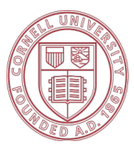
4950/6950 Fall 2026
Special Topics in ECE: AI Hardware

Hardware Specialization, Part 1



Cornell University

CSL

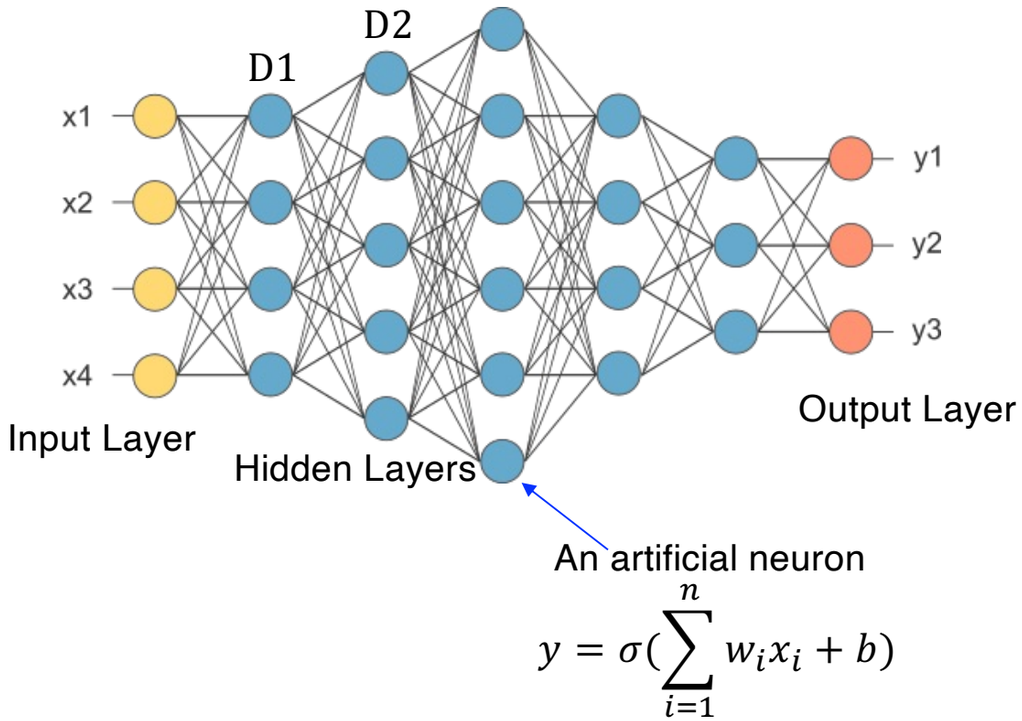
Agenda

- More on CNNs

- Motivation for hardware specialization
  - Main sources of inefficiency in general-purpose computing
  - Principles for improving energy efficiency
  - Essential hardware specialization techniques

Revisiting Multi-Layer Perceptron (MLP)

- In a fully connected layer, compute is directly proportional to the number of weights (i.e., # of edges connecting connecting the input and output neurons)



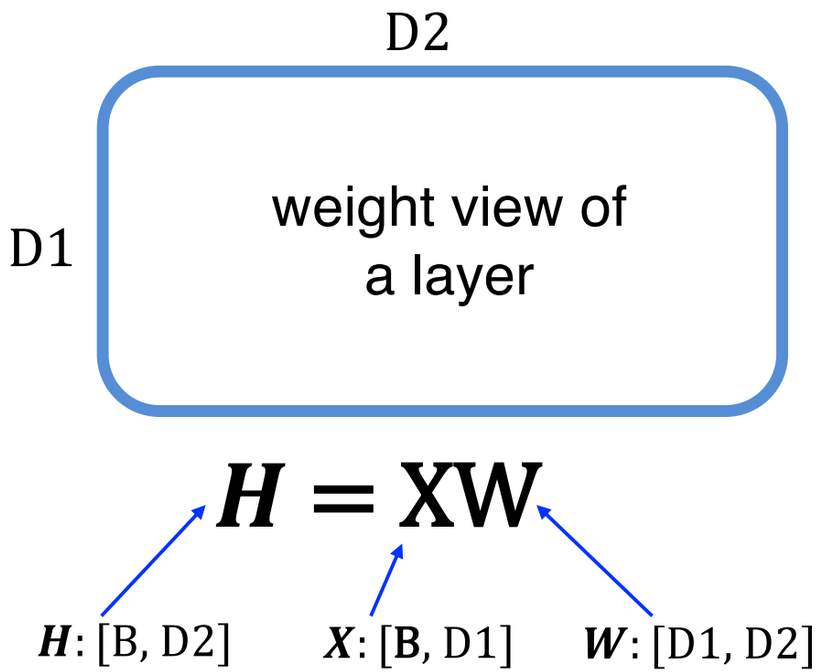
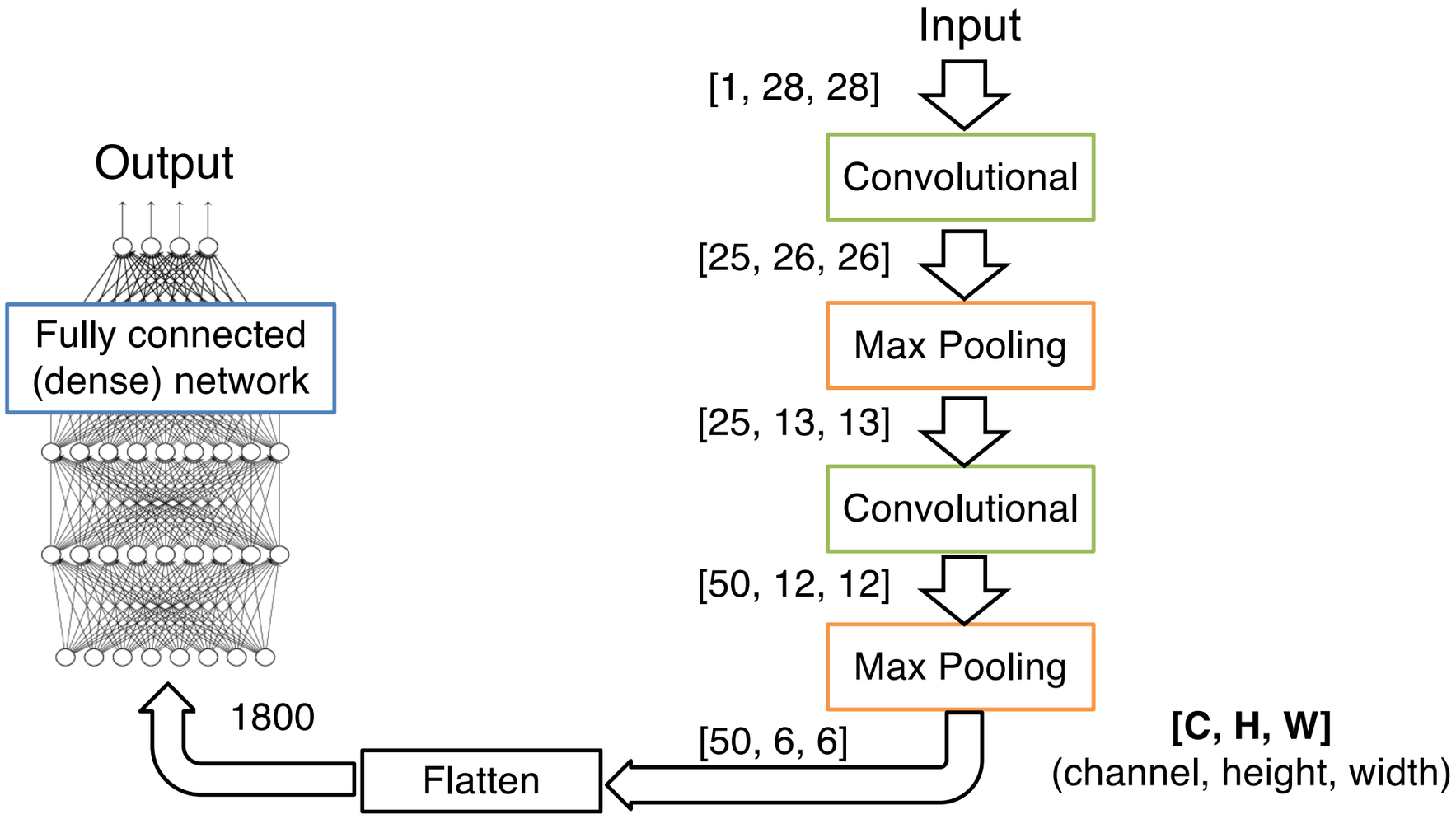
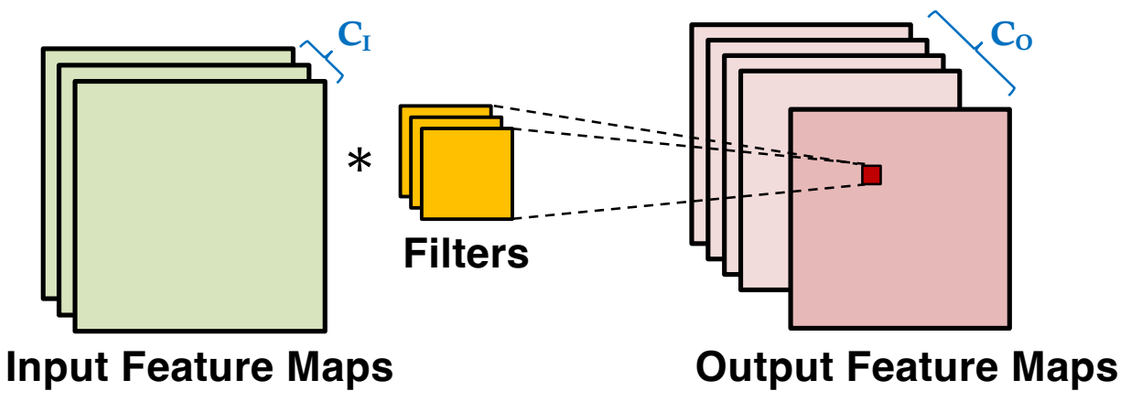


