movq    %r10, %rbp
    movq    %rdx, %rax
    shrq    $2, %rdi
    shrq    $2, %rax
    movq    %rdi, %rsi
    movq    %rax, -56(%rsp)
    testl   %r8d, %r8d
    jle     .L4
    movslq  %r8d, %rcx
    leal    -1(%r13), %eax
    movl    %r13d, %r14d
    xorl    %edi, %edi
    imulq   %rcx, %rdx
    movl    %eax, -80(%rsp)
    movl    %r13d, %eax
    andl    $-4, %r14d
    shrl    $2, %eax
    xorl    %r15d, %r15d
    pxor    %xmm3, %xmm3
    salq    $4, %rax
    movq    %rax, %r12
    movq    %rdx, %rcx
    movq    %rsi, %rax
    movq    %rdi, %rdx
    movq    %r9, %rdi
    movq    %rbx, %r9
    movl    %r14d, %ebx
.L7:
    movq    96(%rsp), %r14
    movq    %rdi, -32(%rsp)
    leaq    (%r11,%rdx,4), %r10
    movslq  %r15d, %rsi
    movq    %rsi, -48(%rsp)
    xorl    %esi, %esi
    movq    %r9, -40(%rsp)
    movq    %rdx, %r9
    movq    %r14, %rdx
    movq    %r10, %r14
.L22:
    movl    $0, -92(%rsp)
    movq    %rax, %rdi
    movl    %esi, -8(%rsp)
    movq    %rcx, -24(%rsp)
    movq    %r15, %rcx
.L21:
    movl    -96(%rsp), %eax
    testl   %eax, %eax
    jle     .L24
    movq    %rcx, -16(%rsp)
    xorl    %esi, %esi
    xorl    %eax, %eax
    pxor    %xmm1, %xmm1
    movl    %r8d, -4(%rsp)
    movq    %r9, (%rsp)
    movq    %rdx, 8(%rsp)
    movq    %r11, 16(%rsp)
.L19:
    movl    -8(%rsp), %ecx
    movslq  %eax, %r9
    movq    %rdi, -88(%rsp)
    xorl    %r8d, %r8d
    imulq   %rdi, %r9
    movl    %eax, -76(%rsp)
    leal    (%rax,%rcx), %r11d
    movq    -64(%rsp), %rcx
    movq    %rsi, -72(%rsp)
    movslq  %r11d, %r11
    imulq   %rcx, %r11
    movq    -104(%rsp), %rcx
    leaq    (%rcx,%r11,4), %r10
    leaq    (%r14,%rsi,4), %rcx
.L15:
    testl   %r13d, %r13d
    jle     .L13
    movl    -92(%rsp), %eax
    cmpl    $2, -80(%rsp)
    leal    (%rax,%r8), %edx
    jbe     .L23
    movslq  %edx, %rdx
    movq    -104(%rsp), %rdi
    movq    %rdx, %rax
    imulq   %rbp, %rax
    addq    %r11, %rax
    leaq    (%rdi,%rax,4), %rsi
    xorl    %eax, %eax
.L9:
    movups  (%rsi,%rax), %xmm0
    movups  (%rcx,%rax), %xmm4
    addq    $16, %rax
    mulps   %xmm4, %xmm0
    addss   %xmm0, %xmm1
    movaps  %xmm0, %xmm2
    shufps  $85, %xmm0, %xmm2
    addss   %xmm2, %xmm1
    movaps  %xmm0, %xmm2
    unpckhps %xmm0, %xmm2
    shufps  $255, %xmm0, %xmm0
    addss   %xmm1, %xmm2
    movaps  %xmm0, %xmm1
    addss   %xmm2, %xmm1
    cmpq    %rax, %r12
    jne     .L9
    movl    %ebx, %eax
    cmpl    %ebx, %r13d
    je      .L13
.L8:
    movslq  %r8d, %rdi
    imulq   %rbp, %rdx
    movslq  %eax, %rsi
    imulq   %rbp, %rdi
    leaq    (%rsi,%r9), %r15
    addq    %rdx, %rsi
    addq    %rdi, %r15
    movss   (%r14,%r15,4), %xmm0
    mulss   (%r10,%rsi,4), %xmm0
    leal    1(%rax), %esi
    addss   %xmm0, %xmm1
    cmpl    %esi, %r13d
