UC San Diego

DSC 102
Systems for Scalable Analytics

Rod Albuyeh

Topic 3: Parallel and Scalable Data Processing

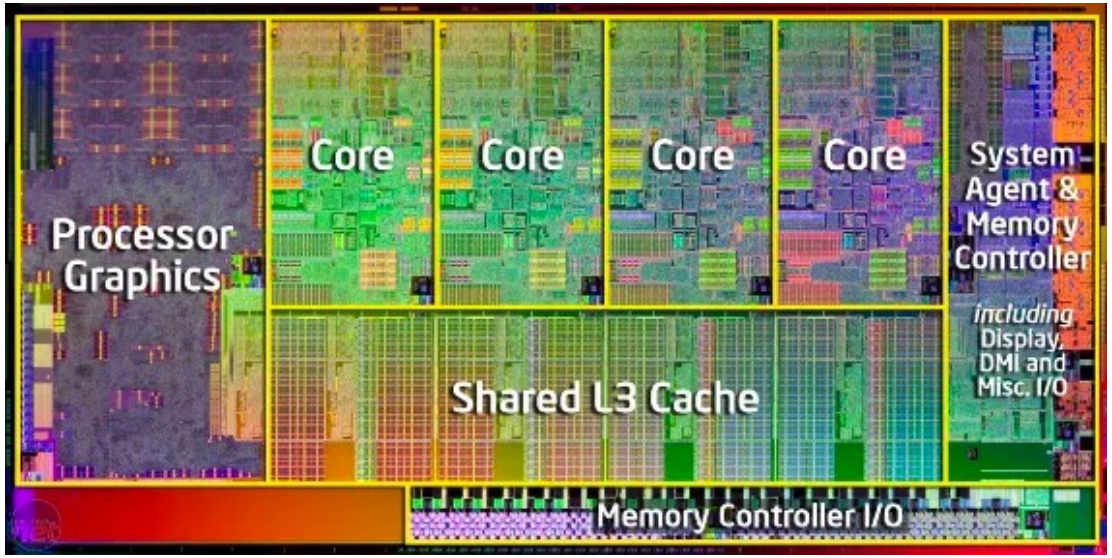
Part 1: Parallelism Basics

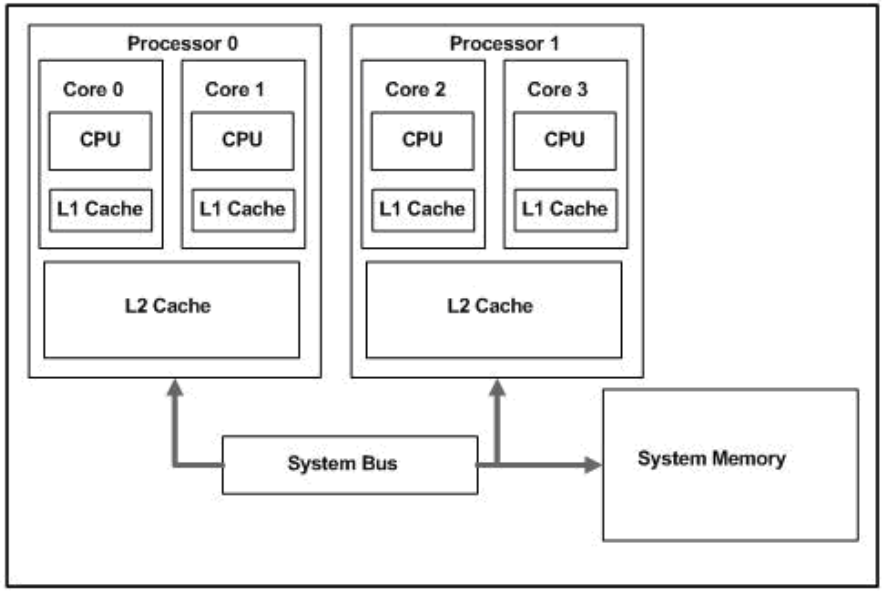
Outline

- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism

Multi-core CPUs

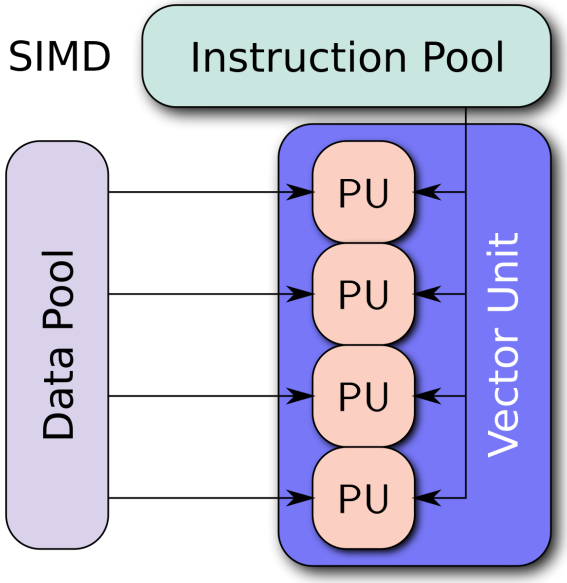
- Modern machines often have multiple processors and multiple *cores* per processor; hierarchy of shared caches
- OS Scheduler now controls what cores/processors assigned to what processes/threads when