Image credit: http://www.opennn.net/

A Small CNN (for Digit Recognition)

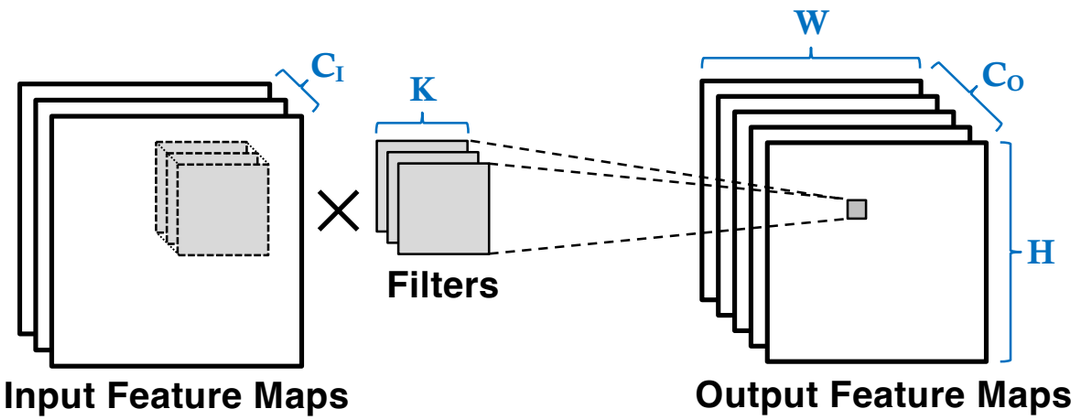


The Convolutional (Conv) Layer



- C_I input and C_O output **feature maps**
  - Also called input and output **channels**
- Each output map uses C_I filters, 1 per input map
- $C_I \times C_O$ total filters

Peusdo-Code of the Conv Layer




```
1 for row in range(H) :
2     for col in range(W) :
3         for to in range(C_O) :
4             for ti in range(C_I) :
5                 for ki in range(K) :
6                     for kj in range(K) :
                          output_fm[to][row][col] += (
                          weights[to][ti][ki][kj] *
                          input_fm[ti][S*row+ki][S*col+kj]
                          )
```