    jle     .L13
    movslq  %esi, %rsi
    addl    $2, %eax
    leaq    (%rsi,%r9), %r15
    addq    %rdx, %rsi
    addq    %rdi, %r15
    movss   (%r14,%r15,4), %xmm0
    mulss   (%r10,%rsi,4), %xmm0
    addss   %xmm0, %xmm1
    cmpl    %eax, %r13d
    jle     .L13
    cltq
    addq    %rax, %rdx
    addq    %r9, %rax
    addq    %rdi, %rax
    movss   (%r10,%rdx,4), %xmm0
    mulss   (%r14,%rax,4), %xmm0
    addss   %xmm0, %xmm1
.L13:
    movq    -112(%rsp), %rdi
    leal    1(%r8), %eax
    addq    %rdi, %rcx
    cmpl    %eax, -96(%rsp)
    je      .L11
    movl    %eax, %r8d
    jmp     .L15
.L11:
    movq    -88(%rsp), %rdi
    movl    -76(%rsp), %eax
    movq    -72(%rsp), %rsi
    leal    1(%rax), %edx
    addq    %rdi, %rsi
    cmpl    %r8d, %eax
    je      .L28
    movl    %edx, %eax
    jmp     .L19
.L23:
    xorl    %eax, %eax
    movslq  %edx, %rdx
    jmp     .L8
.L28:
    movq    -16(%rsp), %rcx
    movl    -4(%rsp), %r8d
    movq    (%rsp), %r9
    movq    8(%rsp), %rdx
    movq    16(%rsp), %r11
.L14:
    movq    88(%rsp), %rax
    addss   (%rax,%rcx,4), %xmm1
    comiss  %xmm1, %xmm3
    jbe     .L16
    pxor    %xmm1, %xmm1
.L16:
    movslq  -92(%rsp), %rax
    movq    -56(%rsp), %r15
    movq    %rax, %rsi
    imulq   %r15, %rax
    movq    -48(%rsp), %r15
    addl    $1, %esi
    movl    %esi, -92(%rsp)
    addq    %r15, %rax
    movss   %xmm1, (%rdx,%rax,4)
    cmpl    %esi, %r8d
    jne     .L21
    movl    -8(%rsp), %esi
    movq    %rcx, %r15
    movq    -24(%rsp), %rcx
    movq    %rdi, %rax
    addl    $1, %esi
    addq    %rcx, %rdx
    cmpl    %esi, %r8d
    jne     .L22
    movq    %r9, %rdx
    movq    -32(%rsp), %rdi
    movq    -40(%rsp), %r9
    addq    $1, %r15
    addq    %r9, %rdx
    cmpq    %r15, %rdi
    jne     .L7
.L4:
    addq    $32, %rsp
    popq    %rbx
    popq    %rbp
    popq    %r12
    popq    %r13
    popq    %r14
    popq    %r15
    ret
.L24:
    pxor    %xmm1, %xmm1
    jmp     .L14
```

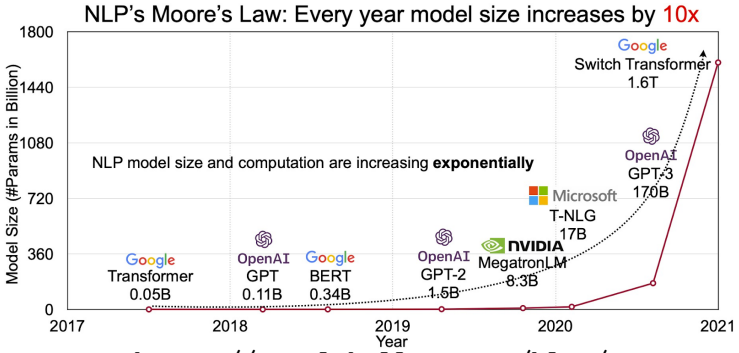
ANN Models: Accuracy, Operations and Parameters



A. Canziani, E. Culurciello, A. Paszke, *"An Analysis of Deep Neural Network Models for Practical Applications"*, 2018 (EfficientNet-B0/B7 added by myself)

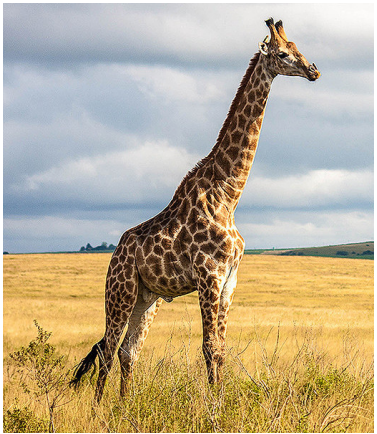
`https://culurciello.medium.com/analysis-of-deep-neural-networks-dcf398e71aae`

ANN Models: Accuracy, Operations and Parameters



https://www.labellerr.com/blog/an-introduction-to-large-language-models-llms

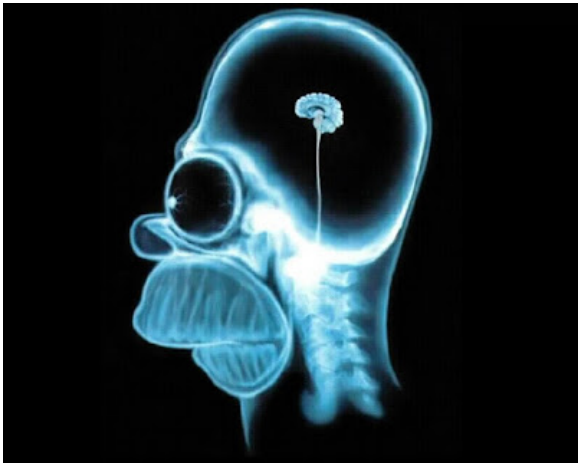
What AI do we really need?



D. PerÃ¨z, Flikr, 2015

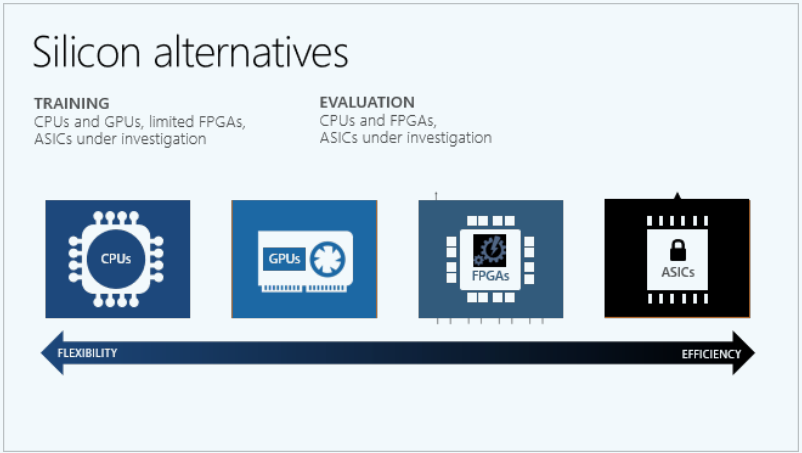
What AI do we really need?

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



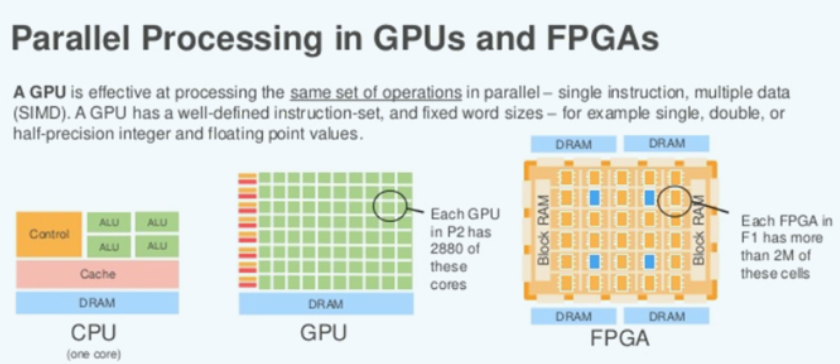
K. Usher, *"The Dwarf in the Dirt"*, Bones, 2009

What's the interest?



