MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

A Tour of accelerator architectures for AI/ML applications

EECS 573 Topic Lecture

Peijing Li, peijli@umich.edu & Mason Nelson, nelsontm@umich.edu

Monday, November 13, 2023

Learning Objectives

High level: learn the design process behind working ML accelerators.

- Characterize common operations in Machine Learning that can be accelerated
- Understand why Machine Learning is an inefficient task for general-purpose processors and why accelerators are necessary
- Become acquainted with various architecture paradigms, both historical and contemporary, to AI accelerator design
- Review current academic research and commercial products that implement the basic architecture paradigms
- Reflect on possible next steps in accelerator design

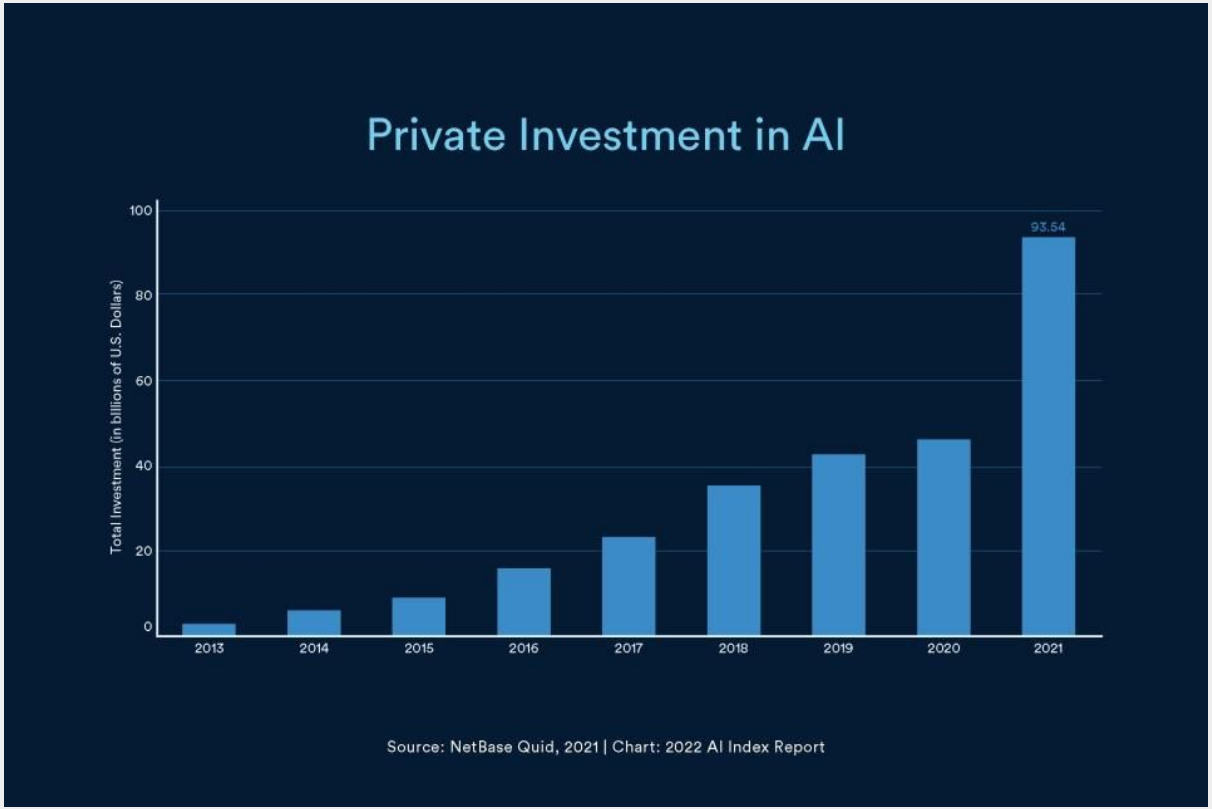
Lecture Outline

1. Introduction to Machine Learning
2. Why accelerators?
3. Foundations of accelerator architecture paradigms
4. Examples of accelerator implementations
5. Next steps for the accelerator design

Lecture Outline

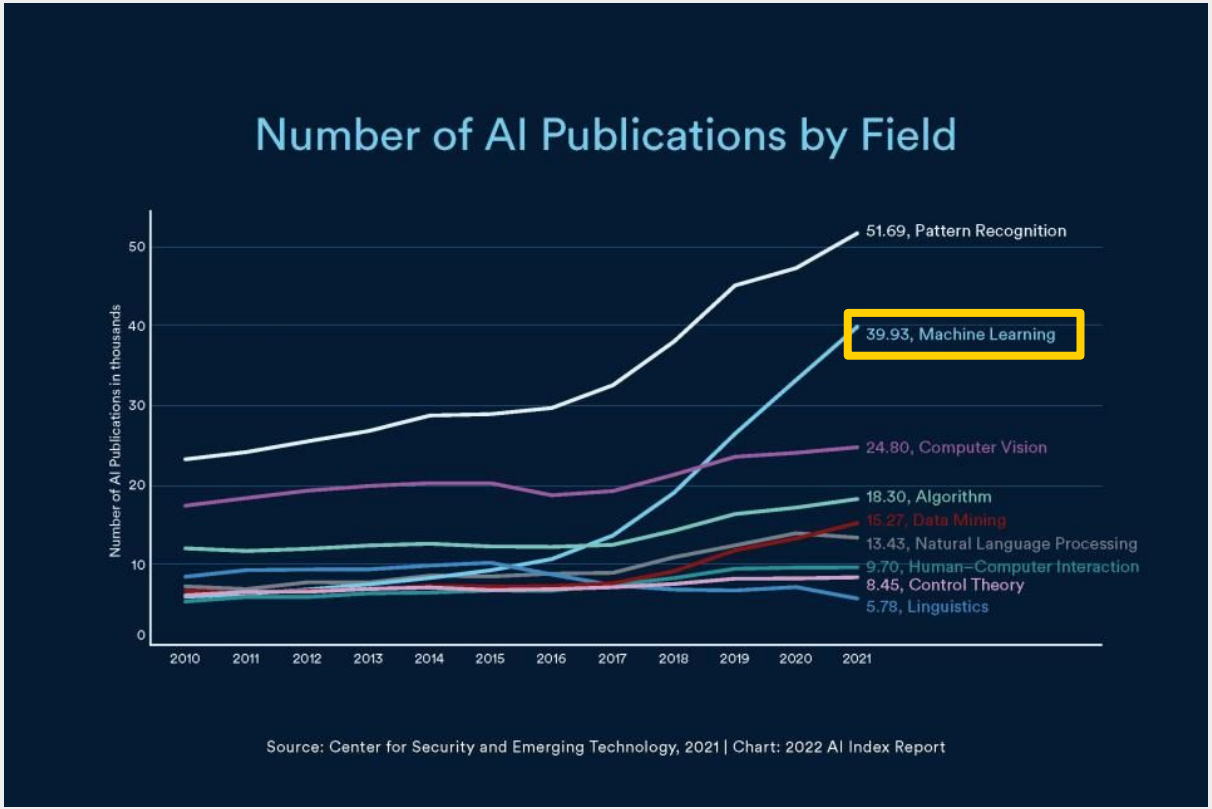
1. Introduction to Machine Learning
    a. The meteoric rise of AI/ML
    b. Basic concepts and examples of neural networks in machine learning
    c. Computational patterns of AI/ML algorithms
2. Why accelerators?
3. Foundations of accelerator architecture paradigms
4. Examples of accelerator implementations
5. Next steps for the accelerator design

The Rise of AI/ML



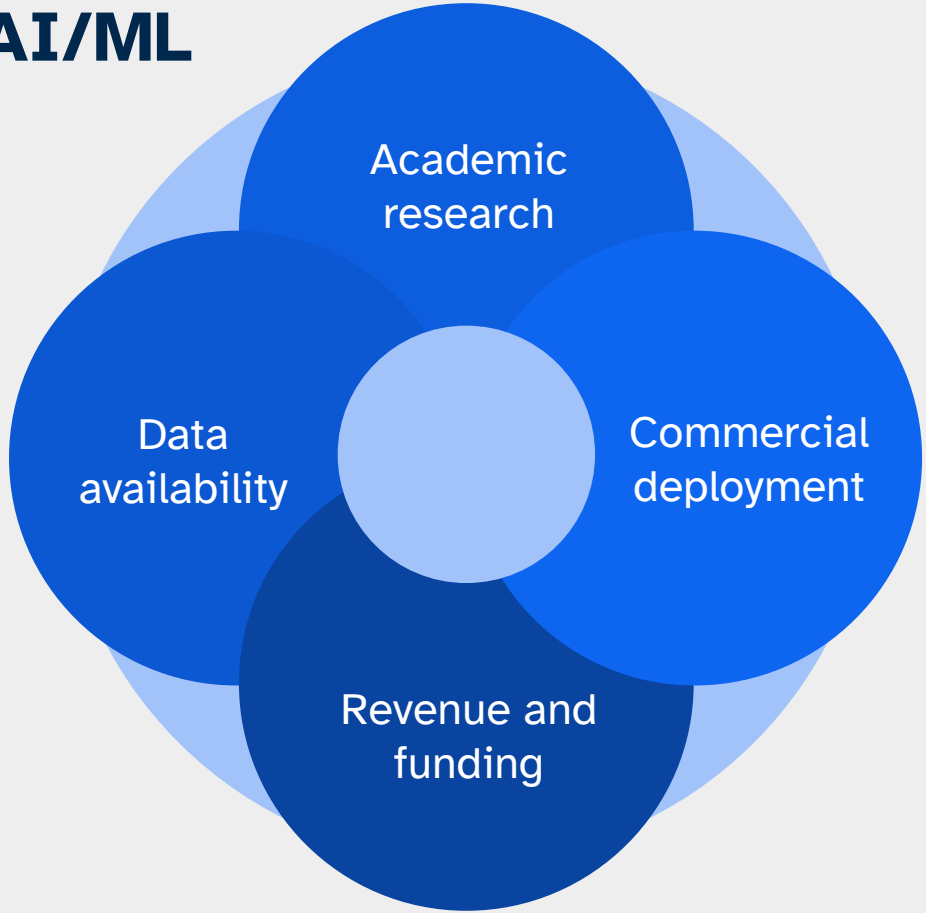
Source: https://hai.stanford.edu/news/state-ai-9-charts

The Rise of AI/ML



Source: https://hai.stanford.edu/news/state-ai-9-charts

The Rise of AI/ML



Basics of Neural Networks: The most common ML

Multiple ***layers***.

Each layer takes in some vector of ***features*** representing 1 data point. Output some vector of other features (labels, values, etc.) for that data point. $\mathbf{y} = \mathbf{f}(\mathbf{x})$

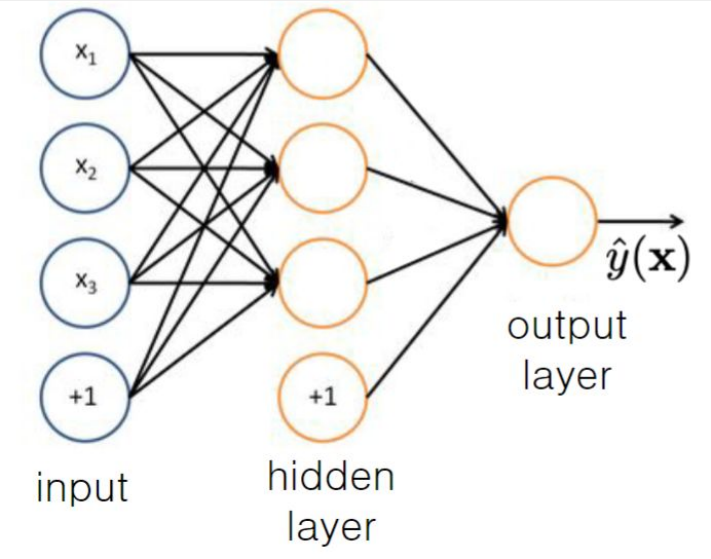


Image Credit: Yoshua Bengio via EECS 545 W23 Slides

Machine Learning Inference

Fancy word for just running the model through each layer

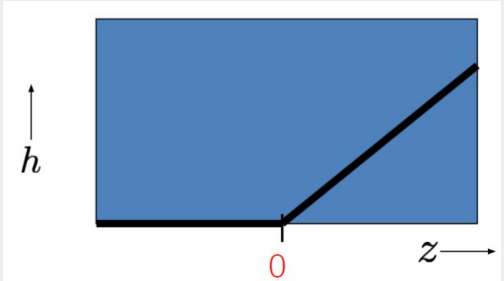
$$z = b + \sum_i x_i w_i$$

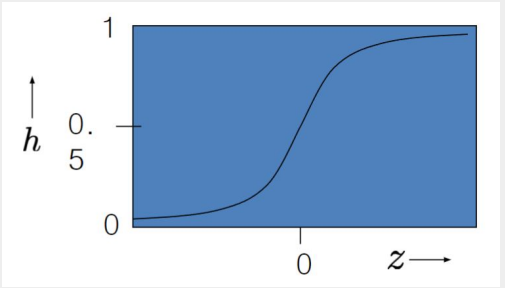
Typical linear layer setup: for each output value, perform scalar product of the input vector and ***weight*** vector. (This is linear since scalar products are linear transformations)

Combining the results of every product, we get ***matrix multiplication***.

Nonlinear layers: can apply any parameterized nonlinear function elementwise

$$h = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$





$$h = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$

Image Credit: Geoff Hinton via EECS 545 W23 Slides

Machine Learning Training

The process of developing the weight vectors. This is the learned part - the layers themselves are determined by the programmer.

Do some number of times until convergence or near-convergence:

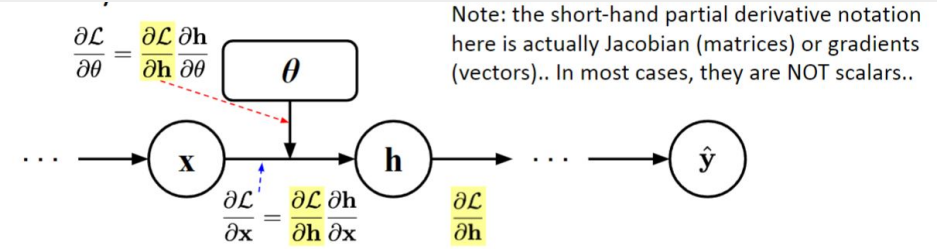
$(\mathbf{x}, \mathbf{y})$: Sample an example (or a mini-batch) from data

$\hat{\mathbf{y}} \leftarrow f(\mathbf{x}; \theta)$: Forward propagation

Compute $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$

$\nabla_\theta \mathcal{L}$: Backward propagation

$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$: Update weights using (stochastic) gradient descent



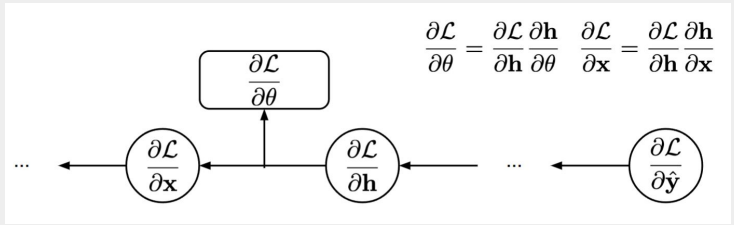
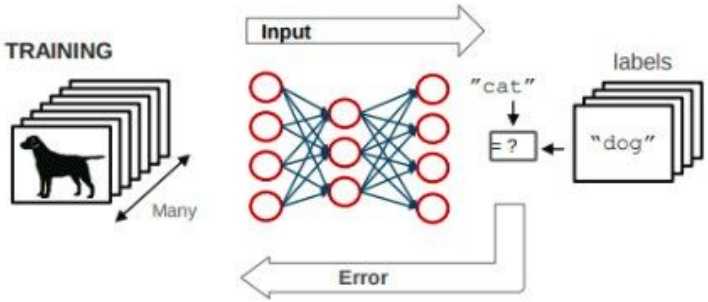


Image Credit: EECS 545 W23 Slides

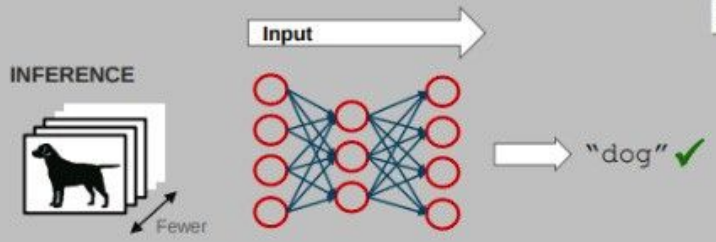
Computational Patterns of AI/ML

- Massive data-level **parallelism**
- Only a **handful of operations**
- Control & data flow **known at compile time**

Recap



Training: Process for machine to "learn" and optimize model from data



Focus

Inference: Using trained models to predict/estimate outcomes from new observations in efficient deployments

Image Credit: "Deep Learning Training vs. Inference: What's the Difference?" from Xilinx, EECS 545 W23 Slides