Loop over output pixels

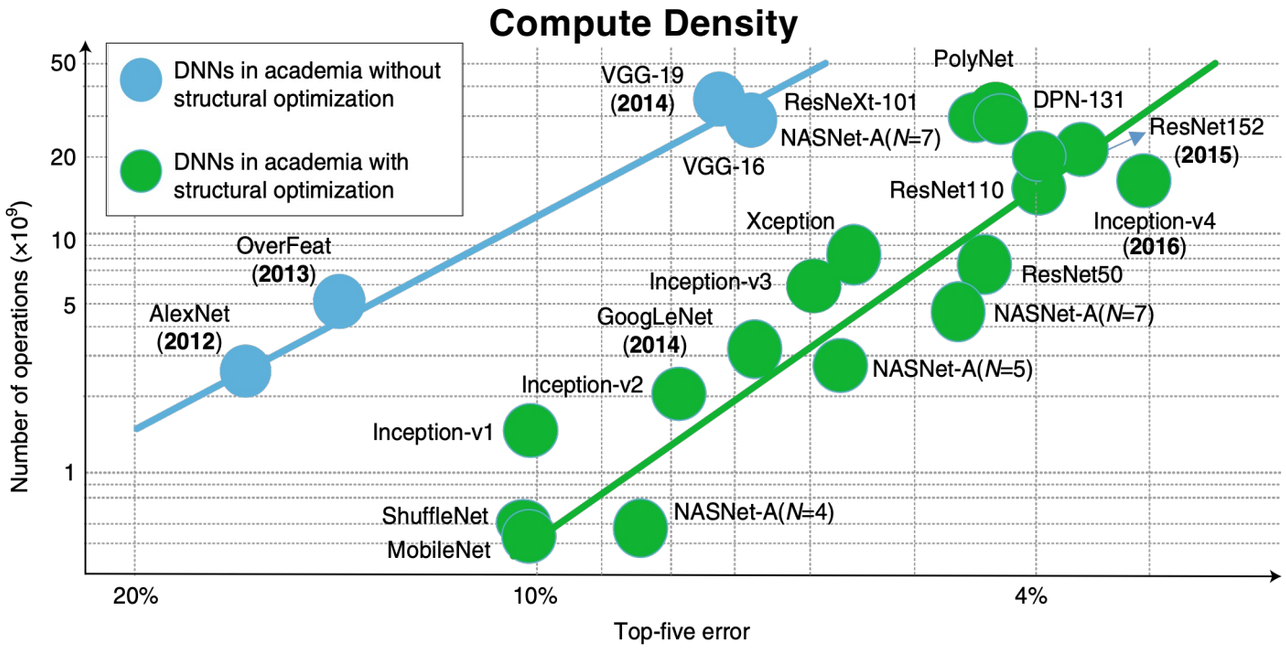
Loop over output feature maps

Loop over input feature maps

Loop over filter pixels

S is the stride

Modern CNN Models are Computationally Expensive

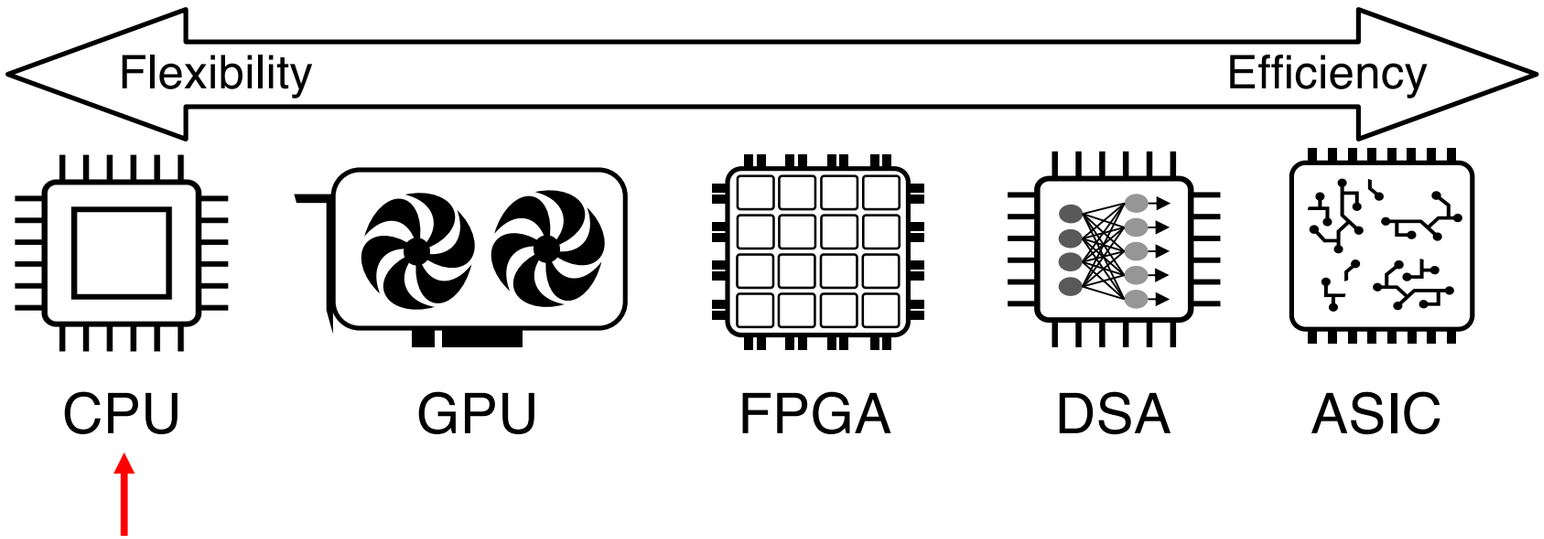


Source: X. Xu, et al. **Scaling for Edge Inference of Neural Networks**. *Nature Electronics*, vol 1, Apr 2018.

For example, ResNet50, a 70-layer CNN model performs 7.7 billion operations required to classify one image

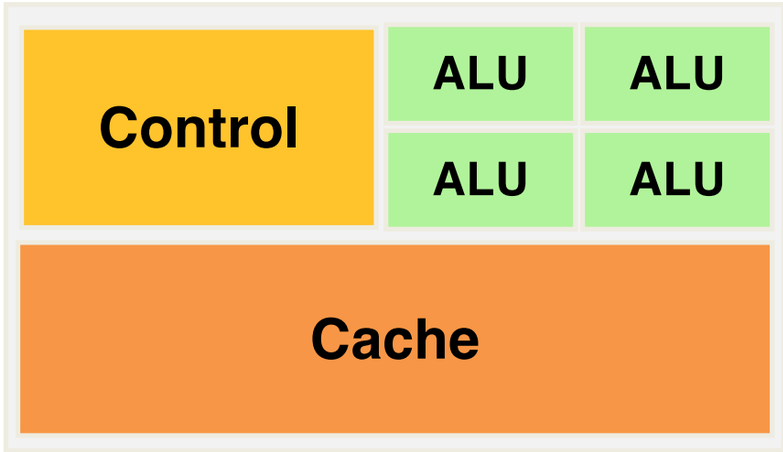
Era of Hardware Specialization

To improve performance and energy efficiency, modern computer systems increasingly integrate **special-purpose hardware accelerators** customized for specific applications or domains of applications



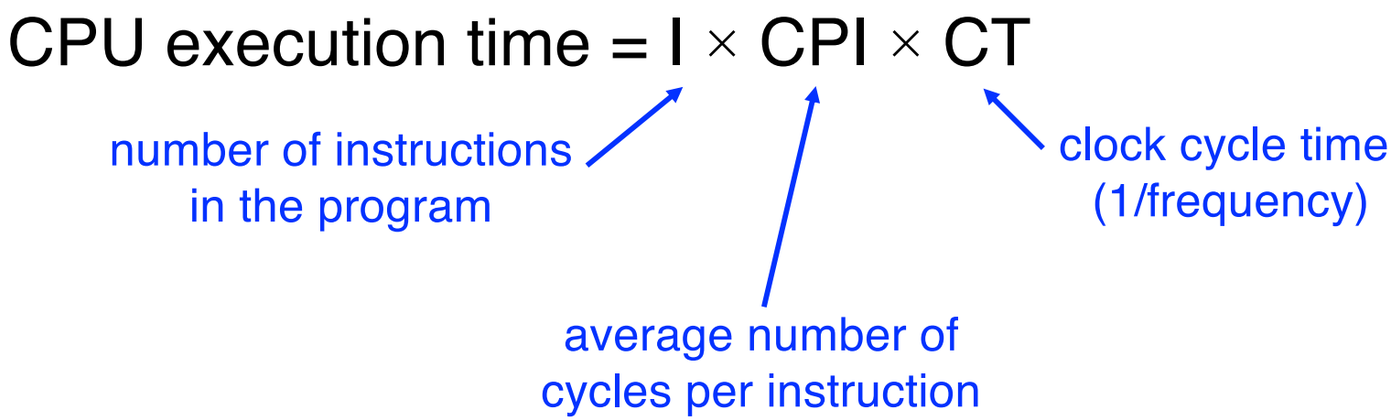
CPU Core Architecture

- Core = limited number of compute units + large caches + complex control
  - Scalar & vector instructions: Backward compatible ISA
  - Complex control logic: decoding, hazard detection, exception handling, etc.



- Mainly optimized to **reduce latency of running serial code**
  - Shallow pipelines (< 30 stages)
  - Superscalar, OOO, speculative execution, branch prediction, prefetching, etc.
  - Low throughput, even with multithreading