Microsoft Azure Machine Learning Documentation

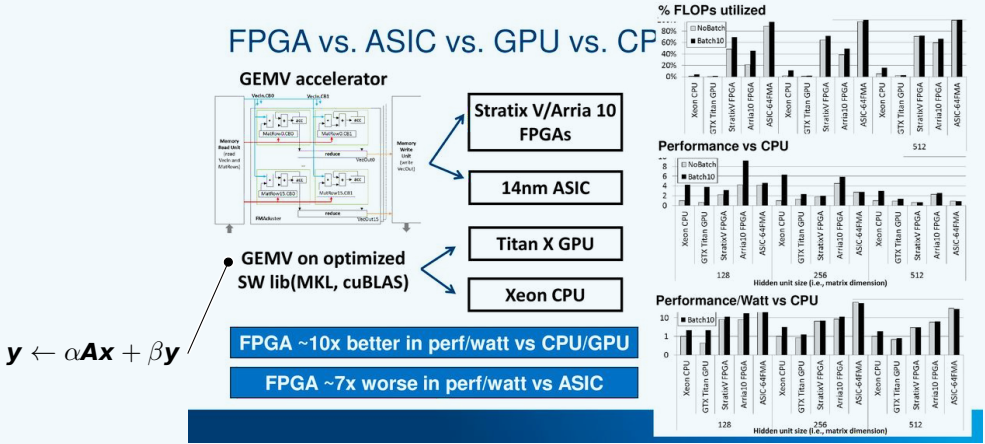
What's the interest?



Amazon Web Service Documentation

Hardware Accelerated Neural Network

What's the interest?



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

Hardware Accelerated Neural Network

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 µW 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 access

Burden on SW implementation:

- Compilation Frameworks needed!
- No ”one-size fits all” network

Computation Demanding

Inference involves a lot of computations, ...

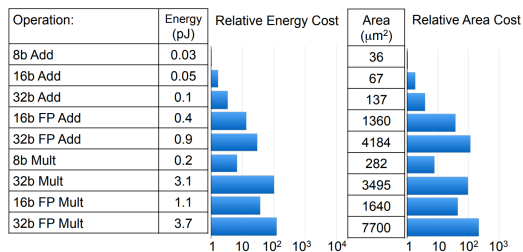
- High number of floating point (FP) operations

  0.5G ≤ 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

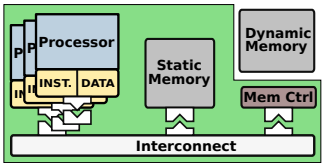


Area (μm^2)
36
67
137
1360
4184
282
3495
1640
7700

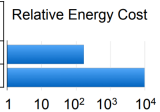
"Hardware Architectures for Deep Neural Networks", ISCA Tutorial, 2017

Memory demanding

Inference involves a lot of memory accesses, ...



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



"Hardware Architectures for DNN", ISCA Tutorial, 2017

- Mem holds millions of (64 or 32-bit) weights
  ⇒ 4M (GoogLeNet), 60M (AlexNet), 130M (VGG)
- Mem temporarily retains intermediate results
  ⇒ 1M to several M depending on network
- Mem access becomes the bottleneck
  ⇒ Each op needs 2 operands and produces a result
- High power consumption

Coping with GFLOPs and GBytes

Alternatives: trade FLOPs for (some) accuracy loss

Simplify the operations

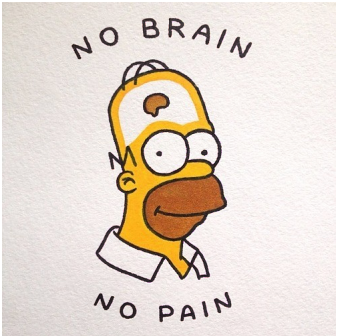
- Avoid sigmoid, tanh, sqrt and stuff
- FP arithmetic is not really HW friendly

Alternatives: trade bytes for (some) accuracy loss

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

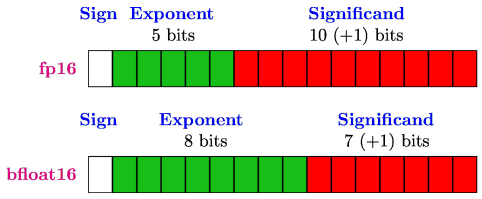
Alternatives: re-architect the “system”

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



Using small floats

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



N. Higham, 2020

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

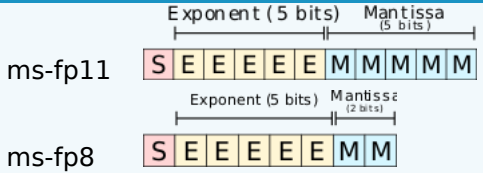
Quickly adopted and implemented in HW and SW

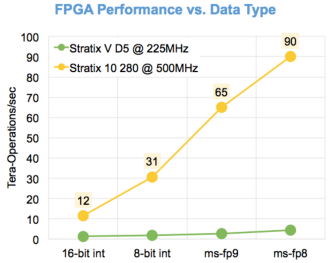
- Google TPU v2/v3, TensorFlow
- Intel Nervana, Intel Quartus FPGAs
- CPUs: Intel Xeon (AVX-512), ARMv8.6-A, IBM Power10
  Supported from `gcc` 10.1 and `llvm 9.0` on

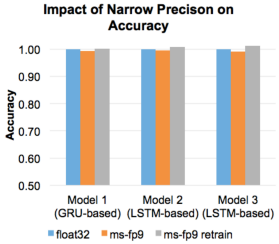
Using smaller floats!

Microsoft BrainWave project:

`ms-fp` Microsoft Floating-Points







Using even smaller floats!

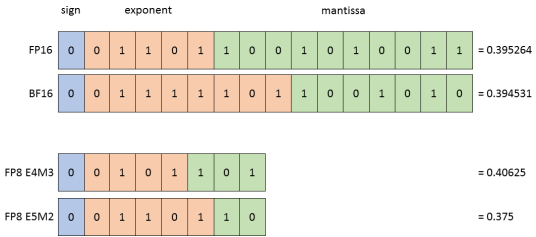
Sep 14, 2022

NVIDIA, Arm, and Intel have jointly authored a whitepaper, FP8 Formats for Deep Learning, describing an 8-bit floating point (FP8) specification. It provides a common format that accelerates AI development by optimizing memory usage and works for both AI training and inference. This FP8 specification has two variants, E5M2 and E4M3.

This format is natively implemented in the NVIDIA Hopper architecture and has shown excellent results in initial testing. It will immediately benefit from the work being done by the broader ecosystem, including the AI frameworks, in implementing it for developers.