ML Phases on Different Hardware

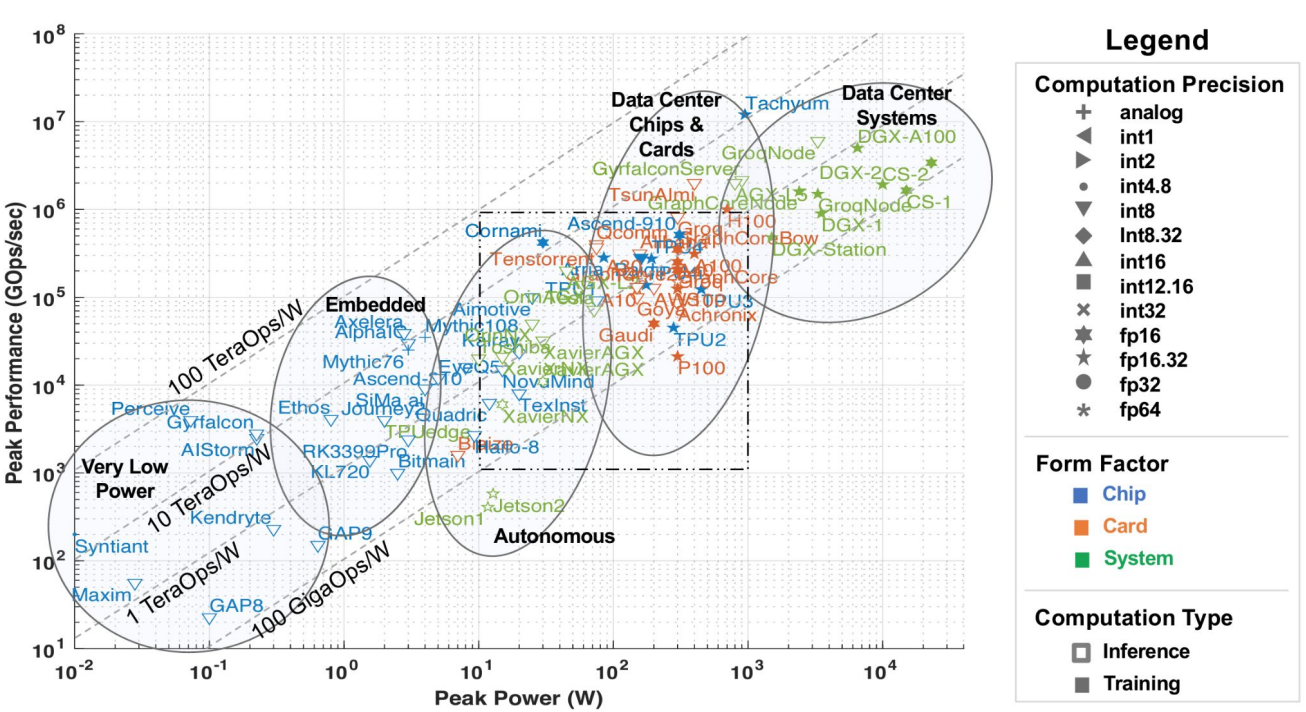


Fig. 2: Peak performance vs. power scatter plot of publicly announced AI accelerators and processors.

Image Credit: AI and ML Survey and Trends

ML Phases on Different Hardware

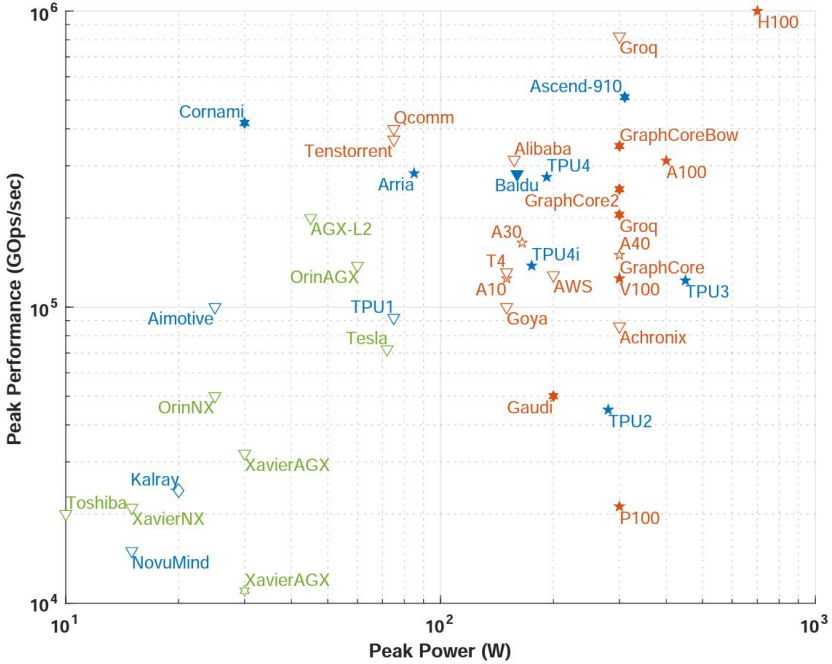


Fig. 3: Zoomed region of peak performance vs. power scatter plot.

Image Credit: AI and ML Survey and Trends

A few types of Neural Networks

Deep Neural Networks (DNNs): neural networks with multiple hidden layers.

Fully Connected Neural Networks (FCNNs): each node of each layer is connected to each node of the next layer

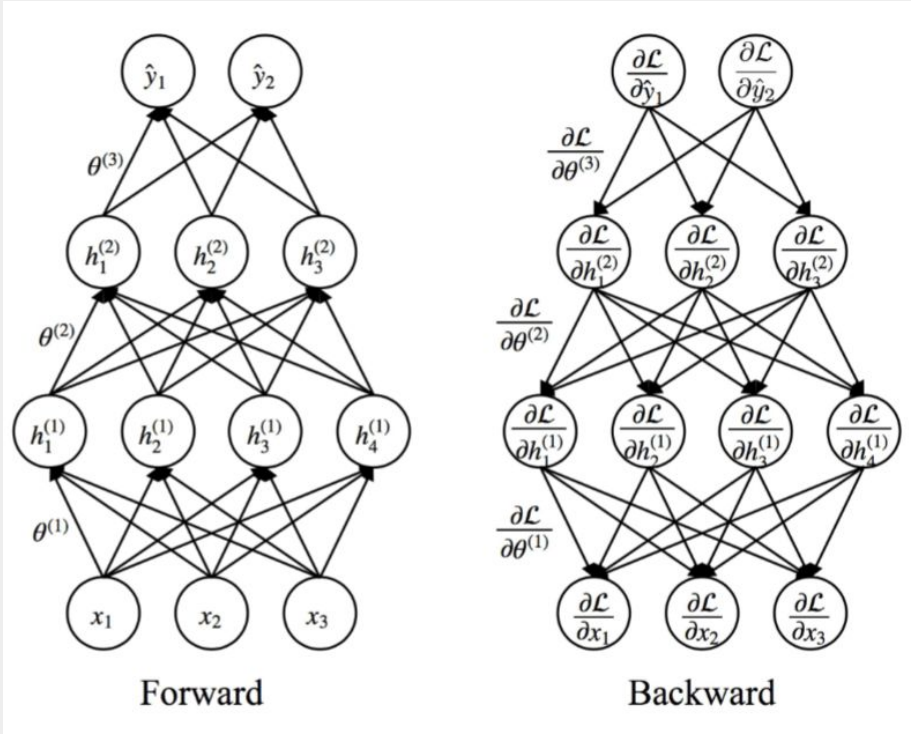
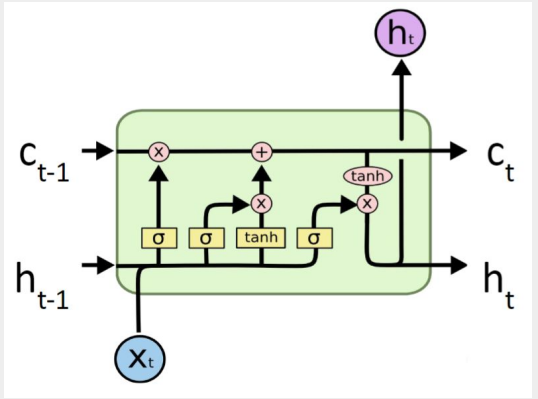


Image Credit: EECS 545 W23 Slides

A few types of Neural Networks

Recurrent Neural Networks (RNNs): neural networks to predict sequences. Each output of the RNN is based on the global input, plus all previous outputs. Very good for text, and very common in LLMs (Large Language Models). Many dependencies.



Long Short Term Memories (LSTM): RNNs adjusted to properly focus on the most recent nodes, while not totally ignoring old nodes.

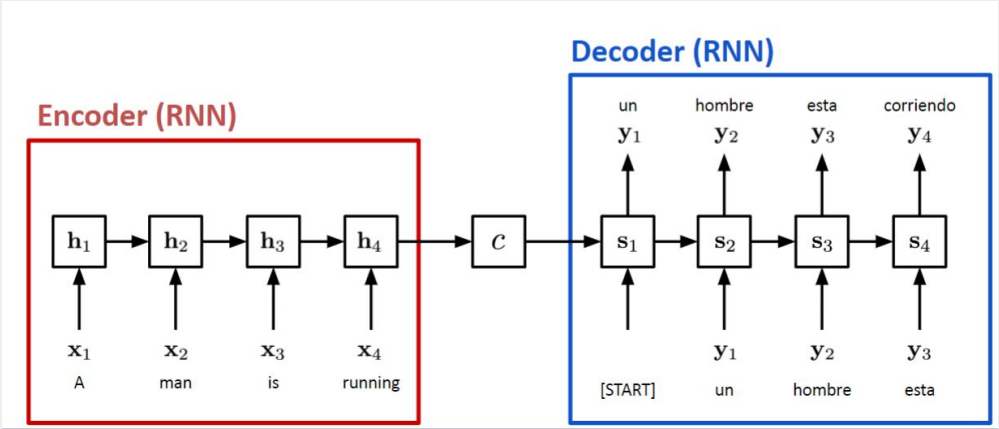
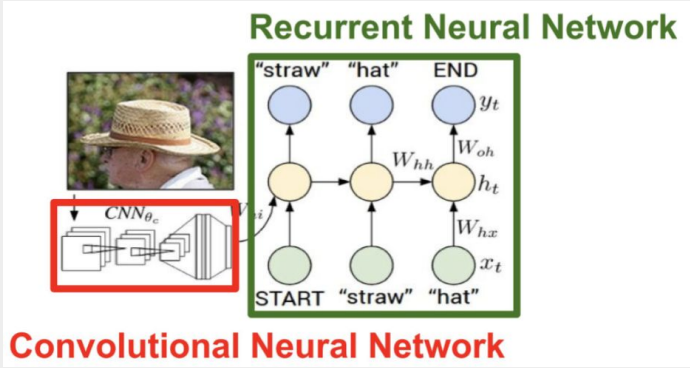


Image Credit: EECS 545 W23 Slides

A few types of Neural Networks

Convolutional Neural Networks (CNNs): neural networks where a few pixels are connected to the pixels in the next layer, weights tied to relative location. 2D and maybe 3D spatial locality.



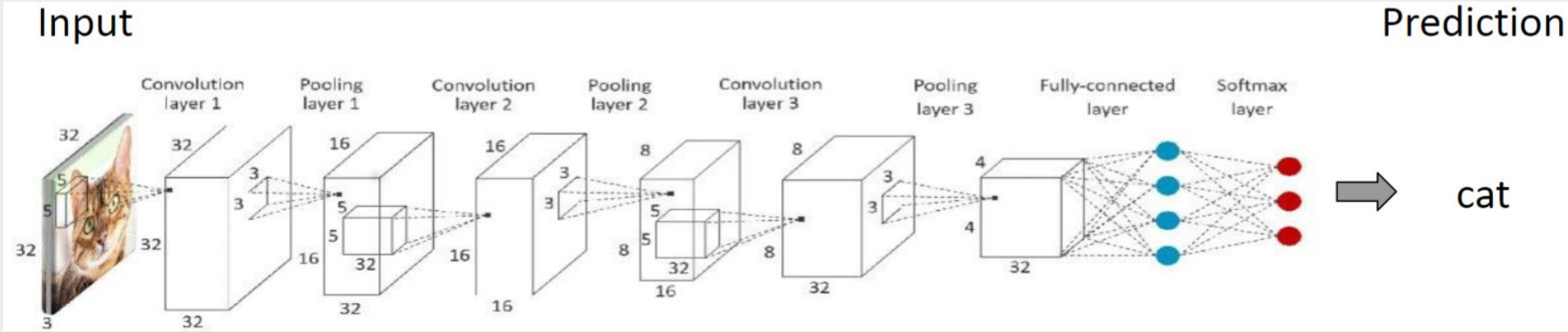


Image Credit: EECS 545 W23 Slides

Closer Look at Convolution

Alternative to stretching out an image and doing scalar product on all pixels

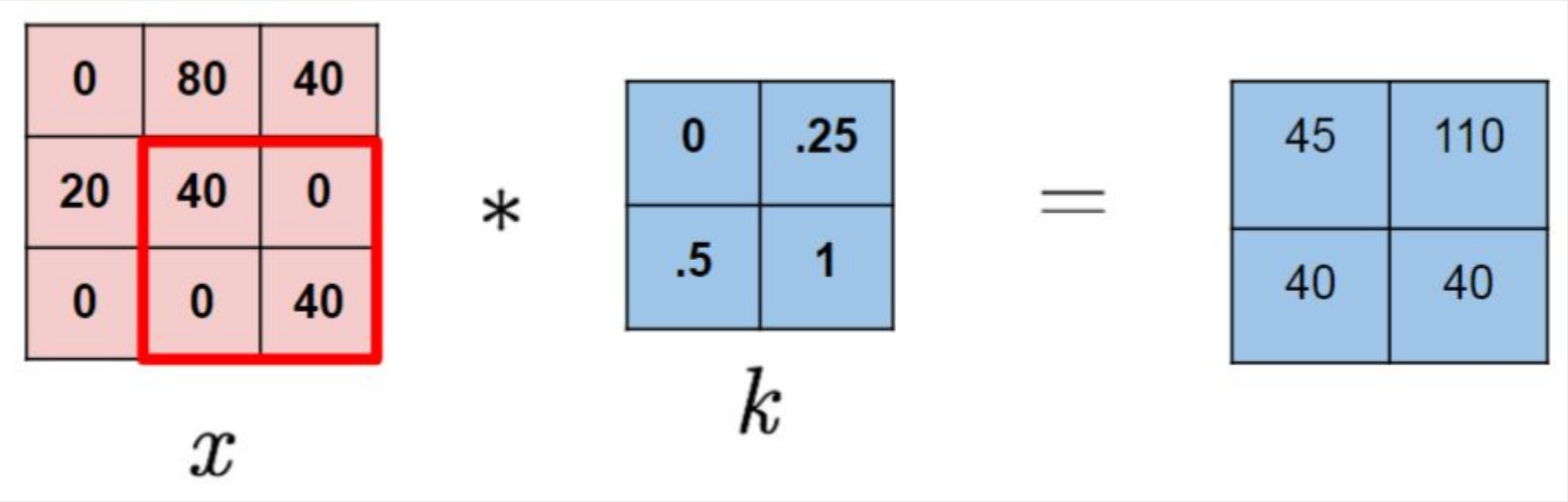
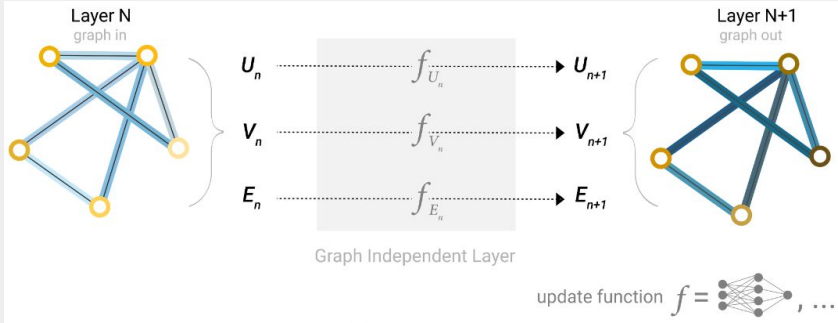


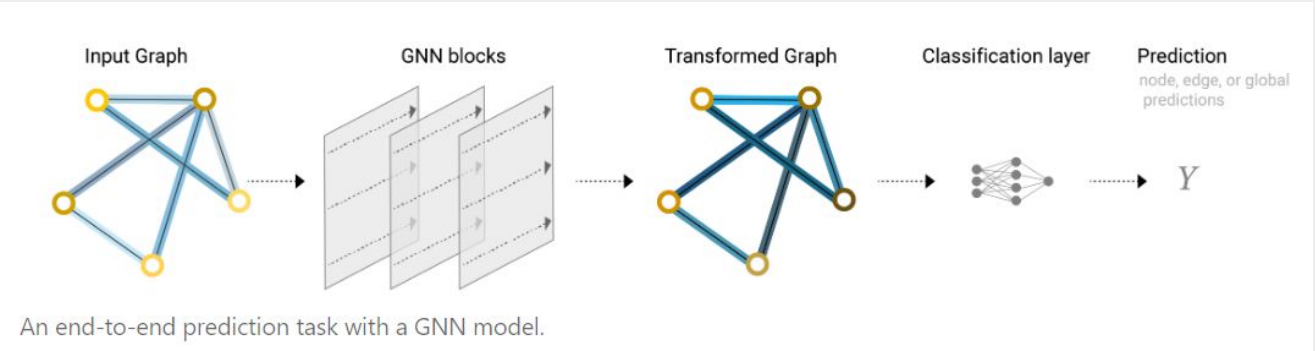
Image Credit: EECS 545 W23 Slides

A few types of Neural Networks

Graph Neural Networks (GNNs): neural networks for vertex-edge graphs. Each layer is iterated to propagate node features to edges and other nodes.