Iron law of Processor Performance

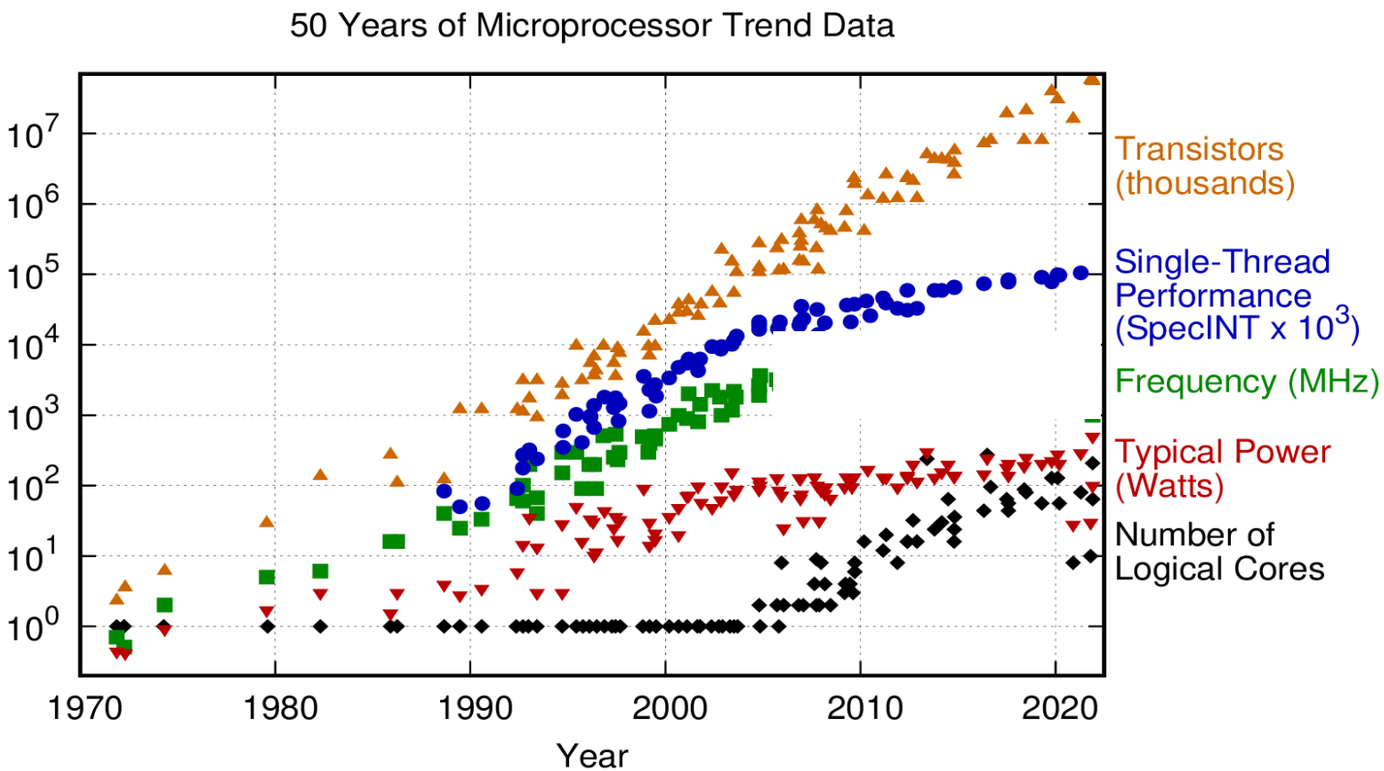


Frequency is no longer improving with new technology nodes!

Classical Technology Scaling (pre-2005)

Transistor counts doubled every ~2 years (**Moore's Law**)

Clock frequencies also rose exponentially (**Dennard scaling**)

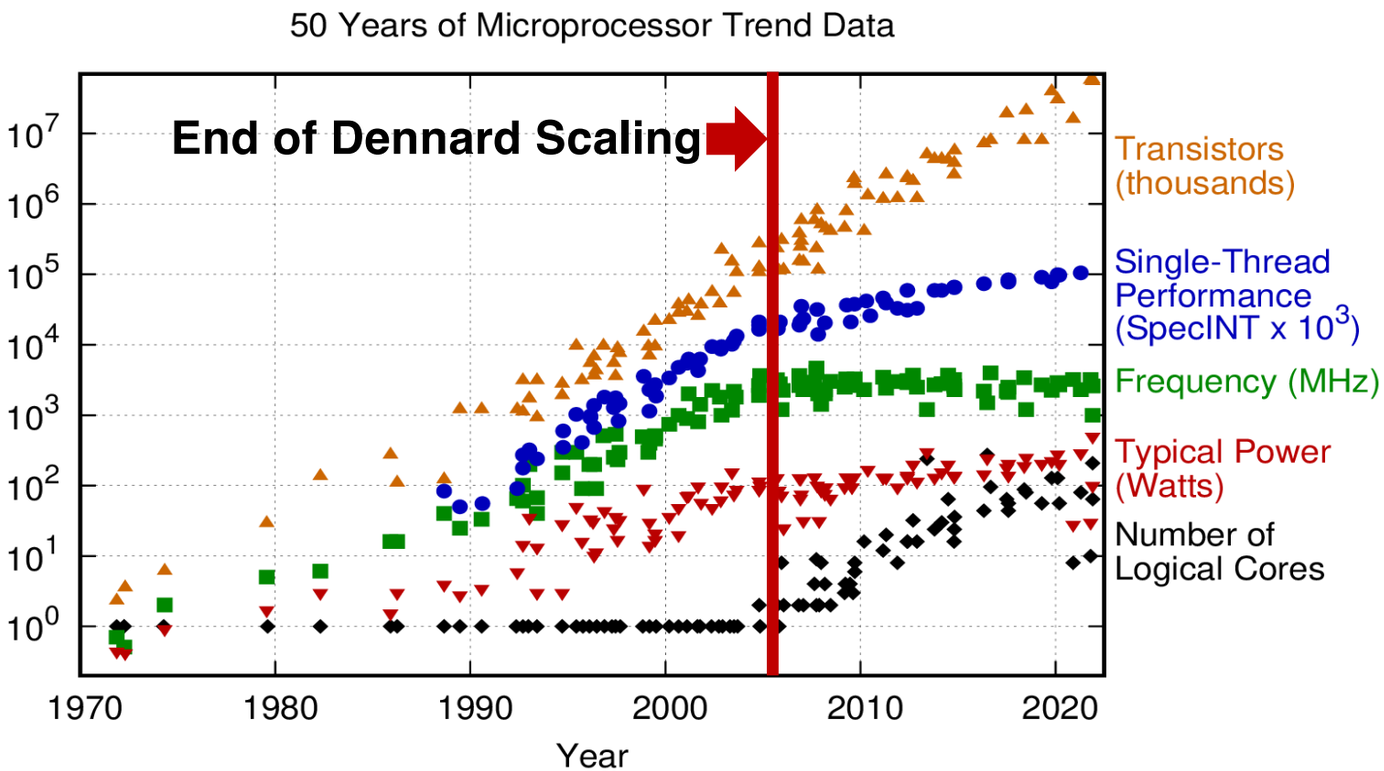


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

End of “Cheap” Technology Scaling (post-2005)

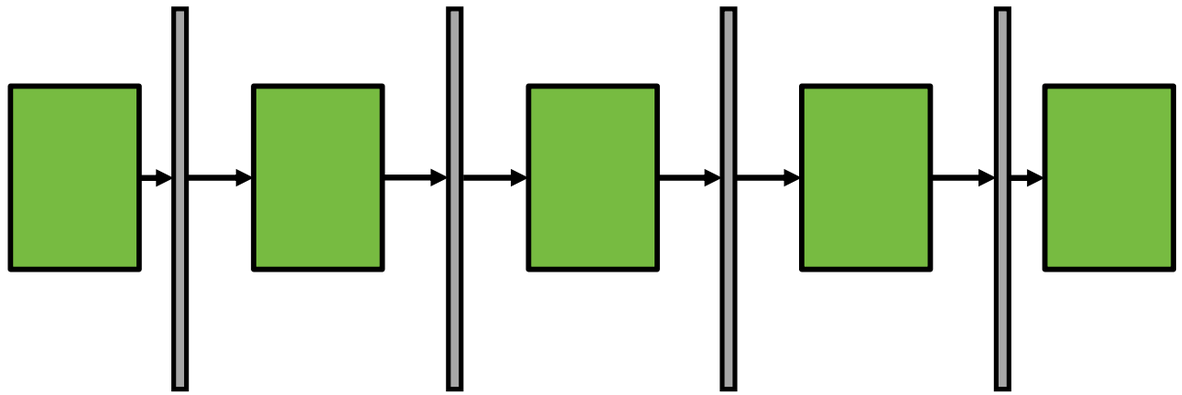
Transistor counts continue to scale

Frequency scaling plateaued as power becomes a limiting factor => led to multicore & greater emphasis on energy efficiency