FP8/FP4 Primer

FP8



https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html

- E4M3 :±448 and NaN
  More precision, used during forward pass
- E5M2 :±57344, ±Inf and NaN
  Higher dynamic range, used during backward pass
- Efficiency demonstrated on training Transformers!

FP4: 1 sign bit, 3 or 2 bits of exponent, 1 or 0 bits of mantissa

Using other fancy representations

Logarithmic Number System

Instead of encoding a real value a, encode its logarithm.
Why? $\log(a \times b) = \log a + \log b$!
However, back to linear for addition, ... But tabulatable for small size numbers
Ooups, looks a lot like FP4 with 0-bit mantissa ...

Balanced Ternary and Binary

$x, w \in \{-1, 0, 1\}$
$x, w \in \{-1, 1\}$
Where will this nonsense stop?

Integer Quantization

Quantization levels and accuracy...

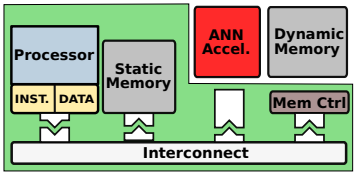
Just reducing precision, reduce hardware cost & increases error

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

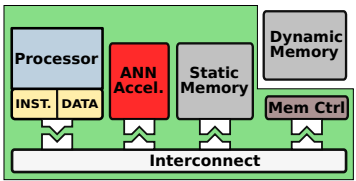
1b, 2b and 4b provide pareto optimal solutions

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

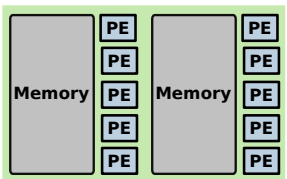
Typical Architectures for HW ANN



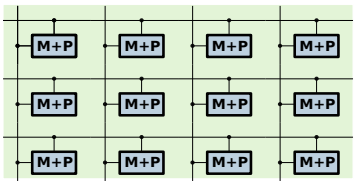
External HW accelerator for training and inference on laptop/servers



Exploit weight sparsity to optimize memory usage and weight placement



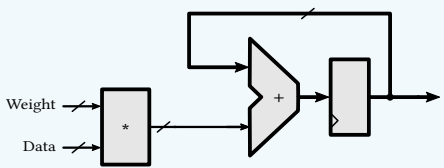
Use low precision/high efficiency computation along with on-chip memory storage of the weights



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

Neuron core design

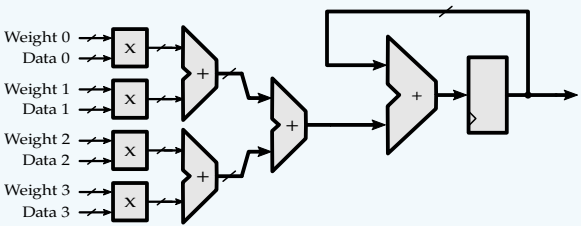
Sequential MAC computation



1 weight and 1 input per cycle per neuron

Neuron core design

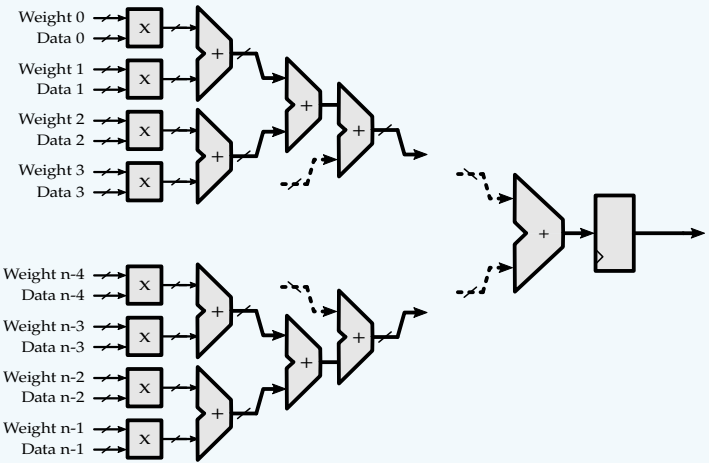
Partly parallel computation



p weights and p inputs per cycle per neuron ($p << n$)

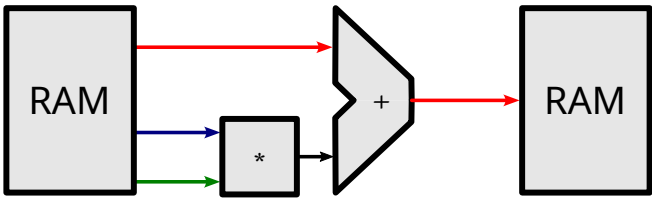
Neuron core design

Fully parallel computation

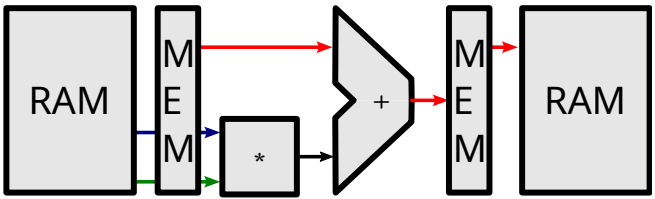


n weights and *n* inputs per cycle per neuron, not really practical!