Memory accesses are very irregular. Mostly gather and scatter.





An end-to-end prediction task with a GNN model.

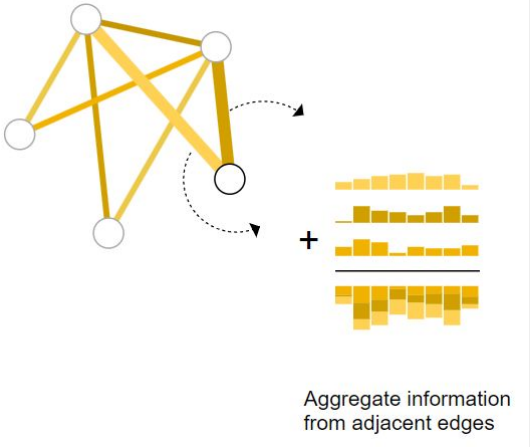


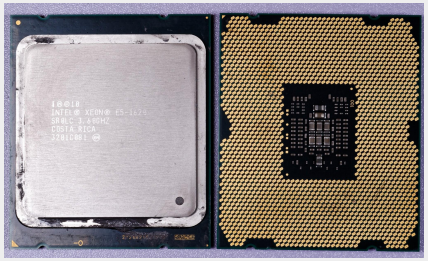
Image Credit: https://distill.pub/2021/gnn-intro/

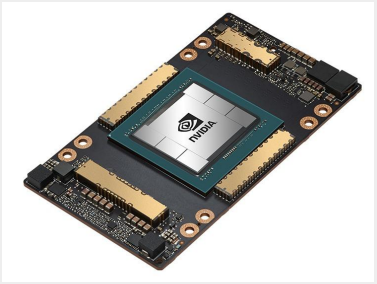
Lecture Outline

1. Introduction to Machine Learning
2. Why accelerators?
    a. Status quo of general purpose hardware
    b. Computational requirements of ML algorithms
    c. Stagnation of general-purpose CPU design
    d. Dark silicon problem and solutions
    e. AI is the perfect field for accelerator solutions!
3. Foundations of accelerator architecture paradigms
4. Examples of accelerator implementations
5. Next steps for the accelerator design

CPU vs GPU

The Status Quo





CPUs

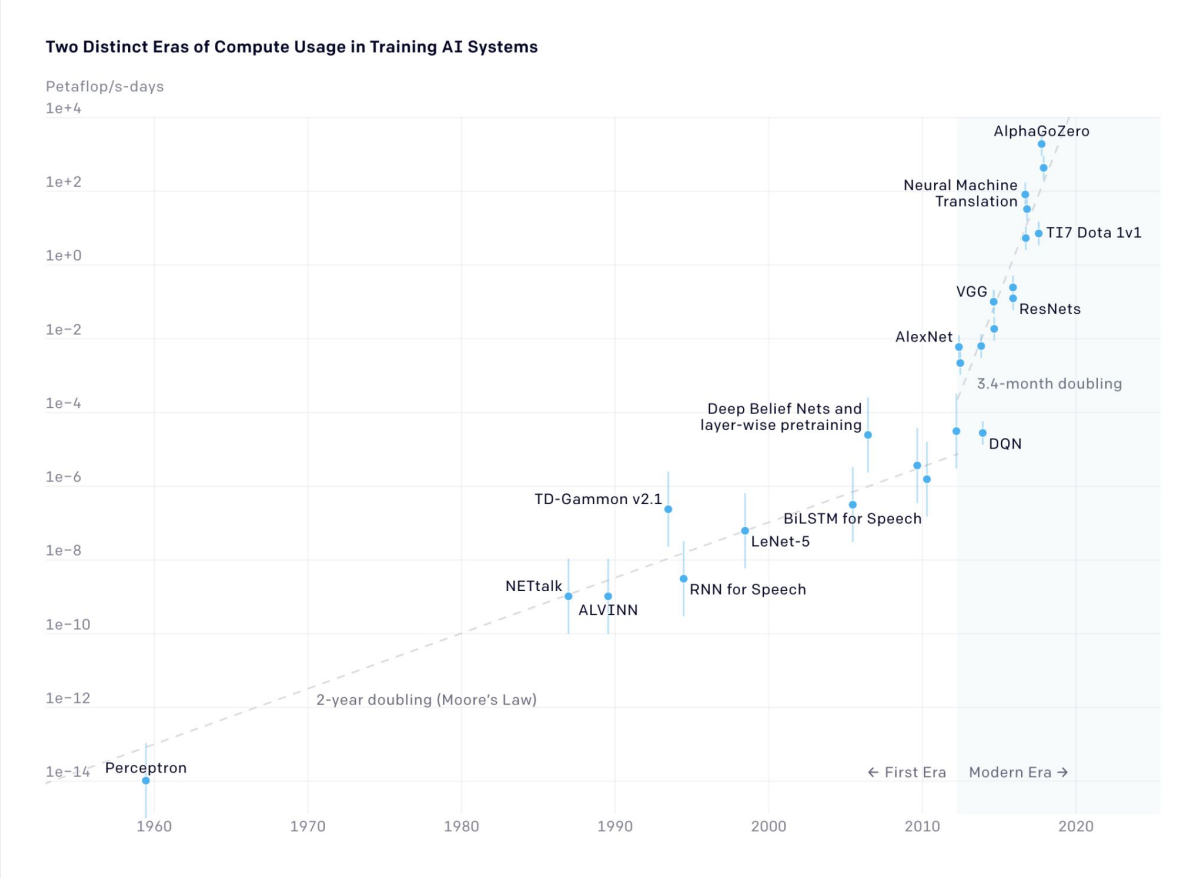
- OoO cores are great for programs with long dependency chains.
- Good for inference LLM and RNN
- Bad for massive tasks with large DLP, i.e., ML training
- High power budget
- Support for other general-purpose instructions

GPUs

- Excellent at simple matrix multiplication and CNNs
- Easy control flow
- High efficiency
- High power budget

“Moore’s Law is still alive in ML!”

Two eras in the complexity and resource usage of ML models: AlexNet and Transformers **Petaflop requirements double every 3-4 months!**

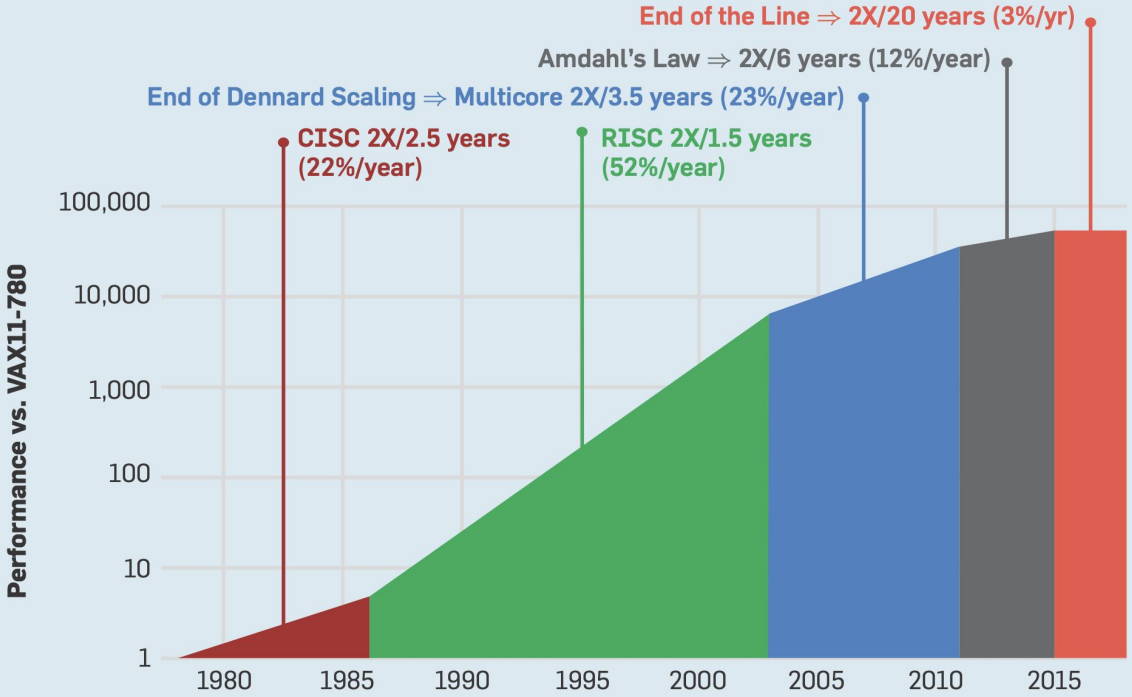


Source: https://openai.com/research/ai-and-compute

… But general purpose HW performance can't keep up!

- Freq scaling
- Multicore systems
- TLP and DLP
- What's next?

Figure 6. Growth of computer performance using integer programs (SPECintCPU).



Source: John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. Commun. ACM 62, 2 (February 2019), 48–60. https://doi.org/10.1145/3282307

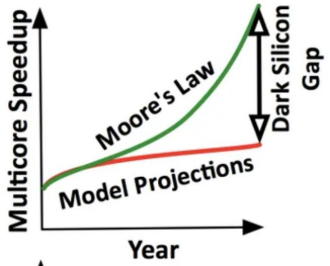
Dark silicon and the “utilization wall”

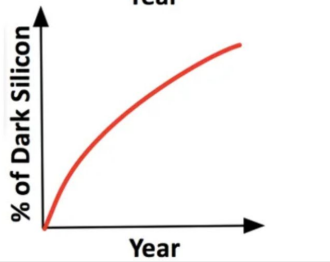
Classical scaling

Device count	S^2	1
Device frequency	S	
Device cap (power)	1/S	
Device V_{dd} (power)	**$1/S^2$**	
Utilization ?		

Leakage-limited scaling

Device count	S^2	S^2
Device frequency	S	
Device cap->power	1/S	
Device V_{dd} (power)	**~1**	
Utilization ?		

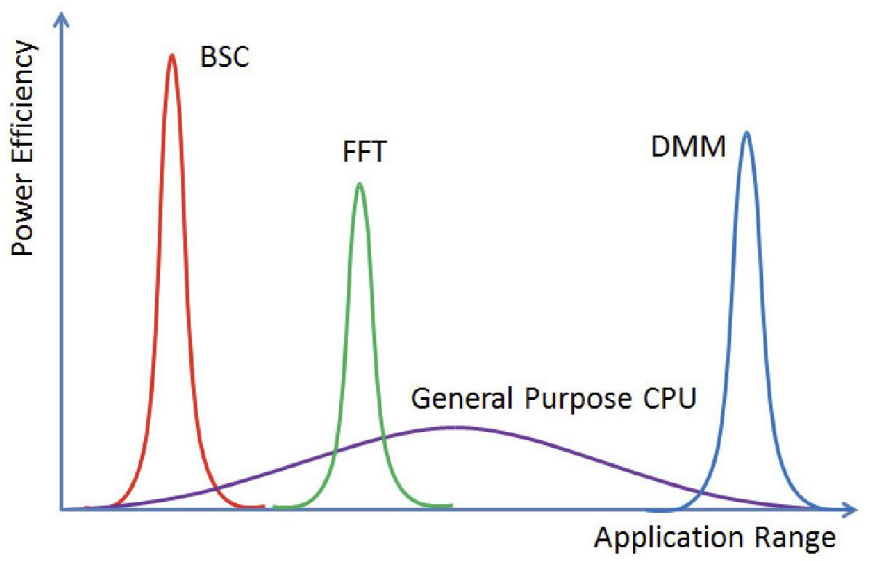




- As long as there’s a power constraint, **we will not be able to utilize more than a given fraction of the chip’s transistors** (and consequently, cores).
- The remaining parts of the chip must be powered down → **“dark silicon.”**
- Exponential increase in dark silicon percentage given constant transistor power

Source: Yavits, L., Morad, A., Weiser, U., & Ginosar, R. (2017). MultiAmdahl: Optimal Resource Allocation in Heterogeneous Architectures (arXiv:1705.06923). arXiv. https://doi.org/10.48550/arXiv.1705.06923

Breaking the “utilization wall”



“Specialization” to the rescue!

High power efficiency for a narrow range of applications…

- Black-Scholes
- Fast Fourier Transform
- Dense Matrix Multiplication

Source: Yavits, L., Morad, A., Weiser, U., & Ginosar, R. (2017). MultiAmdahl: Optimal Resource Allocation in Heterogeneous Architectures (arXiv:1705.06923). arXiv. https://doi.org/10.48550/arXiv.1705.06923

What Makes AI a Good Target for Acceleration?

- Computational pattern
  - Parallelization
  - Few Operations
  - Simple Control flow
- Widely used libraries (e.g. PyTorch)
- Deficiencies/bottlenecks with state-of-the-art general-purpose hardware
  - CPU is not powerful enough
  - GPU may be overkill in mobile/edge applications

Lecture Outline

1. Introduction to Machine Learning
2. Why accelerators?
3. Foundations of accelerator architecture paradigms
    a. Accelerator definition
    b. Walkthrough of different components: Instructions, Scheduling, Data, Memory
4. Examples of accelerator implementations
5. Next steps for the accelerator design

Definition of Accelerators

- Hardware is structured to cater only the mission-specific operations
- Typically come with custom compiler and/or ISA for easier interface between hardware and high-level libraries and algorithms

Very Long Instruction Word (VLIW)

CISCy ISAs with RISC-type portions in each instruction, broken down by functional unit.

Good for launching multiple parallel tasks at once.

Bad when the you don't utilize the entire instruction

Modern VLIW just interpret regular instructions - GP VLIW ISAs failed, but sometimes come back in accelerators.