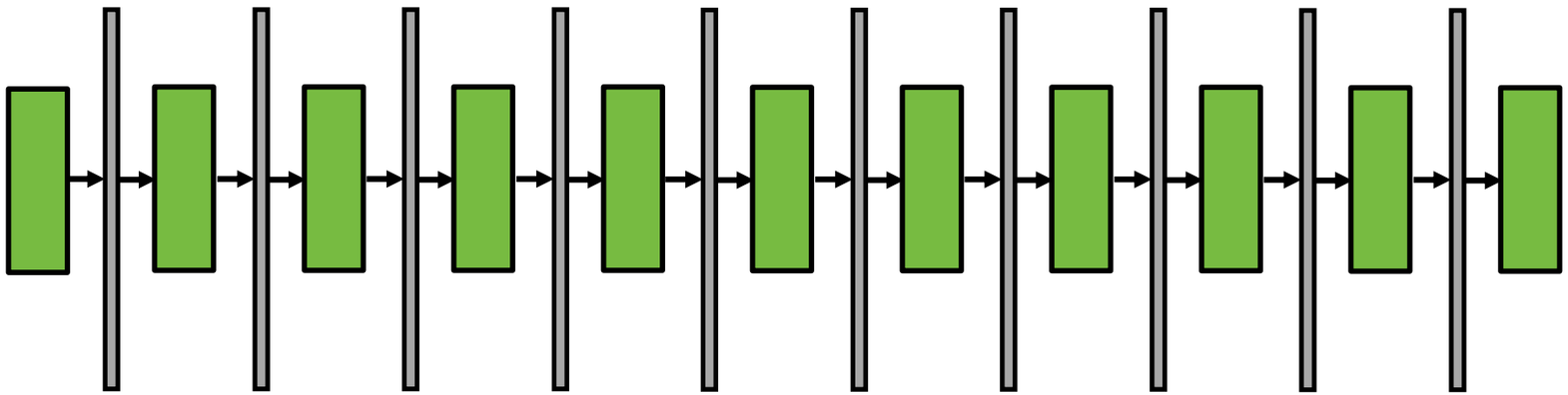


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

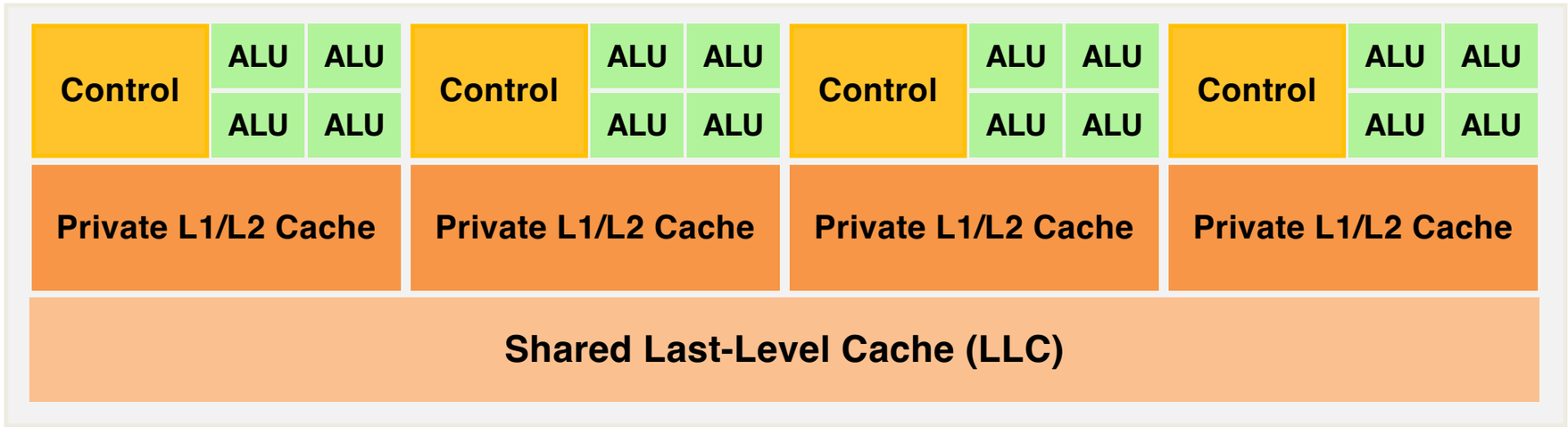
Discussion



Shallow vs. Deep Pipelining

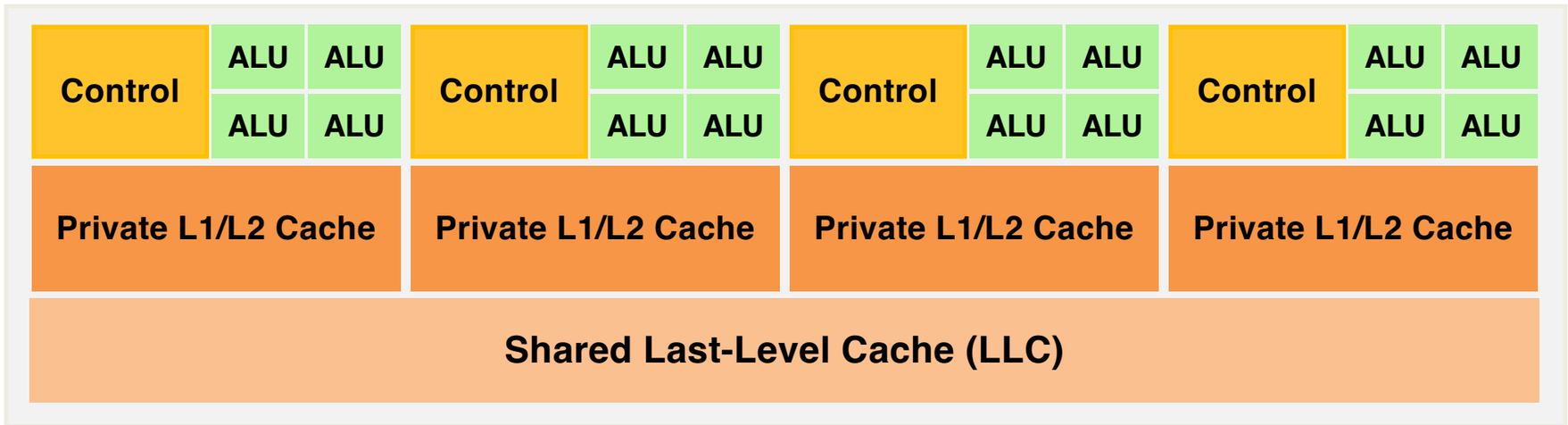


Multi-Core Architecture



- Complex cores spend substantial energy extracting diminishing instruction-level parallelism (ILP) through wide issue, aggressive speculation, etc.
- Multiple simpler cores can exploit thread-level parallelism (TLP) with less energy per operation
- Performance growth shifted toward parallel software that can divide work across cores

Multi-Core Architecture

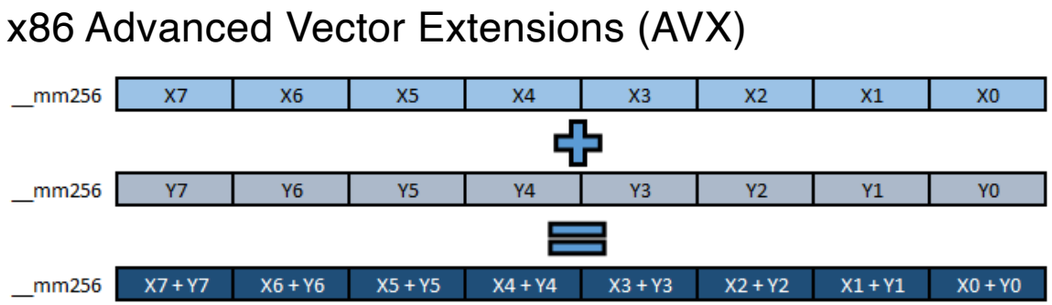


With 4 cores, should we expect a 4x speedup on an arbitrary application?