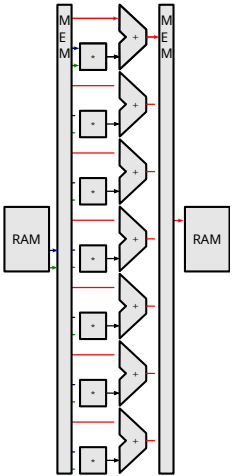
High-Level Architectures



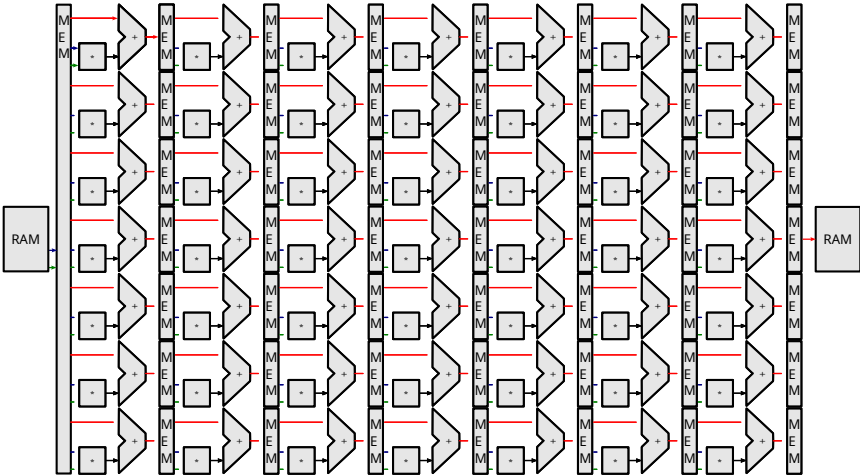
High-Level Architectures



High-Level Architectures



High-Level Architectures



Accelerator Zoo, Sept 2021 Overview

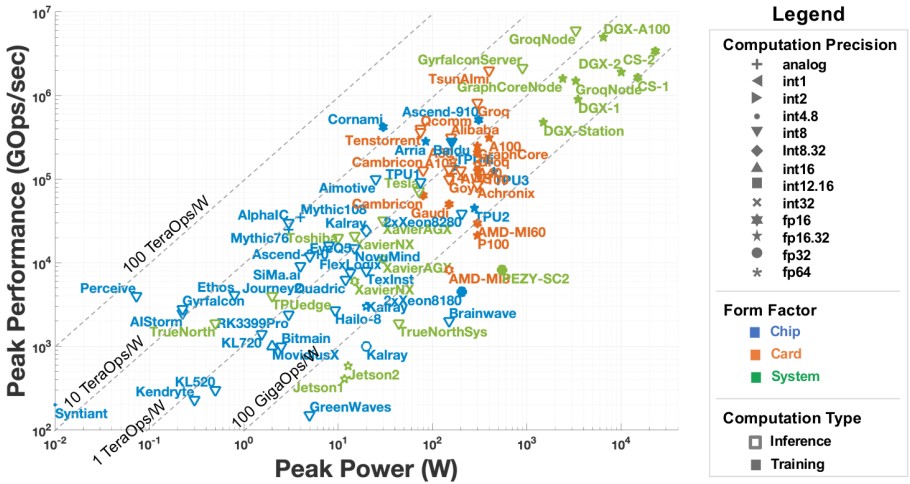


Fig. 2. Peak performance vs. power scatter plot of publicly announced AI accelerators and processors.
A. Reuther et al., *"AI Accelerator Survey and Trends"*, arXiv, 2021

General Purpose Computing on a GPU

Scalable AI Computations thanks to SIMT

NVIDIA Hopper Architecture In-Depth | NVIDIA Technical Blog developer.nvidia.com

General Purpose Computing on a GPU

Latest NVidia GPU: Blackwell

2 Chips on a chiplet, 104 Billions Tr/chip,
814 mm^2, 700 W, 4 nm TSCM, 10 To/s Chip to Chip
×2 in training performance compared to latest Hopper

GPU	B200	B100
FP4 Tensor Core	18 petaFLOPS	14 petaFLOPS
FP8/FP6 Tensor Core	9 petaFLOPS	7 petaFLOPS
INT8 Tensor Core	9 petaOPS	7 petaOPs
FP16/BF16 Tensor Core	4.5 petaFLOPS	3.5 petaFLOPS
TF32 Tensor Core	2.2 petaFLOPS	1.8 petaFLOPS
FP32	80 teraFLOPS	60 teraFLOPS
FP64 Tensor Core	40 teraFLOPS	30 teraFLOPS
FP64	40 teraFLOPS	30 teraFLOPS

Exploit Sparsity and Quantization

Custom hardware for sparse matrix-vector multiplication

Deep Compression Technique

Reduces storage requirements

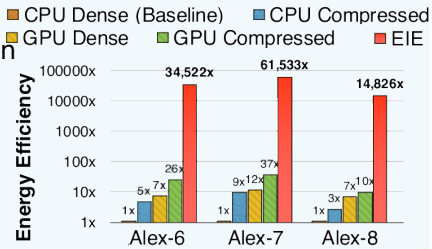
- Dedicated sparse matrix/vector representation
  ⇒ Eliminates redundant connections
- Quantizes weights down to 5 bits

Quantization of AlexNet weights

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

Weights stored into on-chip SRAM

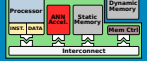
⇒ 5 pJ/access (vs. 640 pJ/access off-chip DRAM)



Power efficiency

600 mW for Alexnet Fully-Connected layers

S. Han et al., *"EIE: Efficient inference engine on compressed deep neural network"*, ISCA, 2016



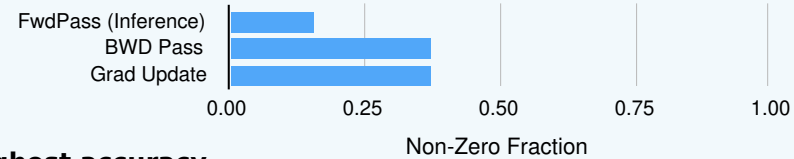
Acceleration using Low-Precision (ternary) weights

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

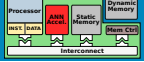
⇒ 93% on the ImageNet object classification challenge

⇒ Divide by 3 the number of FP operations

G. Venkatesh et al., *"Accelerating Deep Convolutional Networks Using Low-Precision and Sparsity"*, ICASSP, 2017

S. Ma et al, *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*, arxiv, 2024

Exploit full-quantization



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

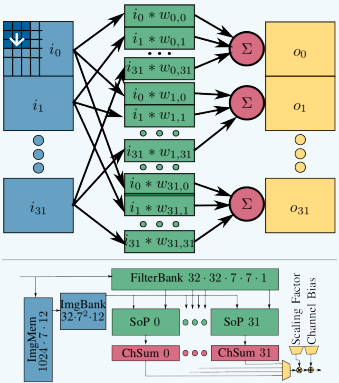
Large on-chip weights storage thanks to their size

- Latch-based standard cell memory

Flexible accelerator

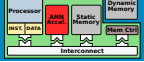
- 7 kernel sizes supported