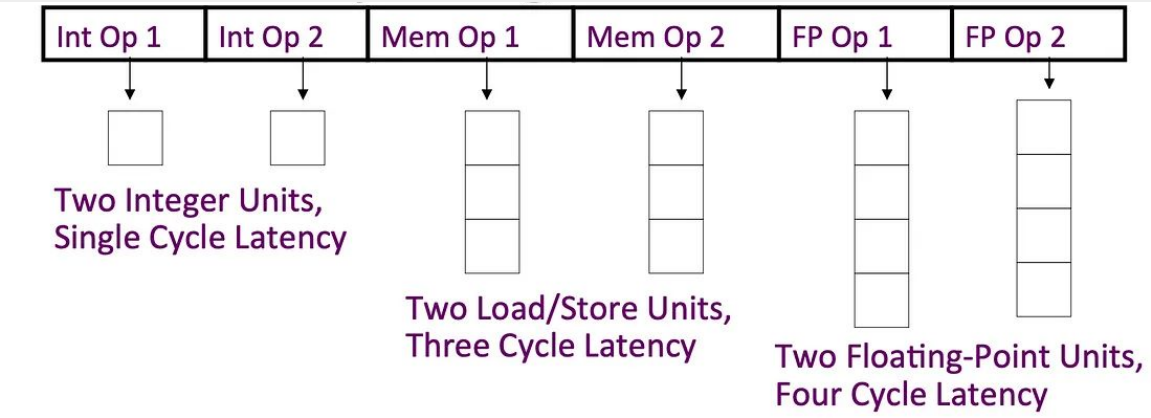


Image Credit: Fuchs

SIMD Instructions: Single Instruction Multiple Data

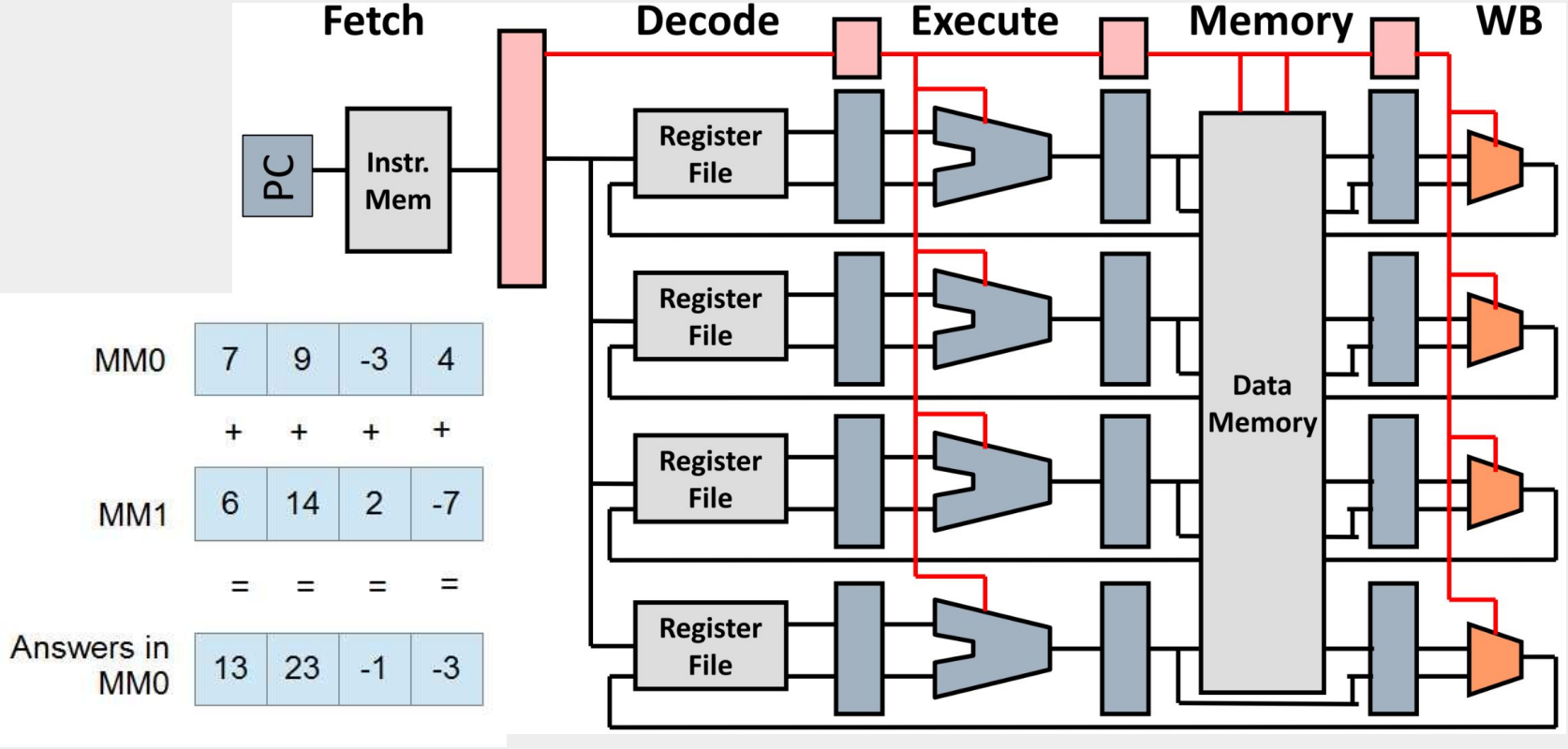
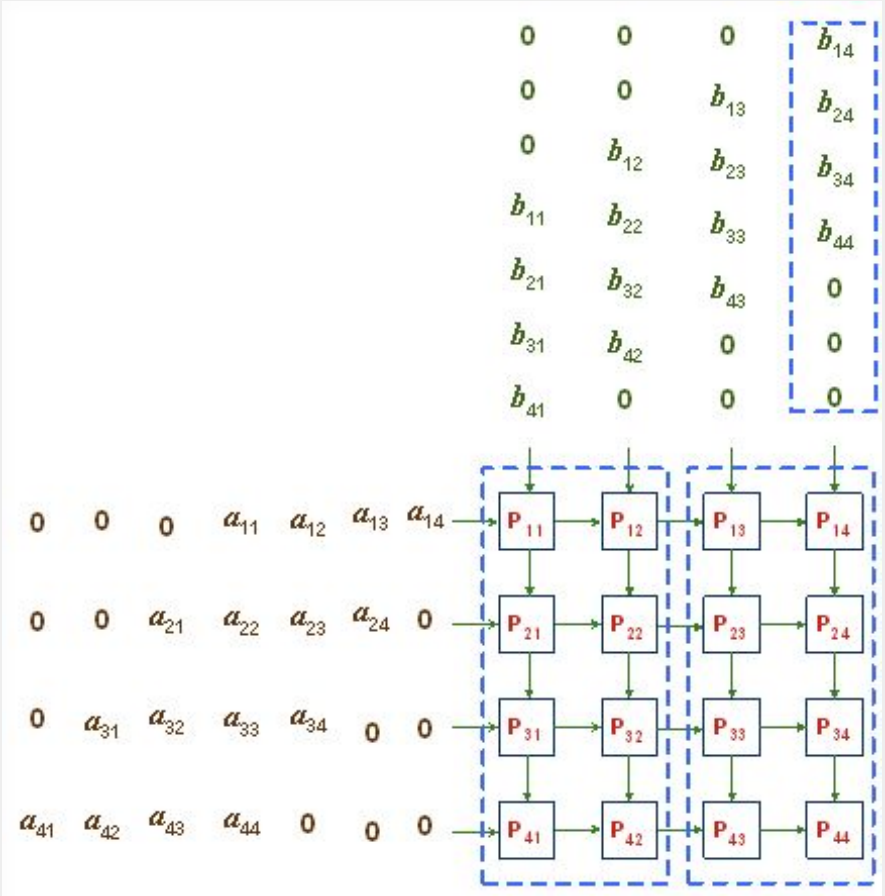


Image Credit: EECS 370 Slides (Brehob), Rose

Systolic Arrays

- Multiple processing elements **hardwired in fixed order**
- Each PE processes a portion of the data each cycle
- Efficient way to perform matrix optimization
- First proposed in 1970s
- Application in Tensor Processing Units and Nvidia's Tensor Cores.



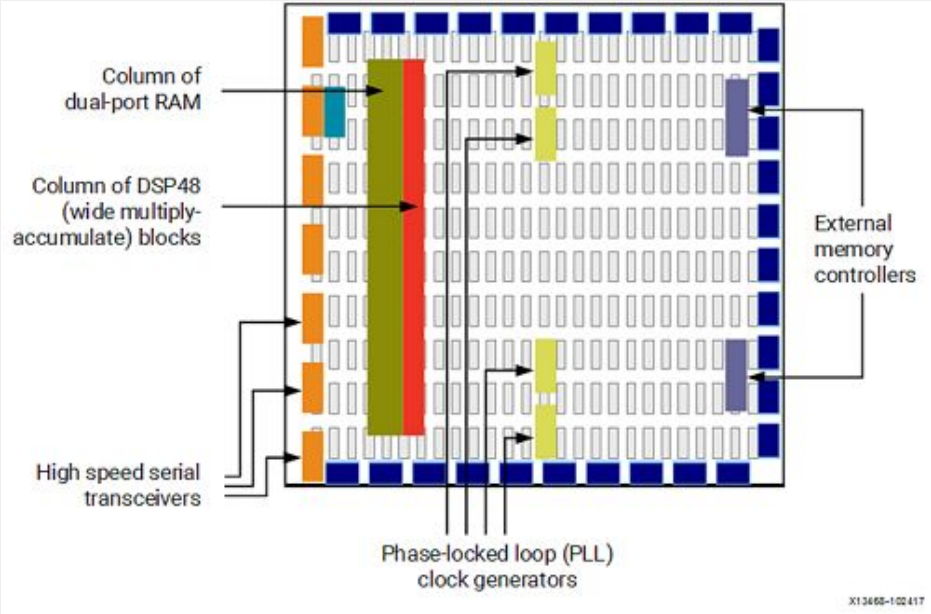
Source: http://ecelabs.njit.edu/ece459/lab3.php, Leiserson, C. E. (1979). Systolic arrays (for VLSI). Sparse Matrix Proceedings 1978, 1, 256–282.

Reconfigurable processor

- **Replicated arrays** containing **interconnected compute units, memory units**
- **Control plane** orchestrates how the data traverses and gets manipulated between the various units as the targeted program executes on the chip
- **Special-purpose compiler** constructs a configuration file that contains control bits that set the behavior of each element in the array
- Examples include FPGA/CGRA

Field-Programmable Gate Array

- Supports a wide computational spectrum by enabling bit-level configurability
- ALU operates on numbers of arbitrary widths
- On-chip memory blocks can be fused to construct memory spaces of varied sizes.
- Model chip designs written in HDL
- Fine-grained bit-level configurability is inefficient



Source: https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/odz1504034293215.html

Coarse-Grained Reconfigurable Array

- Proposed by Mirsky and DeHoin at MIT in 1996
- Array of processing elements connected in 2D network
- Each PE contains functional unit and register file
- Can achieve higher power efficiency due to simpler hardware and intelligent software techniques
- Special register allocation and PE mapping schemes required from compilers

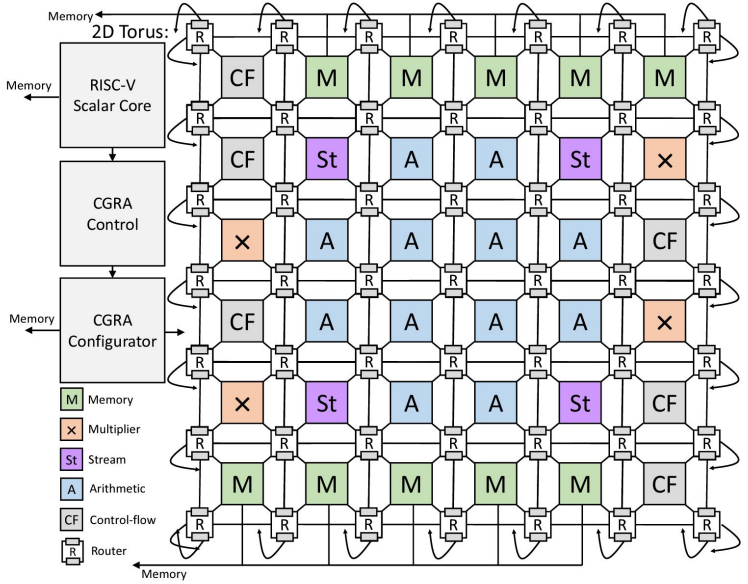
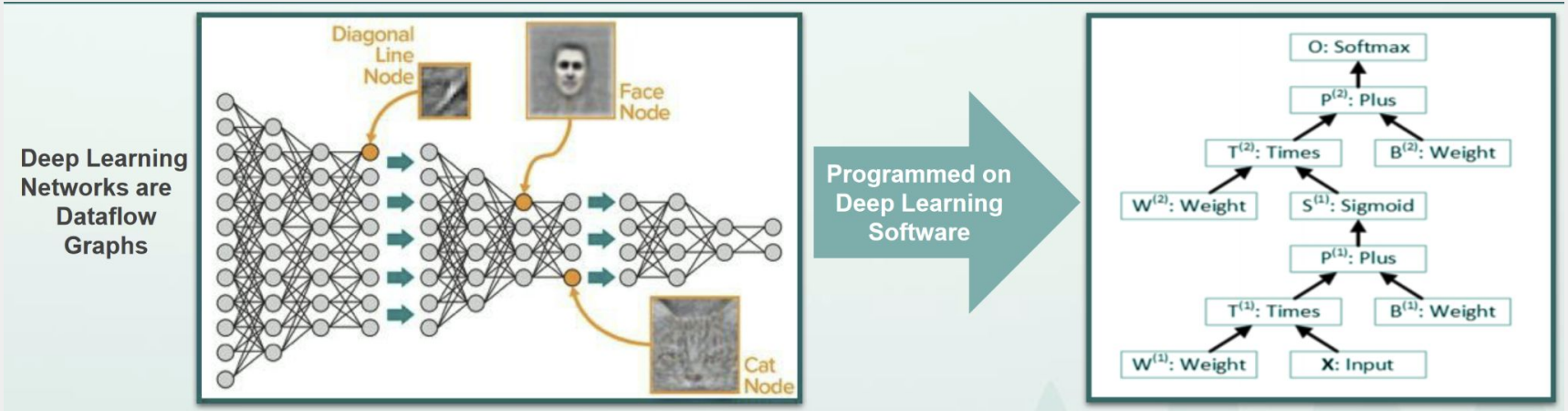


Figure 6: RipTide's ULP CGRA fabric.

Source: https://labs.engineering.asu.edu/mps-lab/research/cgra/

Dataflow processing

In the dataflow model, the program is represented as a dataflow graph (DFG) in which portions of the input data are computed using predetermined operands and the computer data "flows" all the way to the output according to the graph that is being represented and computed by the graph-like hardware. **Hardware is inherently parallel.**



Source: https://old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.60-NeuralNet1-Pub/HC29.22.610-Dataflow-Deep-Nicol-Wave-07012017.pdf

Dataflow processing

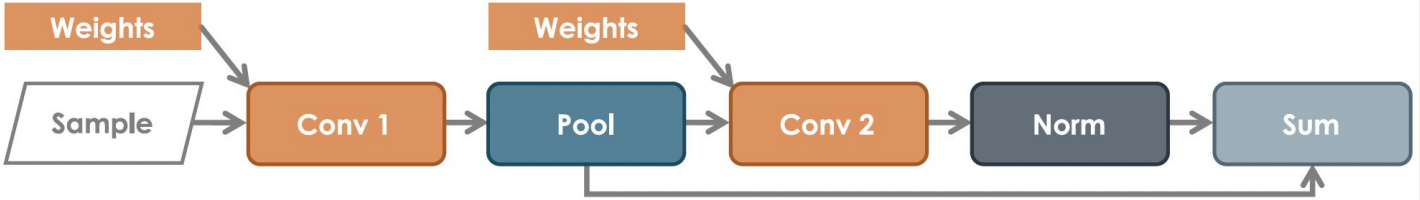


Figure 2 - Simple convolution graph

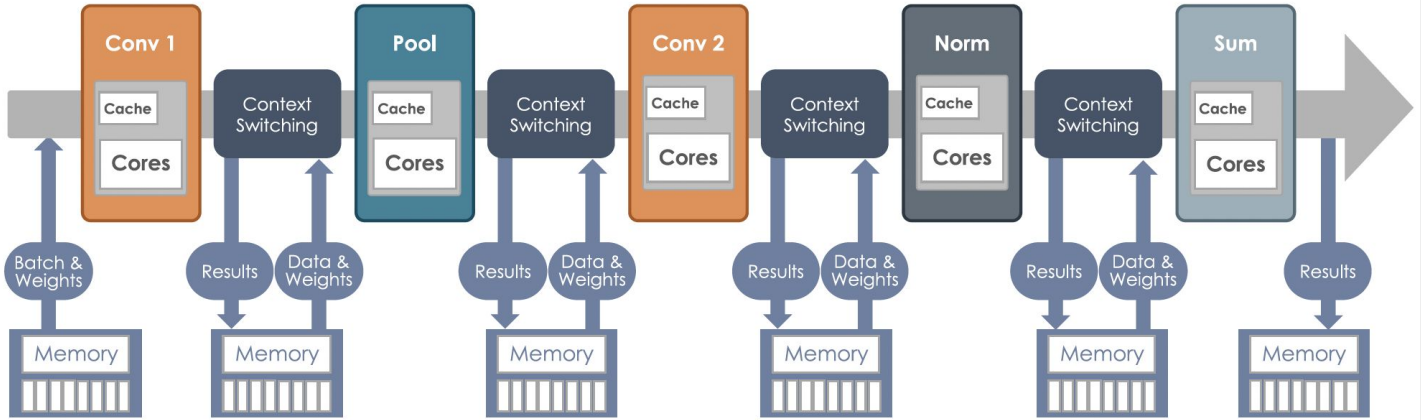


Figure 3 - Core-based kernel by kernel execution

Source:
https://sambanova.ai/wp-content/uploads/2021/04/SambaNova_Accelerated-Computing-with-a-Reconfigurable-Dataflow-Architecture_Whitepaper_English.pdf