Data-Level Parallelism (DLP)

- In the following code, the 8 additions in the loop can be replaced by a single vector addition instruction, via **Single Instruction Multiple Data (SIMD)**

```
for i in range(8):
    Z[i] = X[i] + Y[i]
```



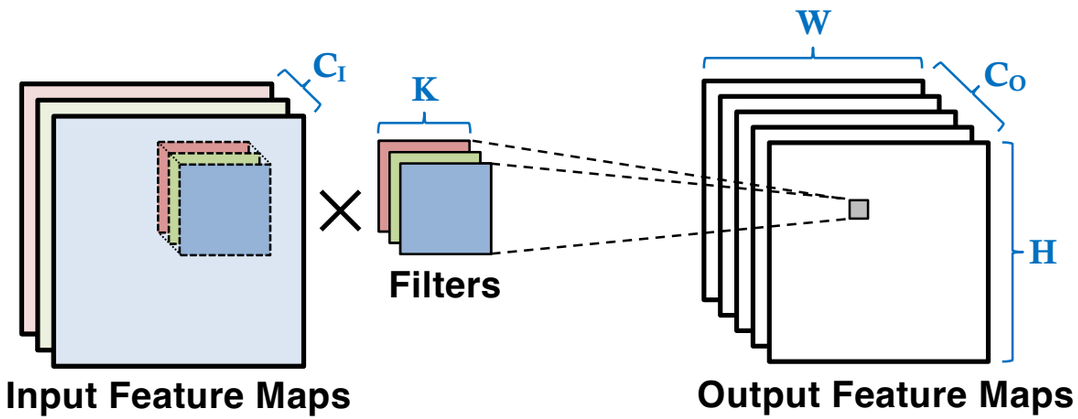
- Modern CPUs include SIMD units to improve both performance and efficiency
  - Well suited for data-parallel operations in ML, DSP, scientific computing
  - Common SIMD extensions (explicit in ISA): x86 AVX, ARM NEON/SVE, and RISC-V Vector; SIMD is also implicitly used in GPUs

Image credit: codingame.com/playgrounds/283/sse-avx-vectorization/what-is-sse-and-avx

DLP in AI Workloads

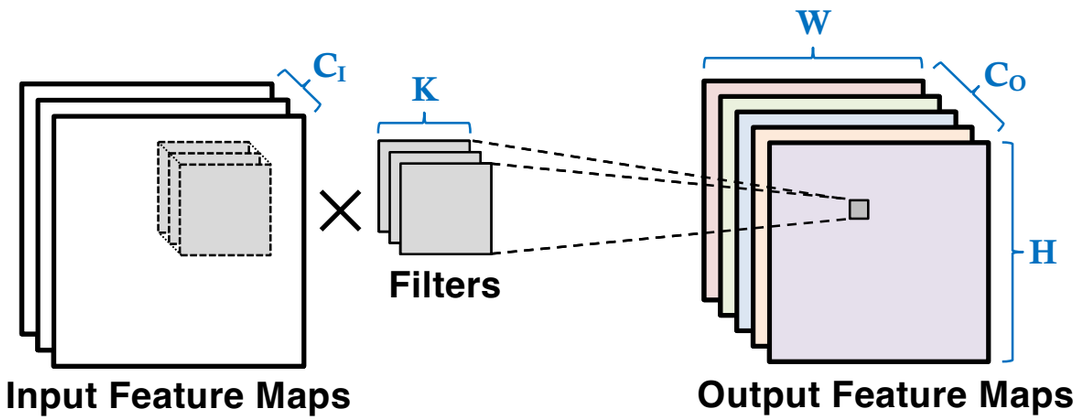
- AI workloads contain abundant DLP across tensors, neurons, channels, and training examples
- Matrix multiplications and convolutions perform the same operations on many independent data elements
- Efficient DLP exploitation is essential for making AI training and inference practical

Case Study: DLP in a Convolutional Layer



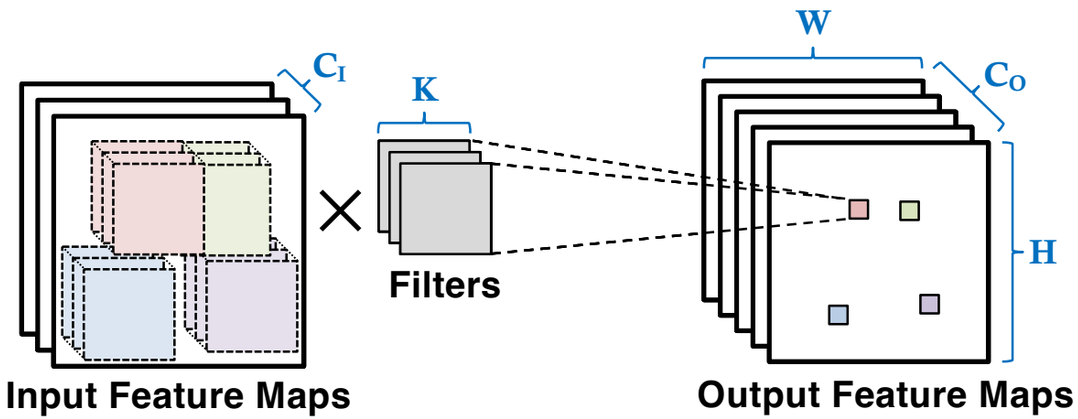
- Four main sources of parallelism
  1. **Across input feature maps**

Case Study: DLP in a Convolutional Layer



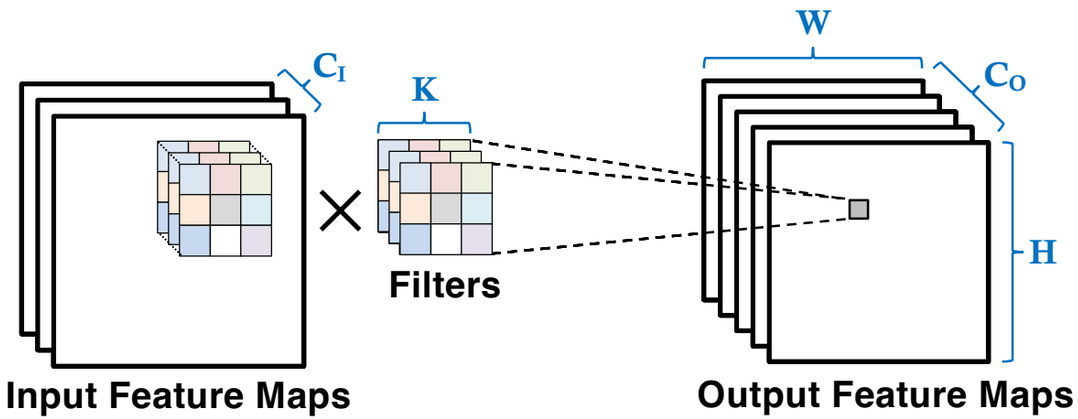
- Four main sources of parallelism
  1. Across input feature maps
  2. **Across output feature maps**

Case Study: DLP in a Convolutional Layer



- Four main sources of parallelism
  1. Across input feature maps
  2. Across output feature maps
  3. **Across different output pixels** (i.e., filter positions)