⇒ 61.2 TOP/s/W at 0.6V

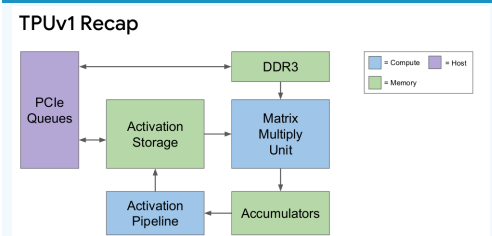


R. Andri et al., *"YodaNN: An architecture for ultralow power binary-weight CNN acceleration"*, IEEE TCAD, 2017

int8_t for parameters and activations



Google Edge Tensor Processor Unit

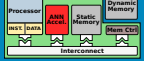


	V1	V2	V3
Clock Frequency (MHz)	800	1066	1066
# of (X, Y)-PEs	(4, 4)	(4, 4)	(4, 1)
PE Memory	2 MB	384 KB	2 MB
# of Cores per PE	4	1	8
Core Memory	32 KB	32 KB	8 KB
# of Compute Lanes	64	64	32
Instruction Memory	16384	16384	16384
Parameter Memory	16384	8192	8192
Activation Memory	1024	1024	1024
I/O Bandwidth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

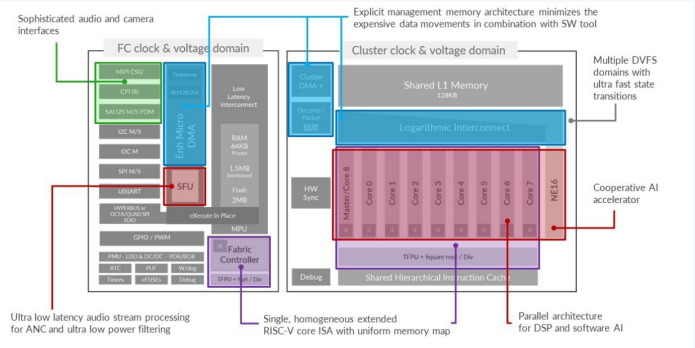
Comes for free in Tensorflow lite

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

8, 4, 2-bit computations



Greenwaves GAP9



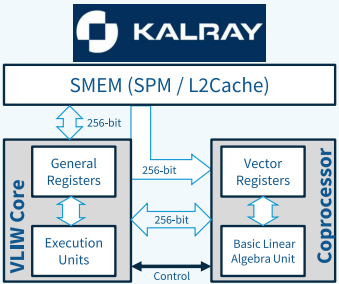
- 330 µW/GOp
- Up to 15.6 GOPs and 32.2 GMACs
- 8, 4, 2-bit SIMD computations
- Support from ML frameworks
- (RISC-V based)

`https://greenwaves-technologies.com/`, 2021

Exploit small FP representations

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 aligment operations
- Arithmetic performances
  ⇒ 128x INT8→INT32 MAC/cycle
  ⇒ 64x INT16→INT64 MAC/cycle
  ⇒ 16x FP16→FP32 FMA/cycle



- Integration within learning frameworks?

Very Heterogeneous Architecture

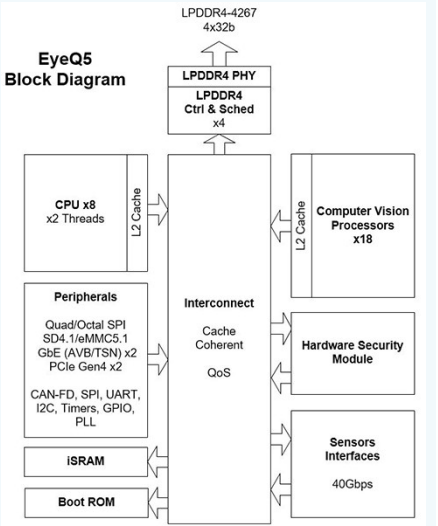
Mobileye EyeQ5

- 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/`

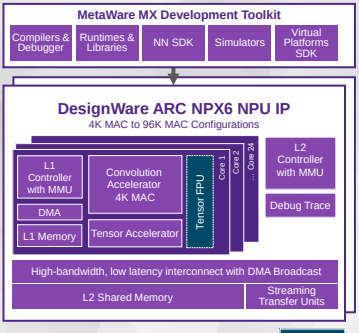


Very Heterogeneous Architecture

Synopsys ARC NPX6

new

DesignWare ARC NPX6 w/ 440 TOPS* Performance

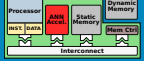


- **Scalable NPX6 architecture**
  - 1 to 24 core NPU up to 96K MACS (440 TOPS*)
  - Multi-NPU support (up to eight for 3500 TOPS*)
- Trusted **software tools** scale with the architecture
- **Convolution accelerator –** MAC utilization improvements with emphasis on modern network structures, including 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

Challenge with ASIC Design



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**
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
	0.7X Model Size	2.3X FLOPs	+1.5% Accuracy	2.4X Time	7.9X Time

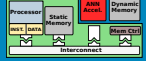
CPU/NNE = 8.1X

CPU/NNE = 2.5X

(Courtesy of Pierre Paulin, Synopsys)

Apple AI NPU not well suited to support Transformers, ...

General Purpose Embedded AI



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

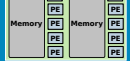
Software support out-of-the-box by major ”generic” frameworks

- Tensorflow[lite]
- Pytorch

Or ad-hoc ones

- OpenVino
- ...