Dataflow processing

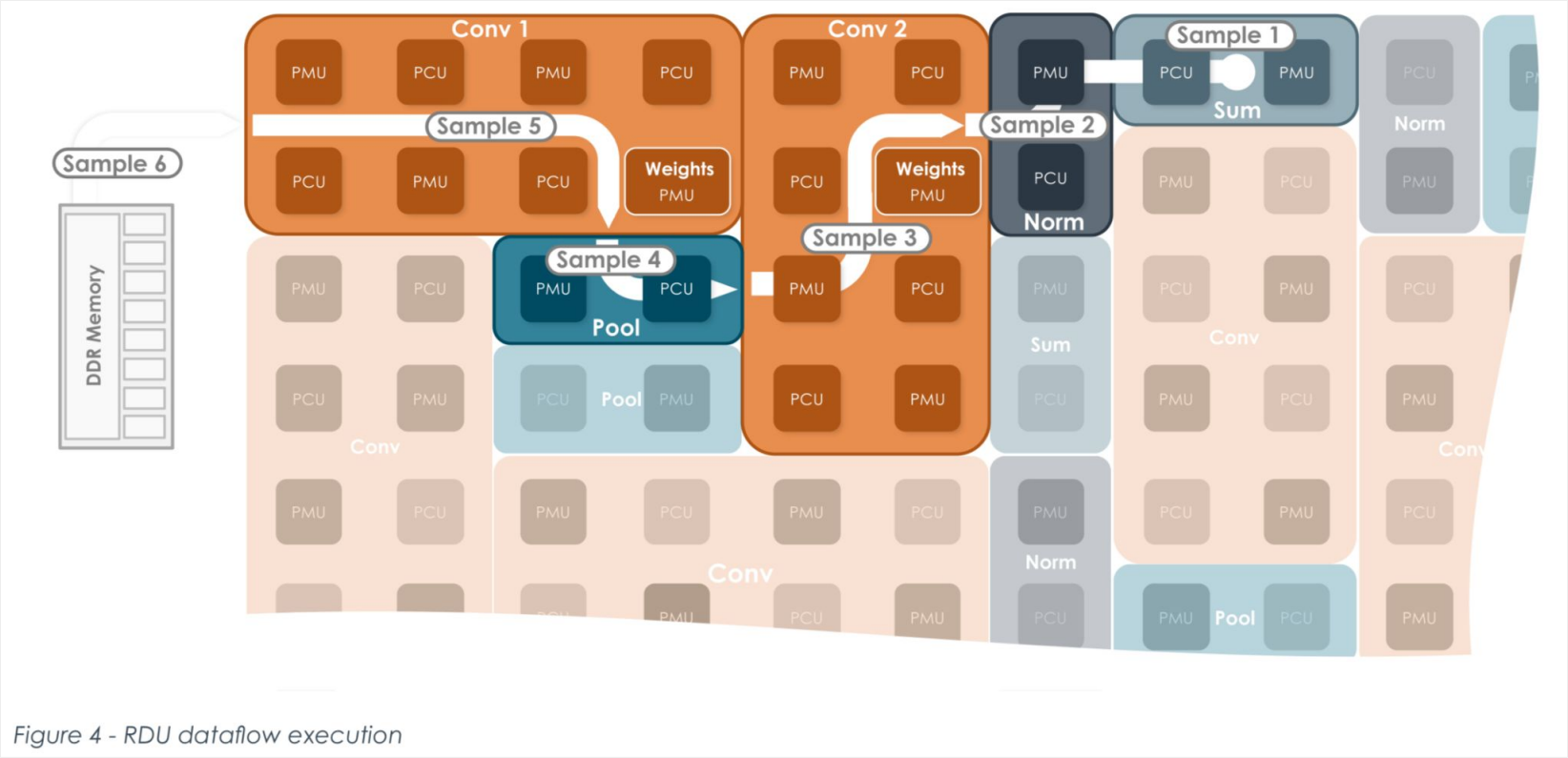
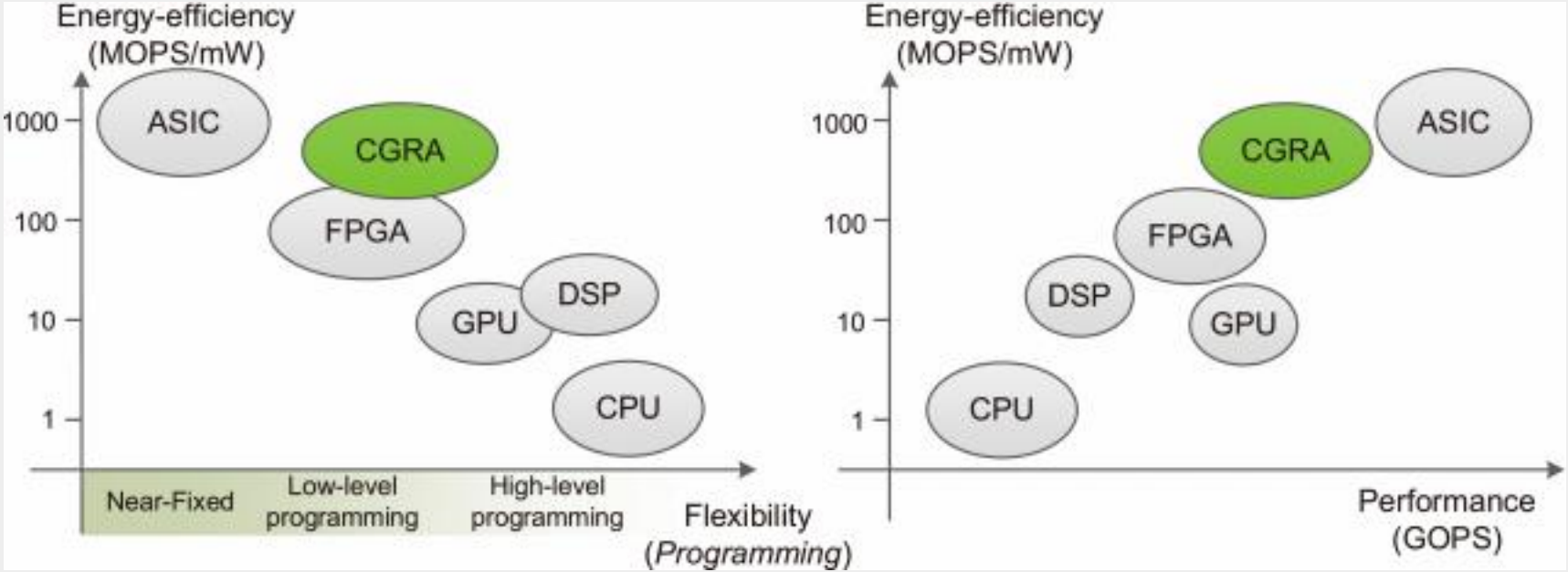


Figure 4 - RDU dataflow execution

Source: https://sambanova.ai/wp-content/uploads/2021/04/SambaNova_Accelerated-Computing-with-a-Reconfigurable-Dataflow-Architecture_Whitepaper_English.pdf

Architectural Comparisons



Source: Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications. ACM Comput. Surv. 52, 6, Article 118 (November 2020), 39 pages. https://doi.org/10.1145/3357375

Architectural Comparisons

Table 1. Comparisons Between CGRAs and Important Computing Fabrics

Architecture	Flexibility	Computation form		Execution mechanism			
		Temporal	Spatial	Reconfiguration time(4)	Configuration-driven	Dataflow-driven	Instruction-driven
CGRA	Domain	√	√	ns-µs	√	√	×
FPGA	General	×(1)	√	ms-s	√	√	×
ASIC	Fixed	×(2)	√	×	× (3)	√	×
In-Order Processor/ VLIW	General	√	×	ns	×	× (5)	√
Out-of-Order Processor	General	√	×	ns	×	√	√
Multicore	General	√	√	ns	×	× (5)	√

Source: Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications. ACM Comput. Surv. 52, 6, Article 118 (November 2020), 39 pages. https://doi.org/10.1145/3357375

Review: Floating Point

Why do we have floating point?

What is easy in floating point?

What is hard in floating point?

Digital Signal Processors as AI Accelerators

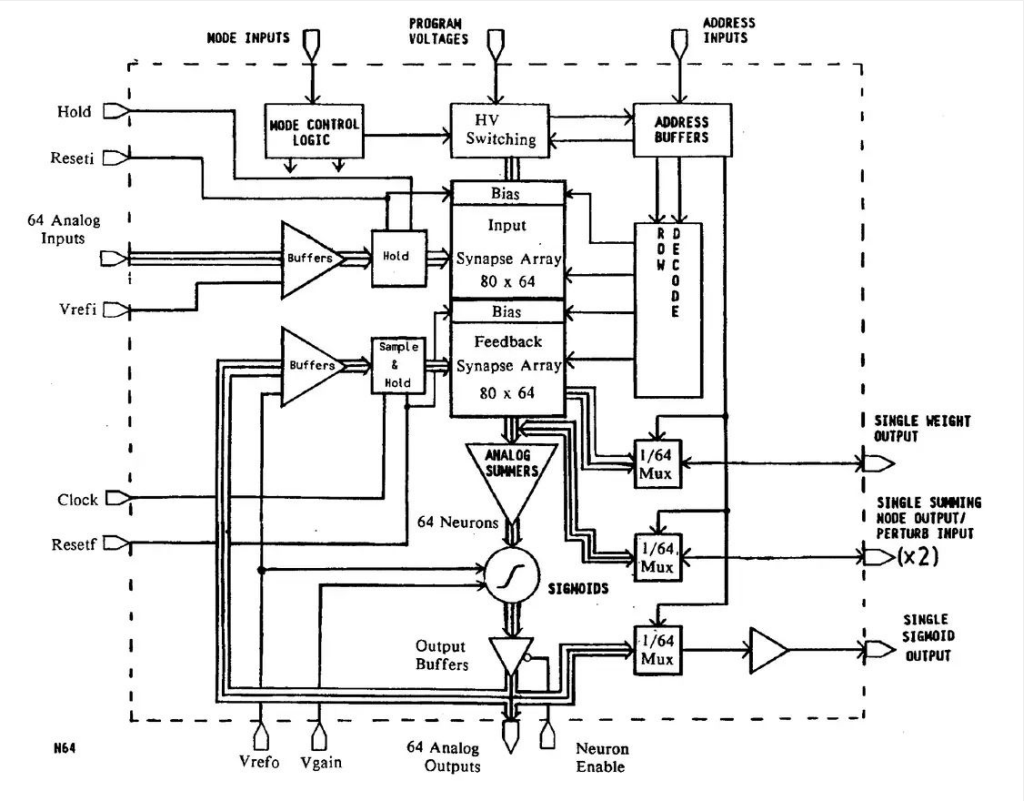
i486 + ATT DSP32C

One of the first chips with 32-bit FP units

12.5 million MACs / s

9760 parameters and 64,660 connections

Intel ETANN: Analog Circuits

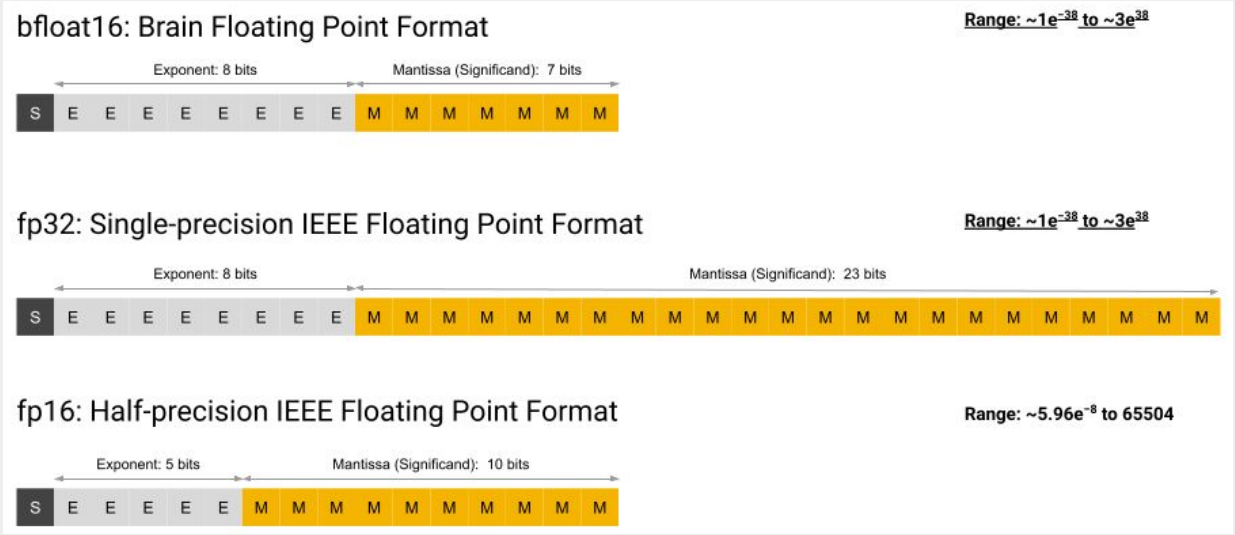


Analog numbers have infinite precision, and the precision loss due to noise is okay.

Image Credit: https://en.wikichip.org/wiki/File:ETANN_block_diagram.png

Brain Floating Point

For weights, the magnitude is often more important than precision.
Small differences in precision can be attributed to training noise, and might be ignored by a dampener.
Idea: use low-space low-precision floats with same range as 32 bit floats



Helpful because FP multipliers scale with the square of the mantissa size

Image Credit: Google

Scratchpads

AKA Software-managed cache
AKA GPU shared memory

A portion of the address space private to some number of cores.

The processor chooses what to put in the scratchpad for low access time.

Good for temporaries and intermediate layers.

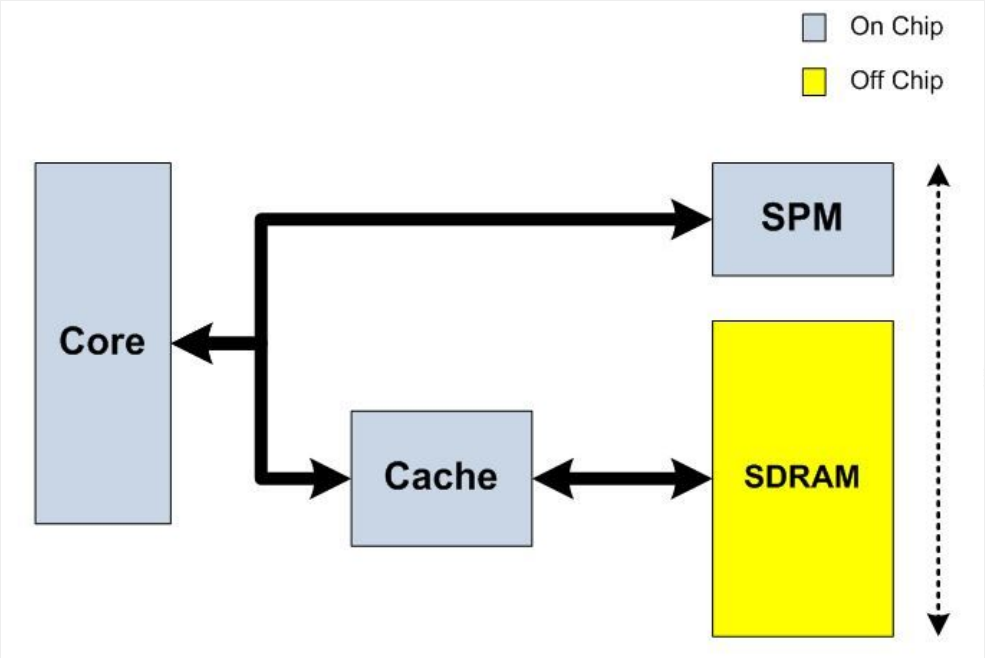


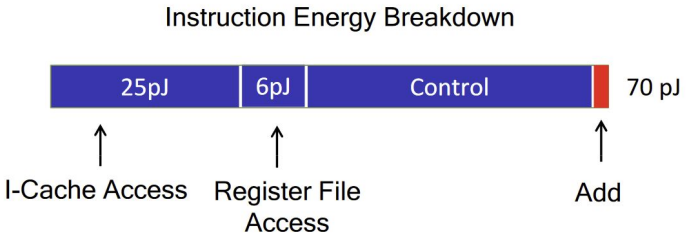
Image Credit: https://commons.wikimedia.org/wiki/File:Scratch-Pad-Memory.jpg

Processing in Memory: SIMD Extended

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ



1. Memory bandwidth is a bottleneck
2. Memory access costs are very great
3. State of the art: near data processing to reduce number of access of off-chip memory

In **PIM**, Main memory modules are manufactured with simple logic elements like adders or multipliers

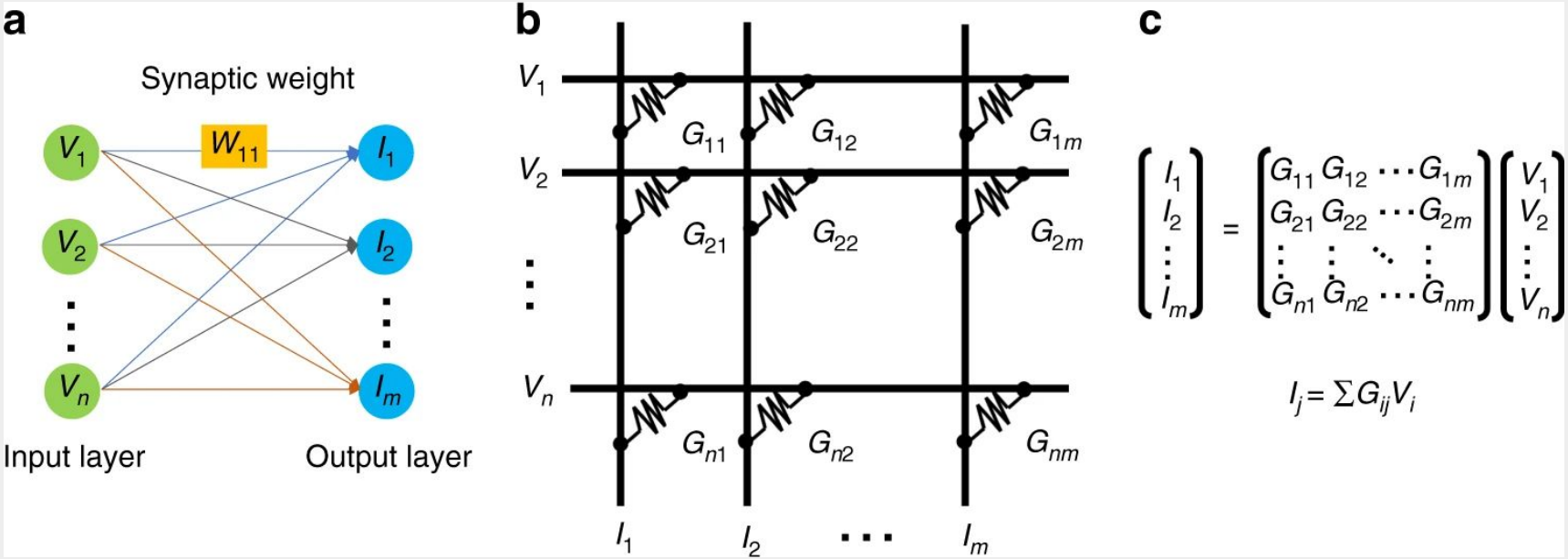
Excellent lecture on PIM from C-M U: https://www.youtube.com/watch?v=r9HbMrb98CQ