Case Study: DLP in a Convolutional Layer

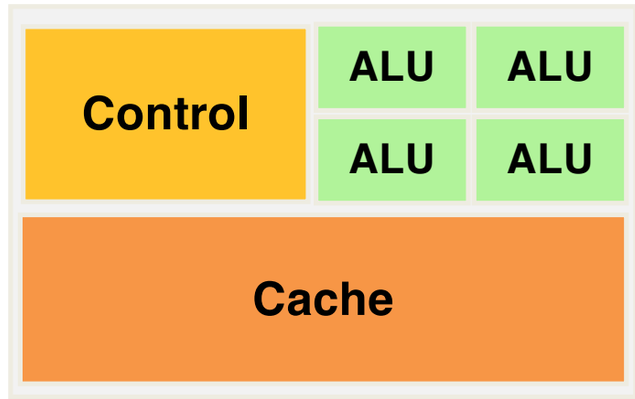


- Four main sources of parallelism
  1. Across input feature maps
  2. Across output feature maps
  3. Across different output pixels (i.e., filter positions)
  4. **Across filter pixels**

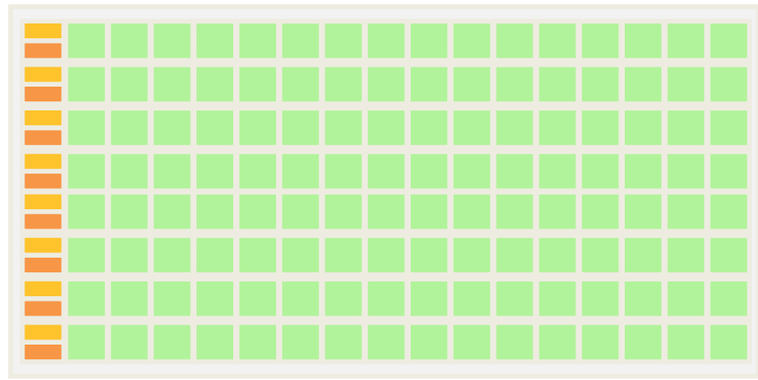
Graphics Processing Unit (GPU)

- GPU has thousands of cores to run many threads in parallel
  - Cores are simpler (compared to CPU)
    - No support of superscalar, OOO, speculative execution, etc.
    - ISA not backward compatible
  - Amortize overhead through Single Instruction, Multiple Threads (SIMT), which implicitly exploits SIMD execution across threads

- Optimized to **increase throughput of running data-parallel workloads**
  - Initially targeting graphics code
  - Latency tolerant with many concurrent threads

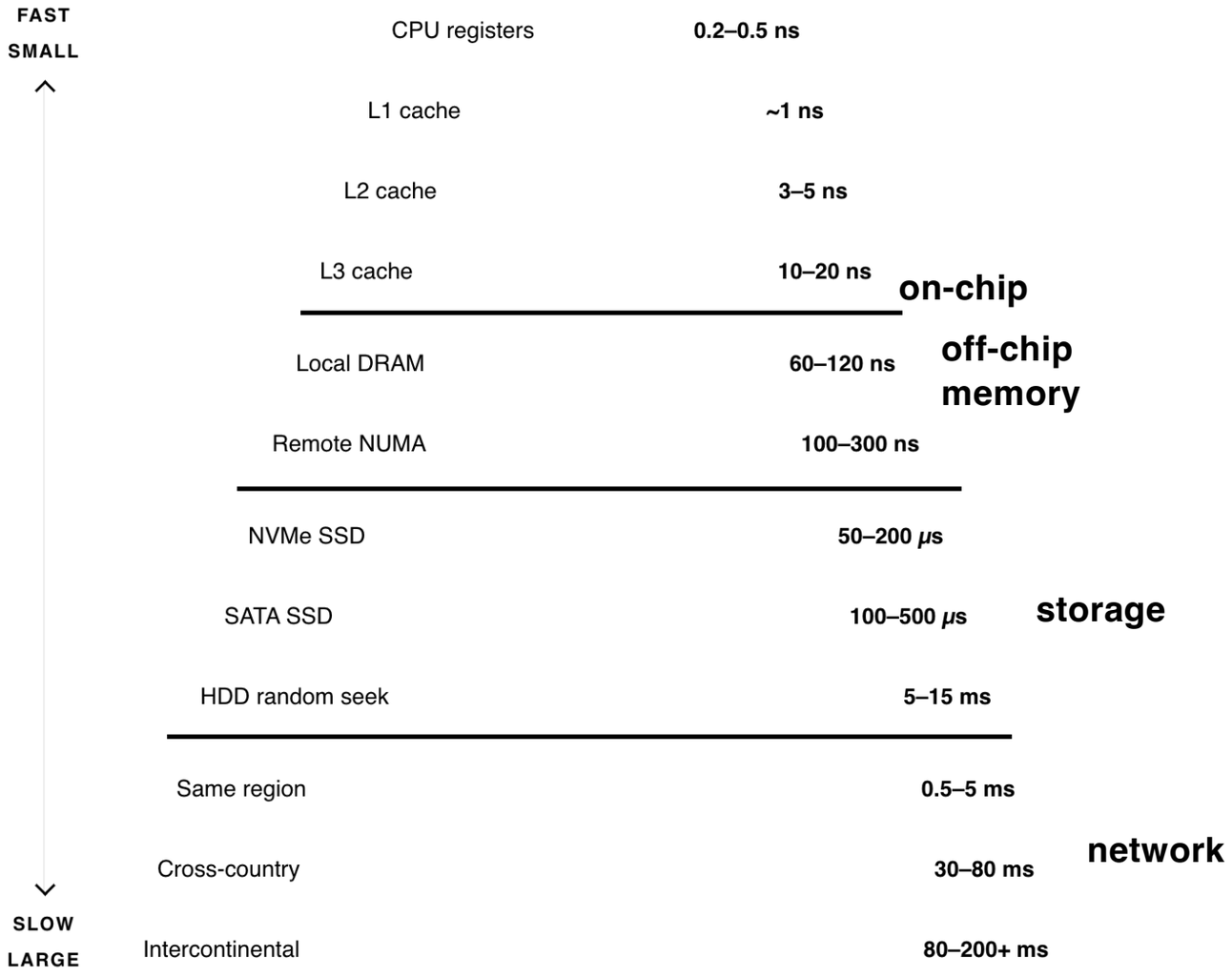


CPU



GPU

Beyond Parallelism: Data Movement is Key to Performance



Latency pyramid for a modern computer

It’s Not Just About Performance: Computing’s Energy Problem

Reading two 32b words from	**Energy**
DRAM	1.3 nJ
Large SRAM (256MB)	58 pJ
Small SRAM (8KB)	5 pJ
Moving two 32b words by	**Energy**
40mm (across a 400mm2 chip)	77 pJ
1mm (local communication)	1.9 pJ
Arithmetic on two 32b words	**Energy**
FMA (float fused multiply-add)	1.2 pJ
IADD (integer add)	0.02 pJ

65,000x

250x

Source: William Dally and Uzi Vishkin, On the Model of Computation, CACM’2022.

Data supply far outweighs arithmetic operations in energy cost

Rough Energy Breakdown for an Instruction

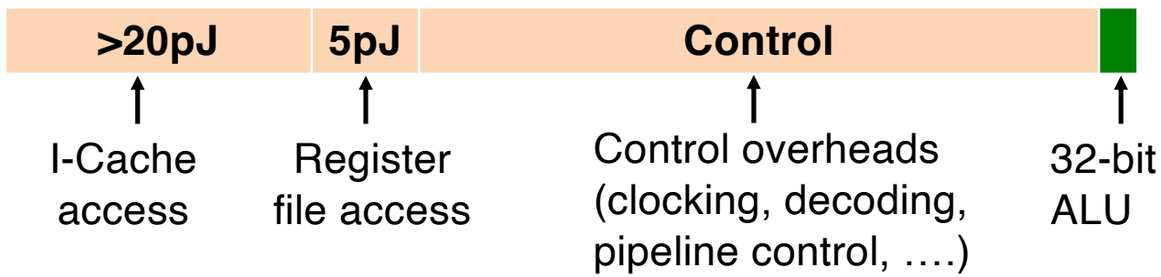


Diagram adapted from W. Qadder, et al., Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing, ISCA'2013.