Binary-Quantized Digital Computations



FINN: Framework for building FPGA

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

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

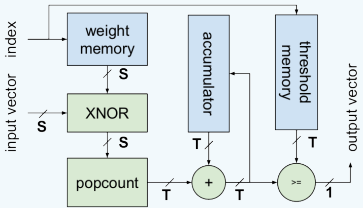
Weights kept in on-chip memory

⇒ Zynq-7000 FPGA technology

⇒ 80.1% accuracy for CIFAR-10

⇒ Total system power 25W

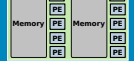
Convolution layer

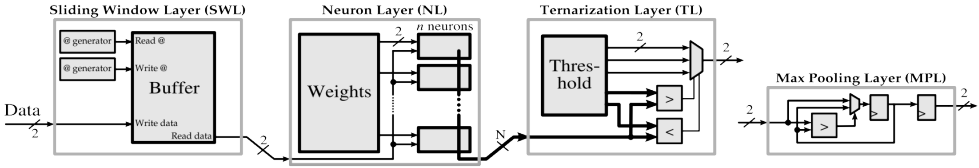


- Dot-product between input vector and row of synaptic weight matrix
- Compares result to a threshold
- Produces single-bit output

Y. Umuroglu et al., *"FINN: A Framework for Fast, Scalable Binarized Neural Network Inference"*, FPGA, 2017

Ternary weights *and* ternary activations





Ternary NN FPGA Architecture

- Balanced ternary values $t \in \{-1, 0, +1\}$
- Ternarization layer $\rightarrow$ Learnable thresholds
- Speed vs Area/Power trade-off possible

NN-64 with parallelism degree 128

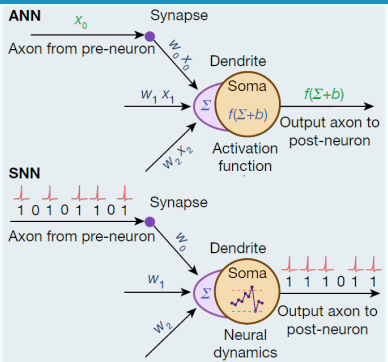
- (LUT+B)RAM throughput of 18.7 Tb/s
- End to end latency: 135 µs
- Max performance: 18.7 T(T)OP/s
- Max power: 11.5 W
- Peak efficiency of 5226 fps per Watt $\Rightarrow$ 1.62 T(T)OP/s/W or 810 G(T)MAC/s/W

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

Neuromorphic accelerators

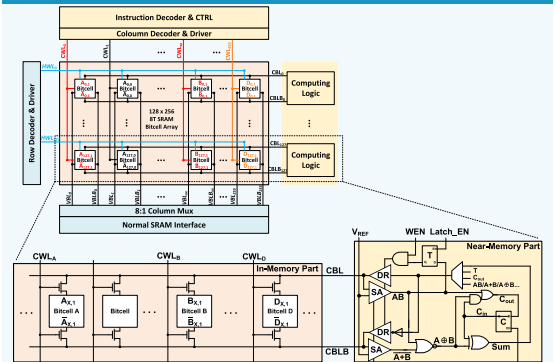
Name suggests it mimics brain-like functioning
Comes in two (orthogonal) flavors

Spiking Neural Networks



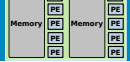
A. Basu et al. Spiking neural network integrated circuits: A review of trends and future directions, CICC 2022.

In/Near Memory Computing

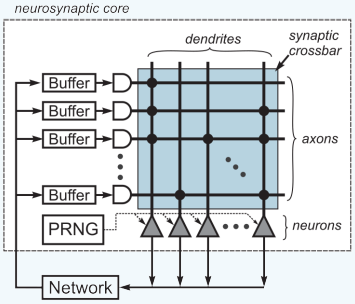


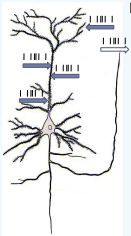
J. Wang et al, A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing, JSSC 2020.

Computations using spikes, not bits



TrueNorth: Integrated Chip for Spiking Neural Networks (IBM)





Romain Brette, *Computing with spikes*

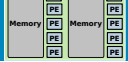
- Neurons communicate sending spikes
- Data encoded according to frequency, time, and spatial distribution of spikes

Non-“Von Neumann” architecture

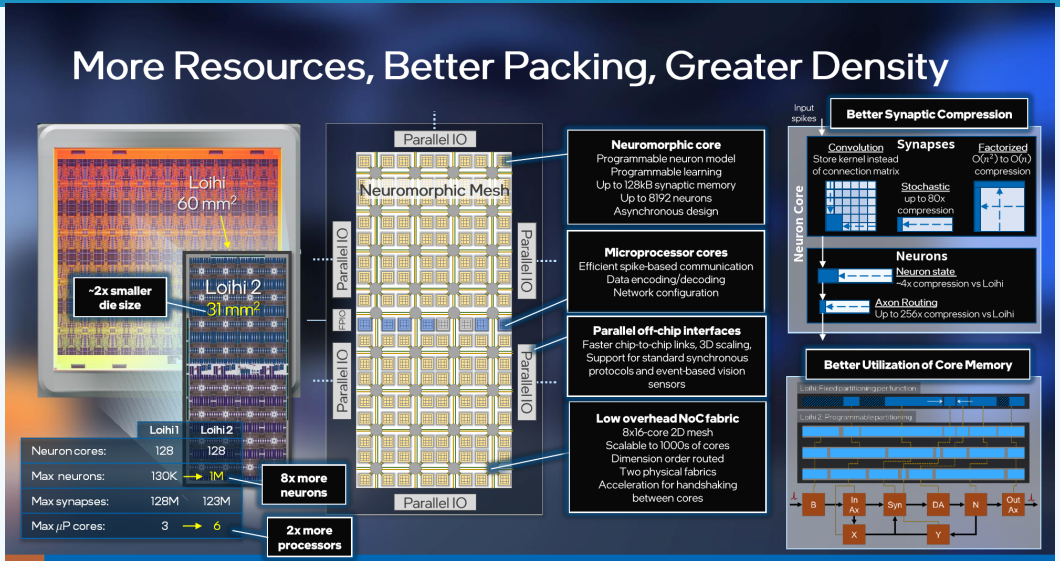
⇒ 4096 neuromorphic cores
⇒ 1 million digital neurons
⇒ 256 millions synapses
⇒ 46 GSOP/s/W at 65 mW
Made w/ Asynchronous logic

- Neuromorphic core
  256 neurons (PE) + 64k synapses (memory)
- Memory and computation physically close to each other
- Reduction of power consumption

Computations using spikes, not bits

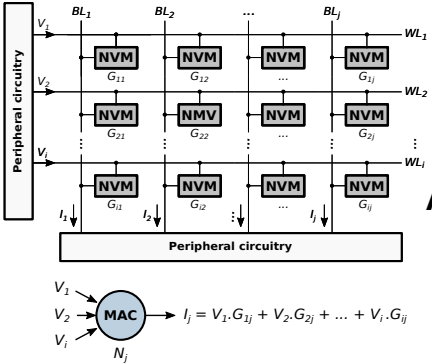


Loihi: Integrated Chip for Spiking Neural Networks (Intel)



Emerging Processing-In-Memory (PIM) Approaches

MAC operations using Non Volatile Memory (NVM)



Computation accelerated by NVM arrays

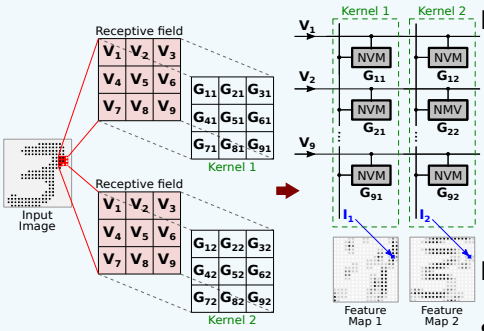
- Synaptic weights are **not** stored in external memories
  ⇒ Zero transfers between memory and processing elements
  ⇒ Reduction of energy consumption

Arrays of resistive RAM devices

- Resistances vary according to voltages
- No CMOS access devices but complex peripheral circuitry
- Analog, intrinsically approximate, computations

Resistive RAM Crossbar Implementations

Convolution Kernel in 12×12 Array



From receptive field to feature maps...