Source: Horowitz, M. (2014). Computing's energy problem (and what we can do about it). 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 10–14. https://doi.org/10.1109/ISSCC.2014.6757323

Processing in Memory

- In PIM systems, the main memory modules are manufactured with digital logic elements (like adders or multipliers), so the compute processing is located inside the memory.
- Many processing in memory architectures rely on analog computations
  - Neuromorphic computing: compute dot products in the analog domain
  - Requires special analog-to-digital and digital-to-analog converters
- Design still considered rigid

Neuromorphic, memristive PiM



Source: Gong, N., Idé, T., Kim, S., Boybat, I., Sebastian, A., Narayanan, V., & Ando, T. (2018). Signal and noise extraction from analog memory elements for neuromorphic computing. Nature Communications, 9(1), Article 1. https://doi.org/10.1038/s41467-018-04485-1

Lecture Outline

1. Introduction to Machine Learning
2. Why accelerators?
3. Foundations of accelerator architecture paradigms
4. Examples of accelerator implementations
    a. SIMD/SIMT: Nvidia, Cerebras; MIMD: GraphCore
    b. Systolic array of single-function PEs: TPU, Groq
    c. Tile-based array of independent PUs: GraphCore, Simba
    d. CGRA-based dataflow processing: SambaNova
    e. RISC-based accelerators: Esperanto
    f. Analog and digital-based PiM: FloatPiM and Mystic
5. Next steps for the accelerator design

Many Terms for the Pretty Much the Same Thing

APU: Accelerated Processing Unit (not specific to AI), AMD

DPU: Data Processing Unit, NVidia

DLP: Deep Learning Processor, DianNao

TPU: Tensor Processing Unit, Google

NPU: Neural Processing Unit, ARM

NNP: Neural Network Processor, Intel Nervana

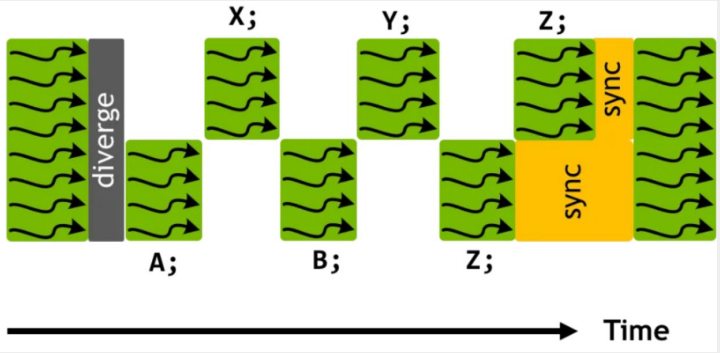
IPU: Intelligence Processing Unit, GraphCore