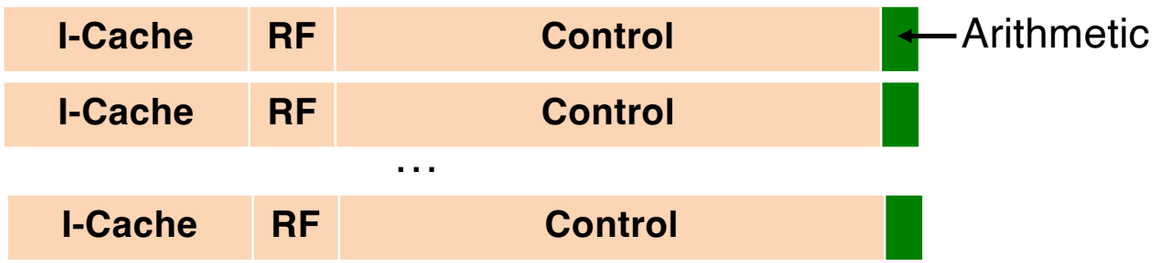
Principles for Improving Energy Efficiency

Do less work!

- **Amortize overhead** in control and data supply across multiple instructions

Amortizing the Overhead

A sequence of energy-inefficient instructions



Single instruction multiple Data (SIMD): tens of operations per instruction

Further specialization (what we achieve using accelerators)

Diagram adapted from W. Qadder, et al., Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing, ISCA'2013.

Principles for Improving Energy Efficiency

Do less work!

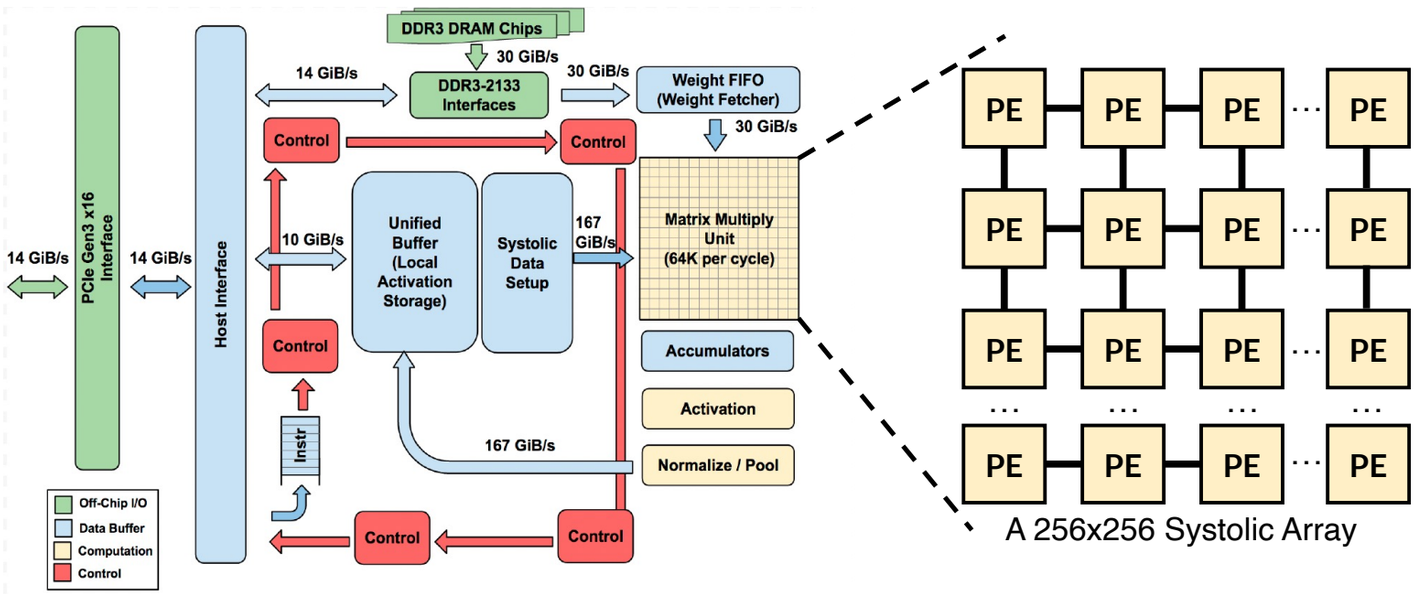
- **Amortize overhead** in control and data supply across multiple instructions

Do even less work!

- **Use smaller (or simpler) data** => cheaper operations, lower storage & communication costs
- **Move data locally and directly**
  - Store data nearby in simpler memory (e.g., scratchpads are cheaper than cache)
  - Wire compute units for direct (or even combinational) communication when possible

Example: Tensor Processing Unit (TPU)

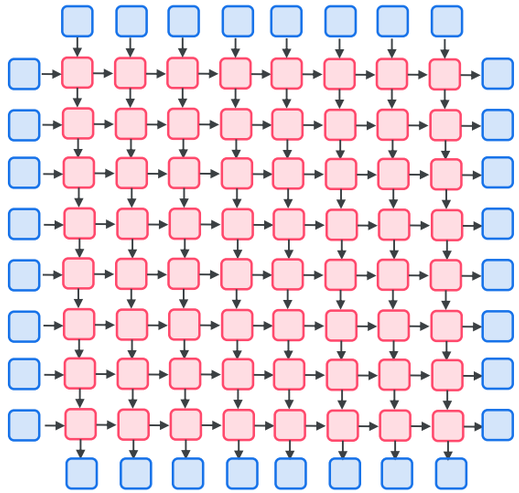
- A domain-specific accelerator for deep learning
  - Main focus is accelerating matrix multiplication (MM) with a systolic array
    - Use CISC instructions: MM Unit may take thousands of cycles
  - TPUv1 does 8-bit integer (INT8) inference; TPUv2 supports a customized floating-point type (bfloat16) for training



Google TPU v1

Source: Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017

Essence of Accelerator Design



- Implement domain- or application-specific compute patterns as processing elements (PEs) and spatially replicate them
  - *Accelerators are inherently throughput machines*
- Key to high performance and efficiency
  - **Instantiate as many PEs as possible** within the area/power budget
  - **Deliver data efficiently** to keep these PEs busy
- Common memory-system bottlenecks (on- & off-chip)
  - Limited (or inefficient use of) **memory capacity**
  - Limited (or inefficient use of) **memory bandwidth**
  - High off-chip **memory access latency**

Common, Cross-Cutting Specialization Techniques

Specialized Compute: Use coarse-grained operations (e.g., SIMD and “ASIC-in-an-instruction”) to improve performance and amortize instruction overhead

Specialized Numerics: Use custom data types to trade precision for higher compute density and lower memory capacity and bandwidth requirements

Specialized Memory: Tailor the memory hierarchy to workload access patterns, improving bandwidth, latency, usable capacity, and energy consumption

Specialized Communication: Tailor on-chip interconnects to workload dataflow patterns for efficient, scalable data movement

Co-Design: These forms of specialization overlap and are often optimized together

Next Lecture

- Hardware Specialization, Part 2

Acknowledgements

- These slides contain/adapt materials from
  - Ritchie Zhao (Cornell PhD, now NVIDIA)
  - Niansong Zhang (Cornell PhD)