- Receptive field = row voltages
- Convolution kernel = column of resistive devices
- Convolution operation = column current

Interactive protocol programs kernels

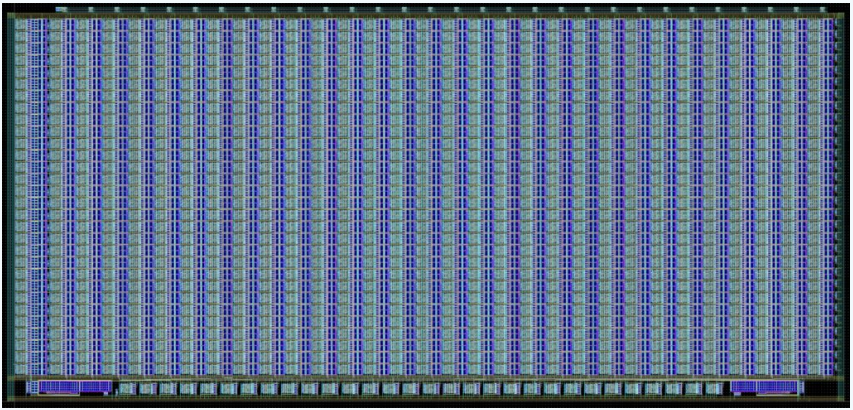
Single demonstration on digits of MNIST

L. Gao et al., *"Demonstration of Convolution Kernel Operation on Resistive Cross-Point Array"*, IEEE Electron Device Letters, 2016

P. Chi et al., *"PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRam-Based Main Memory"*, ISCA, 2016

Analog Inference Engine

aiLINEAR product due Q4 2024

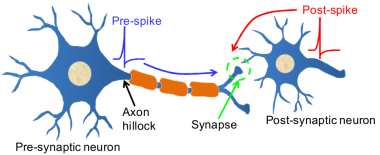


ReRAM based Analog-In-Memory Computing

Forsees 2 to 3 order of magnitude better power efficiency compared to Digital

(`https://ailinear.com/inference/`)

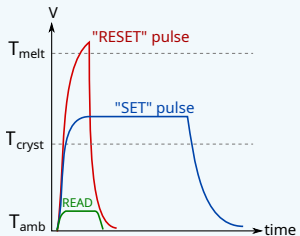
Artificial synapse using NVM

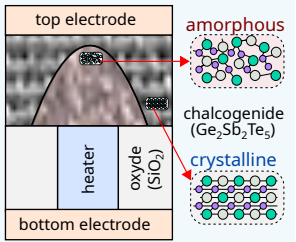


- Modeling synapses between neurons
- Input and output potentials fired between neurons (spikes)
- Synaptic connections are potentiated or depressed

Electronic Synapses Modeled by Phase Change Memory (PCM) Devices

- Programmed in different states (conductances)
- Compatible with CMOS components
- Scalable to nanometric dimensions





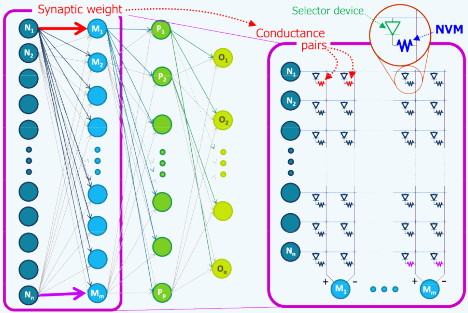
PCM as Synaptic Element

500x661 PCM Crossbar Array

Large scale implementation

- 3-layer perceptron
- 916 neurons
- 164865 synaptic connections

⇒ Accuracy: 82% (MNIST)
⇒ Low-power: at least 120x (vs. GPU)



S. Burc et al., *"Experimental Demonstration of Array-Level Learning with Phase Change Synaptic Devices"*, IEDM, 2013

G.W. Burr et al., *"Experimental Demonstration and Tolerancing of a Large-Scale NN (165000 Synapses) using PCM as the Synaptic Weight Element"*, 2015

G.W. Burr et al., *"Large-Scale Neural Networks Implemented with NVM as the Synaptic Weight Element: Comparative Performance Analysis"*, IEDM, 2015

Take Away

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 size and bandwidth requirements
- Memory organization
  ⇒ Optimized access to parameters and intermediate results
  ⇒ Structured sparsity

In/Near Memory Computing is progressing

- Minimal data movements, in-place computations
- Might even be analog
- But harder to manufacture and use

Take Away

But HW guys cannot do that alone!

- Relevant number representations
- Compression
- Quantization aware training
- Sparsity aware training
- HW accelerated low bit-width learning
- Frameworks to facilitate HW accelerators usage
- ...

Must see

MIT Tutorial on Digital ML Hardware
https://www.rle.mit.edu/eems/publications/tutorials/
NVIDIA Arith keynote on number representation
https://arith2022.arithsymposium.org/keynotes.html#Dally

Thanks !

For listening

Q & A

And to my fellow coworkers

@TIMA	and elsewhere
Adrien Prost-Boucle,	Hande Alemdar (LIG, now METU)
Ana Pinzari,	Vincent Leroy (LIG, now Google)
Liliana Andrade,	Henri-Pierre Charles (CEA),
Olivier Muller,	Florent de Dinechin (CITI),
Alban Bourge,	Fabrice Rastello (INRIA),
...	...