AIU: Artificial Intelligence Unit, IBM

RDU: Reconfigurable Dataflow Unit, SambaNova

NVIDIA, GPGPU, and SIMT

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
__syncwarp()
```



SIMT: The **S**ame **I**nstruction executes concurrently on **M**ultiple cores/**T**hreads, each on its own portion of data as dictated by its assigned thread ID.

SIMT is still conceptually a multi-threaded c-like programming model that was repurposed for AI, and not designed specifically for AI.

Unable to capture graph semantics, rather than serialized an AI graph to a sequence of instructions, and re-parallelize it so it can run on the parallel hardware

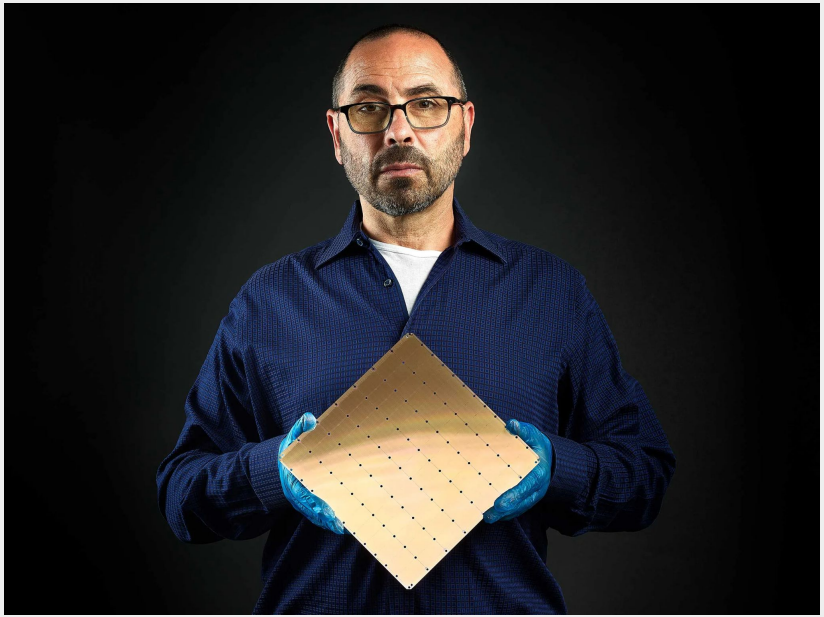
Source: https://developer.nvidia.com/blog/inside-volta/

SIMD scaled up: Cerebras Wafer-scale Engines

Compared to the NVIDIA A100 GPU...

- 56x chip size
- 123x cores
- 1024x on-chip memory
- 12862x memory bandwidth
- 37-40x power consumption

Compiler based around the Linear-Algebra Intermediate Representation



Source: https://spectrum.ieee.org/cerebras-giant-ai-chip-now-has-a-trillions-more-transistors

Google TPU, Systolic Arrays

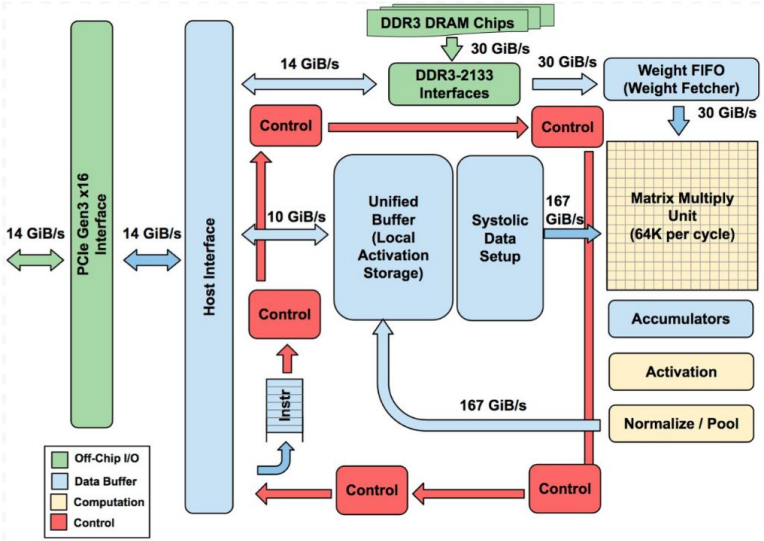


Figure 1. TPU Block Diagram. The main computation part is the yellow Matrix Multiply unit in the upper right hand corner. Its inputs are the blue Weight FIFO and the blue Unified Buffer (UB) and its output is the blue Accumulators (Acc). The yellow Activation Unit performs the nonlinear functions on the Acc, which go to the UB.

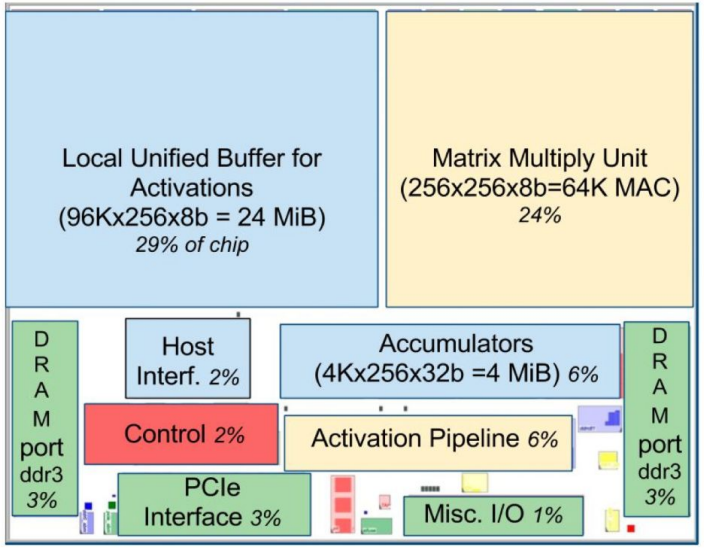
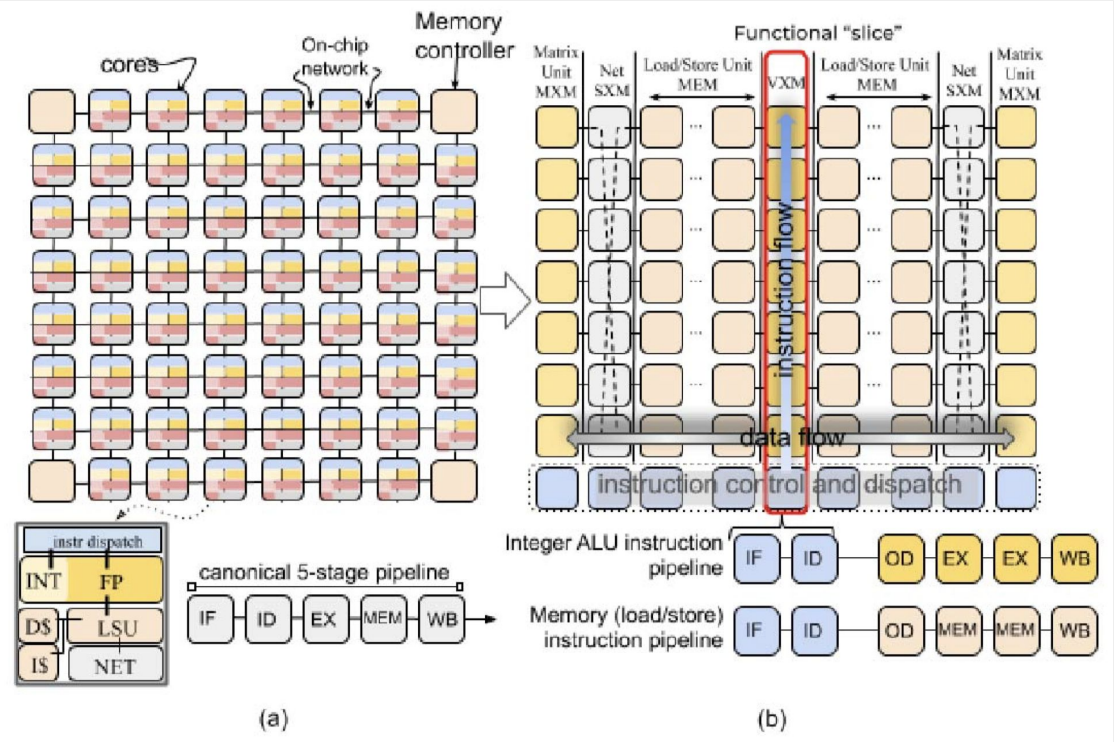


Figure 2. Floor Plan of TPU die. The shading follows Figure 1. The light (blue) data buffers are 37% of the die, the light (yellow) compute is 30%, the medium (green) I/O is 10%, and the dark (red) control is just 2%. Control is much larger (and much more difficult to design) in a CPU or GPU

Source: Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., ... Yoon, D. H. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit (arXiv:1704.04760). arXiv. http://arxiv.org/abs/1704.04760

Groq TSP, Systolic Arrays



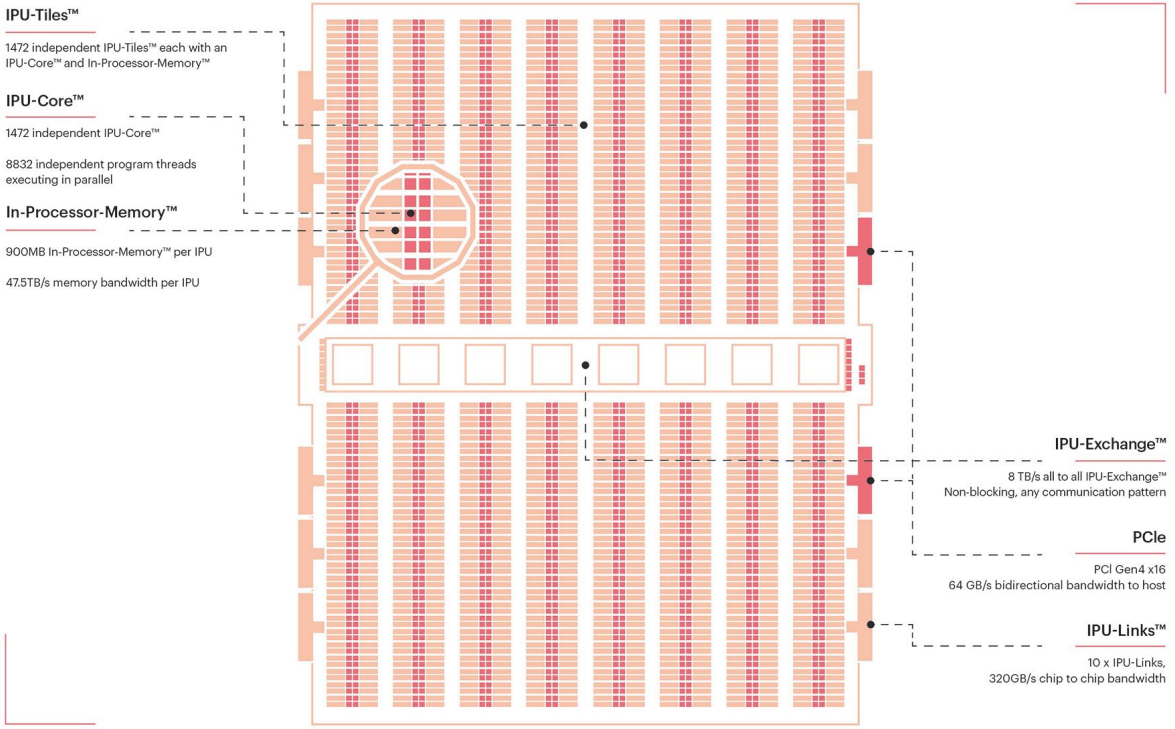
- Achieve local functional homogeneity but chip-wide, global heterogeneity
- Separate IF/ID from EX/WB
- Decouple memory subsystem from FUs
- Each functional slice implements 20-stage vector pipeline
- Each tile produce 16 elements of the 320-element max vector length

Source: Abts, D., Ross, J., Sparling, J., Wong-VanHaren, M., Baker, M., Hawkins, T., Bell, A., Thompson, J., Kahsai, T., Kimmell, G., Hwang, J., Leslie-Hurd, R., Bye, M., Creswick, E. R., Boyd, M., Venigalla, M., Laforge, E., Purdy, J., Kamath, P., … Kurtz, B. (2020). Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads. 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 145–158. https://doi.org/10.1109/ISCA45697.2020.00023

MIMD: GraphCore's Intelligent Processing Unit

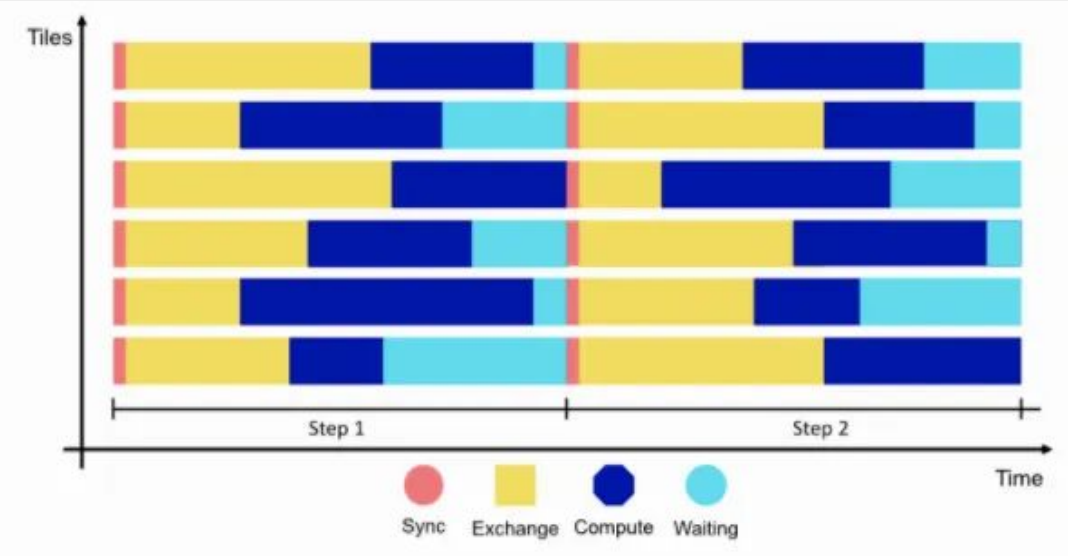
In-house software stack and multi-threaded dataflow execution

- Poplar compiler
- "Tiled" many-core design
- Bulk synchronous parallel execution model
  - Compute, Sync, Exchange



Source: https://www.graphcore.ai/products/ipu

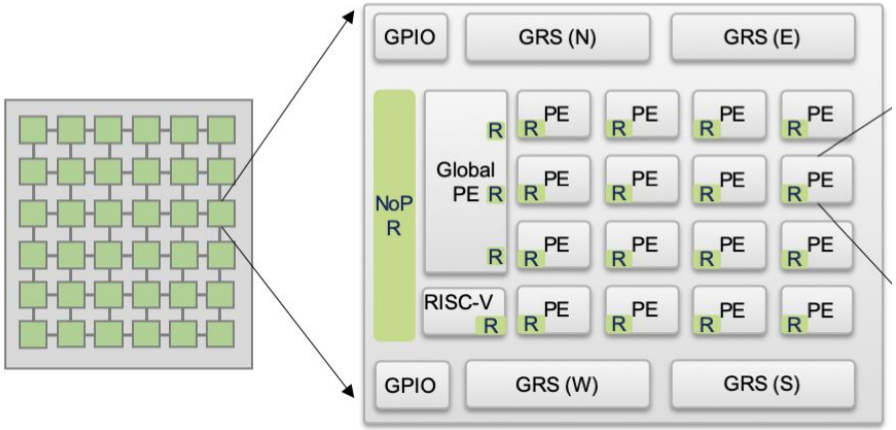
MIMD: GraphCore's Intelligent Processing Unit



Bulk synchronous parallel execution model: compute, sync, exchange
Facilitated by in-house software stack of compilers

Source: https://www.graphcore.ai/products/ipu

Chiplet-based MIMD: Simba



(a) Simba Package (b) Simba Chiplet

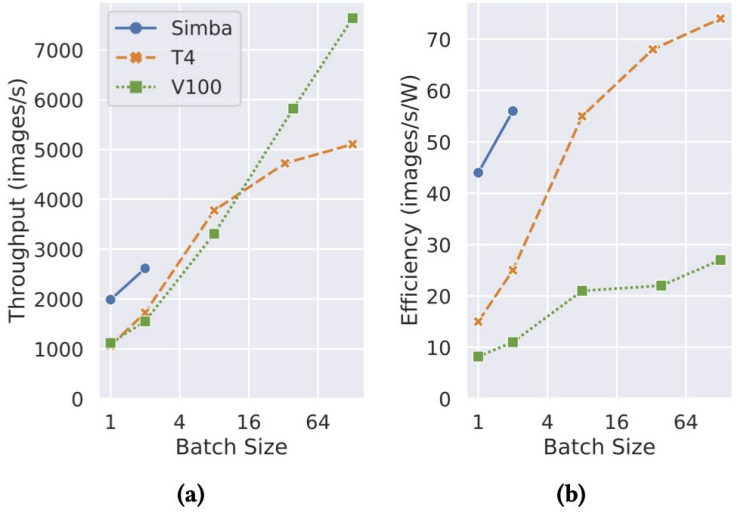
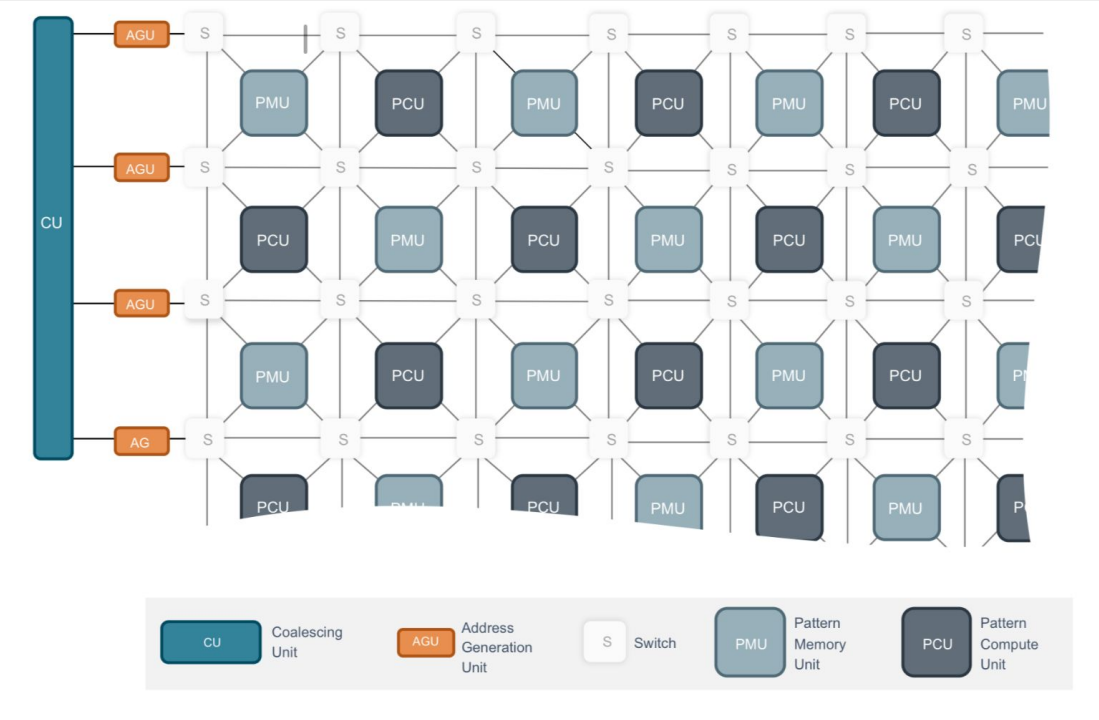


Figure 11: Throughput and efficiency of Simba, V100, and T4 running ResNet-50 with different batch sizes.

Source: Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52). Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

CGRA, Dataflow graph-based hardware



SambaNova Reconfigurable Dataflow Unit

- Pattern compute unit, pattern memory unit, switching fabric
- Network-on-chip: Address generator unit, coalescing unit
- SambaFlow compiler, dataflow graph as IR
- Compiler completes placing and routing by mapping DF graph onto physical RDU
- Israel-based Hailo does similar things with dataflow compilers

Source:
https://sambanova.ai/wp-content/uploads/2021/04/SambaNova_Accelerated-Computing-with-a-Reconfigurable-Dataflow-Architecture_Whitepaper_English.pdf

RISC-based accelerators, TLP = DLP?

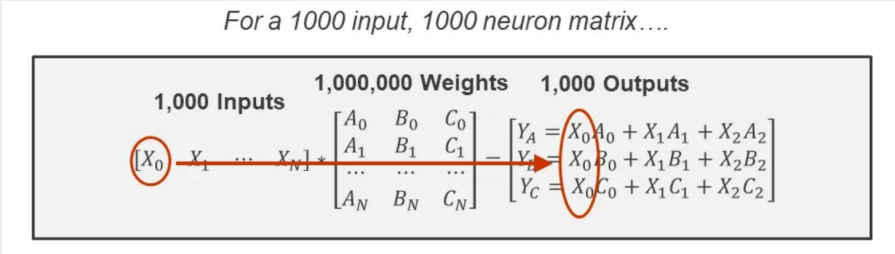
- **Esperanto** ET-SoC1: RISC-V based heterogeneous many-core chip
  - Mixture of 1,088 low-voltage cores for vector computing and 4 high-power general purpose cores
  - Convert data-level to thread-level parallelism with efficient RISC cores
  - Datacenter inference workloads
  - **20w power budget**

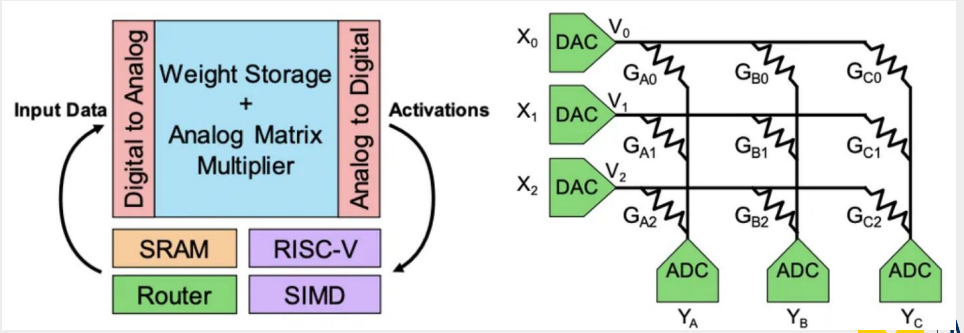
	Power/core	Frequency	Voltage	Cdynamic
Generic x86 Server core (165W for 24 cores)	7 W	3 GHz	0.850v	2.2nF
10mW ET-Minion core (~10W for 1K cores)	0.01 W	1 GHz	0.425v	0.04nF
Reductions needed to hit goals	~700x	3x	4x	58x
		Easy	*Hard Circuit/SRAM*	*Very Hard Architecture*

Source: https://www.esperanto.ai/wp-content/uploads/2021/08/HC2021.Esperanto.Ditzel.Final_.pdf

Mythic: Neuromorphic, off-chip computing

- Put input data in on-chip memory, since they are accessed more during MAC
- Put weights in off-chip memory and compute modules
- Analog Matrix Processing products for edge devices





Source: https://medium.com/mythic-ai/mythic-hot-chips-2018-637dfb9e38b7

Digital PiM in FloatPIM

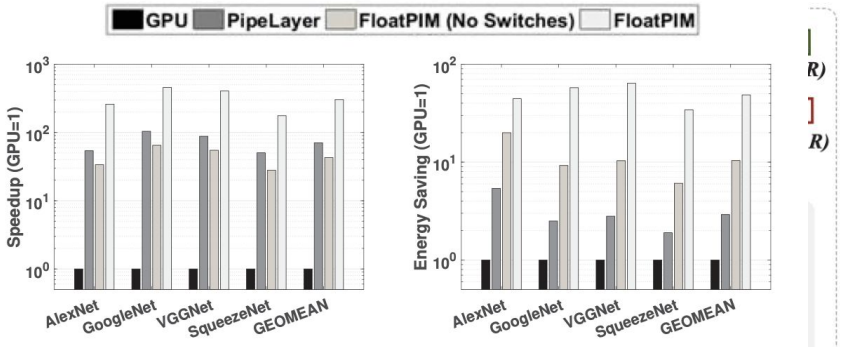


Figure 12: FloatPIM efficiency during training.

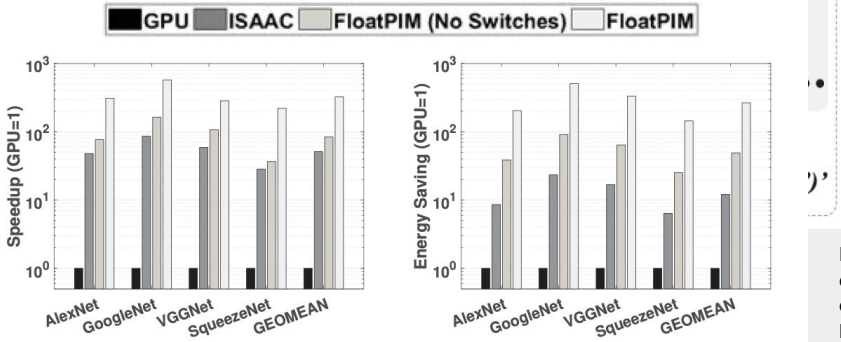


Figure 13: FloatPIM efficiency during the testing.

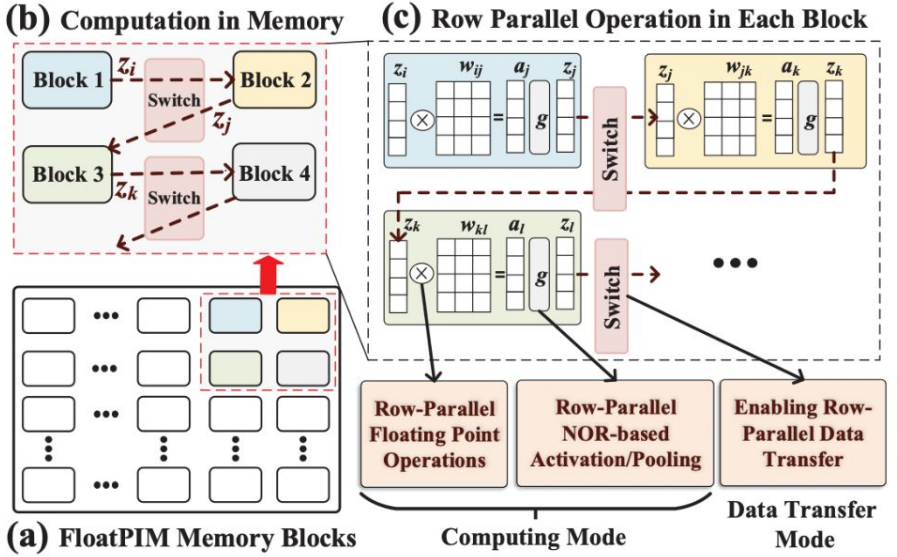


Figure 3: Overview of FloatPIM.

Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. 2019. FloatPIM: in-memory acceleration of deep neural network training with high precision. In Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19). Association for Computing Machinery, New York, NY, USA, 802–815. https://doi.org/10.1145/3307650.3322237

Lecture Outline

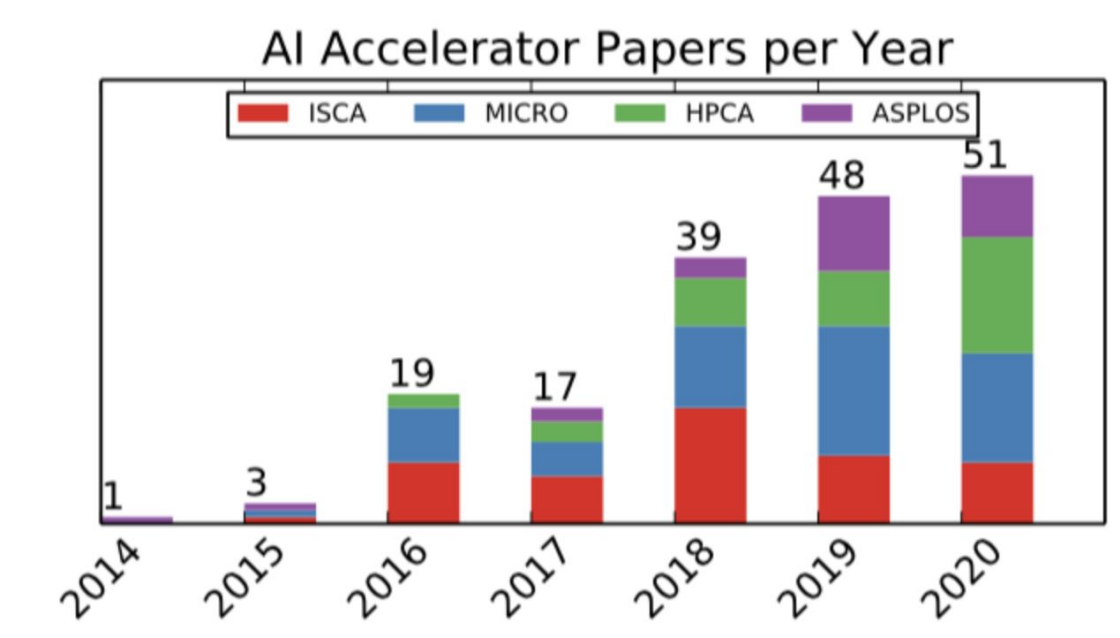
1. Introduction to Machine Learning
2. Why accelerators?
3. Foundations of accelerator architecture paradigms
4. Examples of accelerator implementations
5. Next steps for accelerator design
    a. Academia vs. industry in hardware design
    b. Boundaries of accelerated domains

Academia vs. Industry…

However, on the hardware side, there seems to be some disconnect between academia and industry. While in the past five years, hundreds of AI accelerator papers were published in leading venues like ISCA, HPCA, MICRO, DAC, ISSCC, and VLSI, only a handful of those made it into actual products. Most companies' core architectures are based on "stable" ideas that have already been experimented with within different contexts. Notably, companies combine VLIW architectures with systolic arrays (Google, Groq, Habana) or support the dataflow execution of compiler-lowered computational graphs (Wave Computing, SambaNova, SimpleMachines, Graphcore, Hailo, and possibly, TensTorrent and Cerebras.)

As you might have noticed, almost every overviewed AI acceleration solution started from an academic idea that was decades old: systolic arrays were introduced in 1978, VLIW architectures in 1983, the concept of dataflow programming date back to 1975, early processing-in-memory works were presented in the 1970s. One reason for this "conservatism" is that in such a competitive and fast-moving landscape, bold new architectural ideas are too risky, and people favor well-rounded ideas that have matured and been implemented in other contexts.

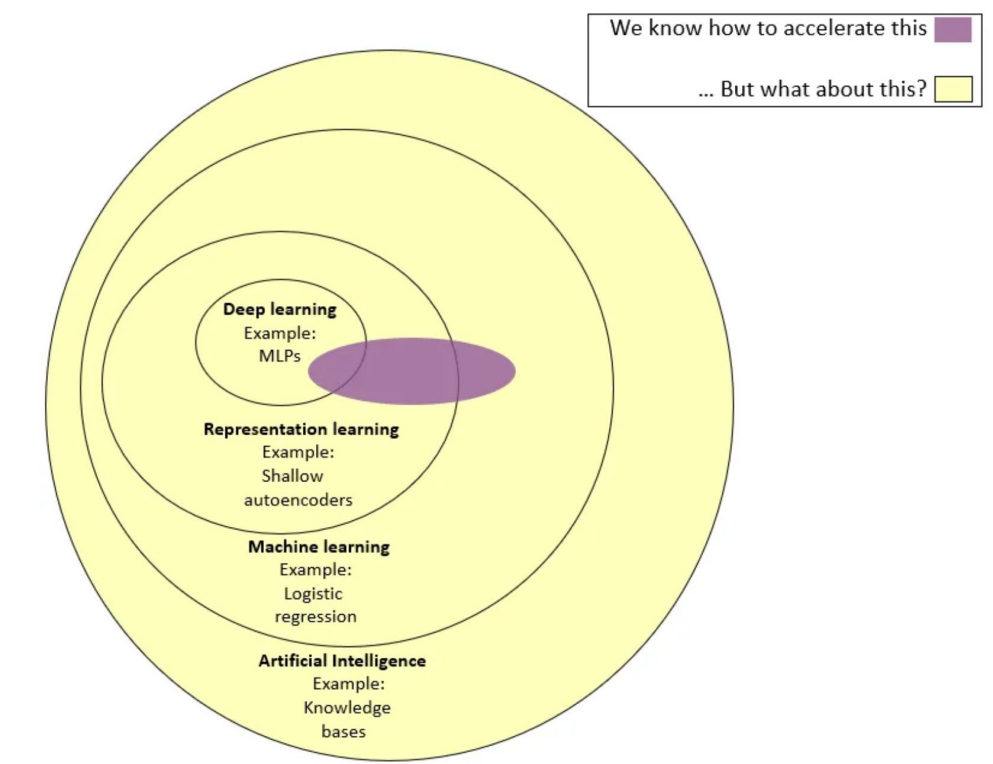
Academia vs. Industry



- The number of novel accelerator architectures in academic literature is exploding!
- Yet commercialized implementations almost all draw from decades old paradigms…
- How can we increase the penetration of innovations in academia?

Source: https://web.archive.org/web/20220331021722/https://birenresearch.github.io/AIChip_Paper_List/

What domain are we actually accelerating?



- Is the advancement of AI/ML constrained by hardware or software?
- What if we want to go beyond the tensor-intensive ML models that are increasingly optimized by state-of-the-art accelerator hardware?
- How can we justify the performance potential of novel models without the optimized hardware to run it? *AlexNet got very lucky in retrospect…*

Source: https://medium.com/@adi.fu7/ushering-in-the-next-wave-of-ai-acceleration-a7a14a1803d1

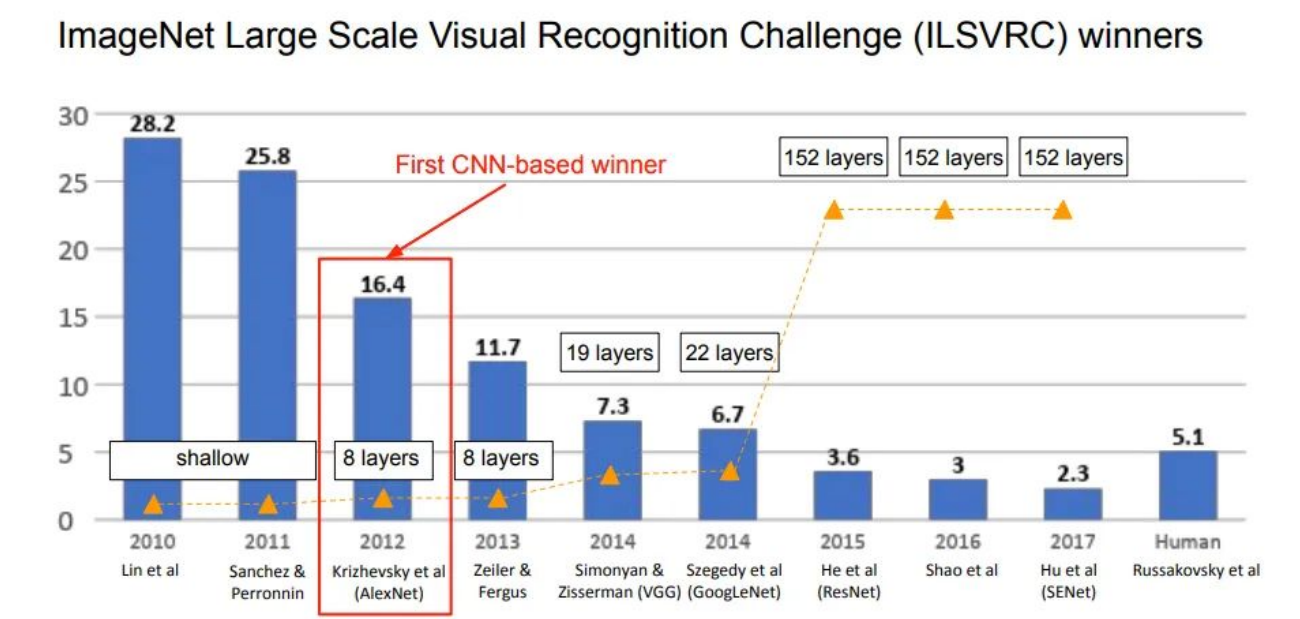
MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

AI in Recent Arch Conferences

Exercise

Thanks!

The Rise of AI/ML



While algorithmic process is important, it was the use of specialized GPU hardware that enabled our ability to train deeper and larger networks at reasonable times, thus improving the overall network's accuracy over the current state of the art.

Semantic gap between User/SW/HW

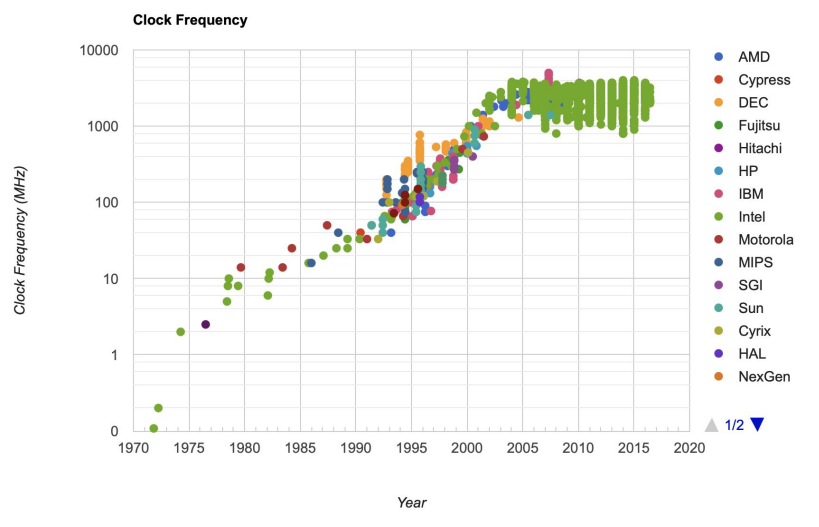
Aka why should I care about hardware for AI?

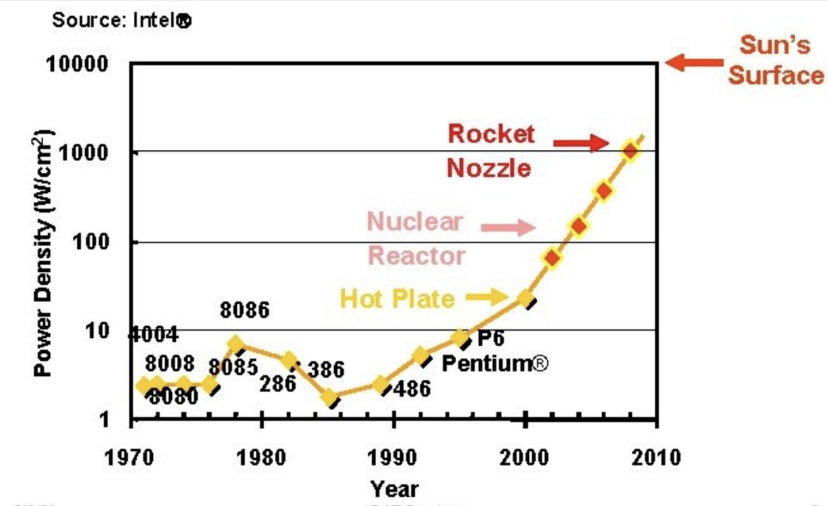
- We're not getting ideal "out-of-the-box" performance for an arbitrary AI model for arbitrary hardware to execute it
- Abundant ways to express complex computation in hardware. Need to establish an understanding of how all the different layers of the HW/SW stack interact.
- Deep learning landscape is rapidly changing, new software released daily, but it takes two years to tape out new processor
- *This part is actually pretty poorly organized, but I think we can do a better job…*

User/SW/HW

- Accelerators are not purely HW-driven
- Much of the AI accelerator industry's focus has been around building robust and sophisticated software libraries and compiler toolchains

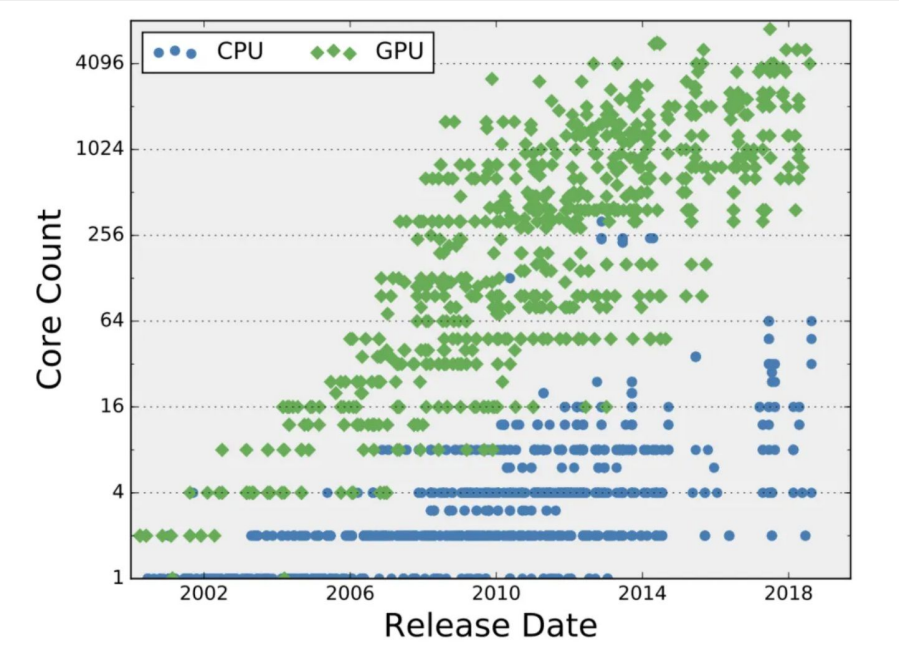
Frequency scaling of processors





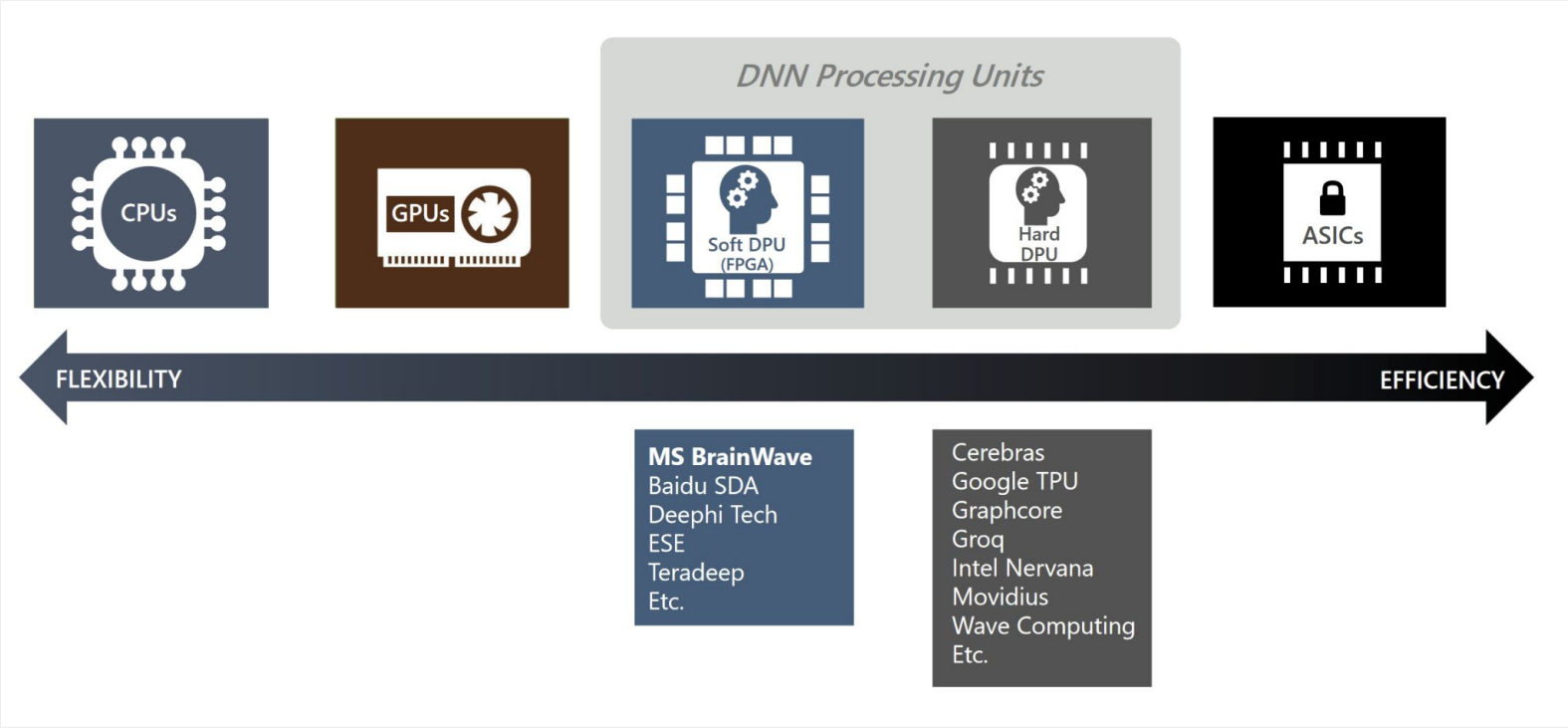
1. Moore's dead
2. Dennard's dead (though not quite as literally…)
3.

Multicore processor design



1. Thread-level parallelism on CPUs
2. Data-level parallelism on GPUs: Historically, GPUs focused on graphic applications, since graphic images (for example in a video) consist of thousands of pixels, which can be processed independently with a series of simple and predetermined computations.

Silicon alternatives for DNNs



Souce:
https://www.microsoft.com/en-us/research/uploads/prod/2017/08/HC29.22622-Brainwave-Datacenter-Chung-Microsoft-2017_08_11_2017.compressed.pdf

Instruction Set Architectures

- Given an existing reduced instruction set architecture (and potentially processing cores), it is possible to reduce it even further by supporting only the subset of instructions needed in the targeted application domain, in our context — AI. Domain-specific ISAs further simplify the processing cores and hardware/software interface to achieve an efficient accelerator design.
- Some architectures extend CISC's notion of combining multiple operations into a single complex instruction. These architectures are known as Very-Long Instruction Word (VLIW)
    - VLIW architectures consist of a heterogenous datapath array of arithmetic and memory units. The heterogeneity stems from the differences in timing and supported functionality of each unit.
    - the cost of orchestrating the processor datapath via instructions is significantly reduced
    - we need to guarantee that the workload is getting balanced between the various units in the datapath to avoid having underutilized resources
    - Complex static scheduling by compiler