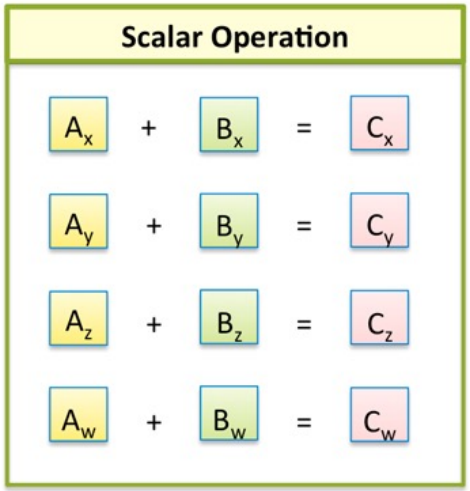


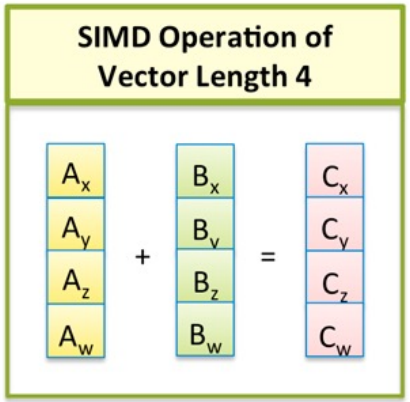
Single-Instruction Multiple-Data

- **Single-Instruction Multiple-Data (SIMD):** A fundamental form of parallel processing in which *different chunks of data* are processed by the *"same" set of instructions* shared by multiple processing units (PUs)
- Aka "vectorized" instruction processing (vs "scalar")
- Data science workloads are very amenable to SIMD
- Note: no "master" scheduler in this scenario



Example for SIMD in data science:



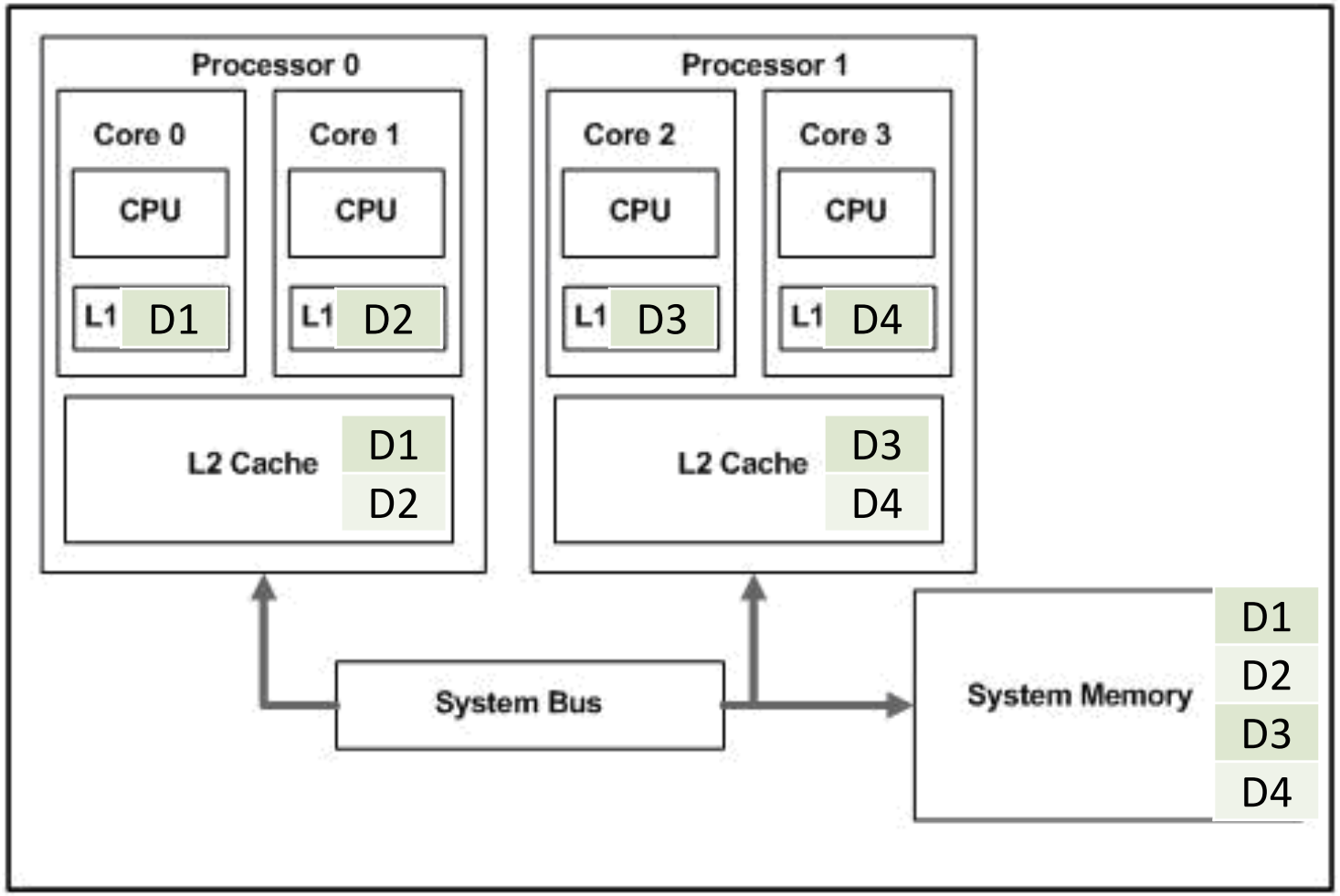


Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

SIMD Generalizations

- **Single-Instruction Multiple Thread (SIMT):** Generalizes notion of SIMD to *different threads* concurrently doing so
  - Each thread may be assigned a core or a whole PU
- **Single-Program Multiple Data (SPMD):** A higher level of abstraction generalizing SIMD operations or programs
  - Under the hood, may use multiple processes or threads
  - Each chunk of data processed by one core/PU
  - Applicable to any CPU, not just vectorized PUs
  - Most common form of parallel programming
  - In this case, work is distributed from a central scheduler or orchestrator.

“Data Parallel” Multi-core Execution



Quantifying Efficiency: Speedup

Q: *How do we quantify the runtime performance benefits of multi-core parallelism?*

- As with task parallelism, we measure the speedup:

$$\textbf{Speedup} = \frac{\text{Completion time given only 1 core}}{\text{Completion time given n (>1) core}}$$

- In data science computations, an often useful surrogate for completion time is the instruction throughput **FLOP/s**, i.e., *number of floating point operations* per second
- Modern data processing programs, especially deep learning (DL) may have billions of FLOPs aka GFLOPs!

Amdahl's Law

Q: *But given n cores, can we get a speedup of n?*

It depends! (Just like it did with task parallelism)

- **Amdahl's Law:** Formula to upper bound possible speedup
  - A program has 2 parts: one that benefits from multi-core parallelism and one that does not
  - Non-parallel part could be for control, memory stalls, **traversing a linked list...**

1 core: n cores:

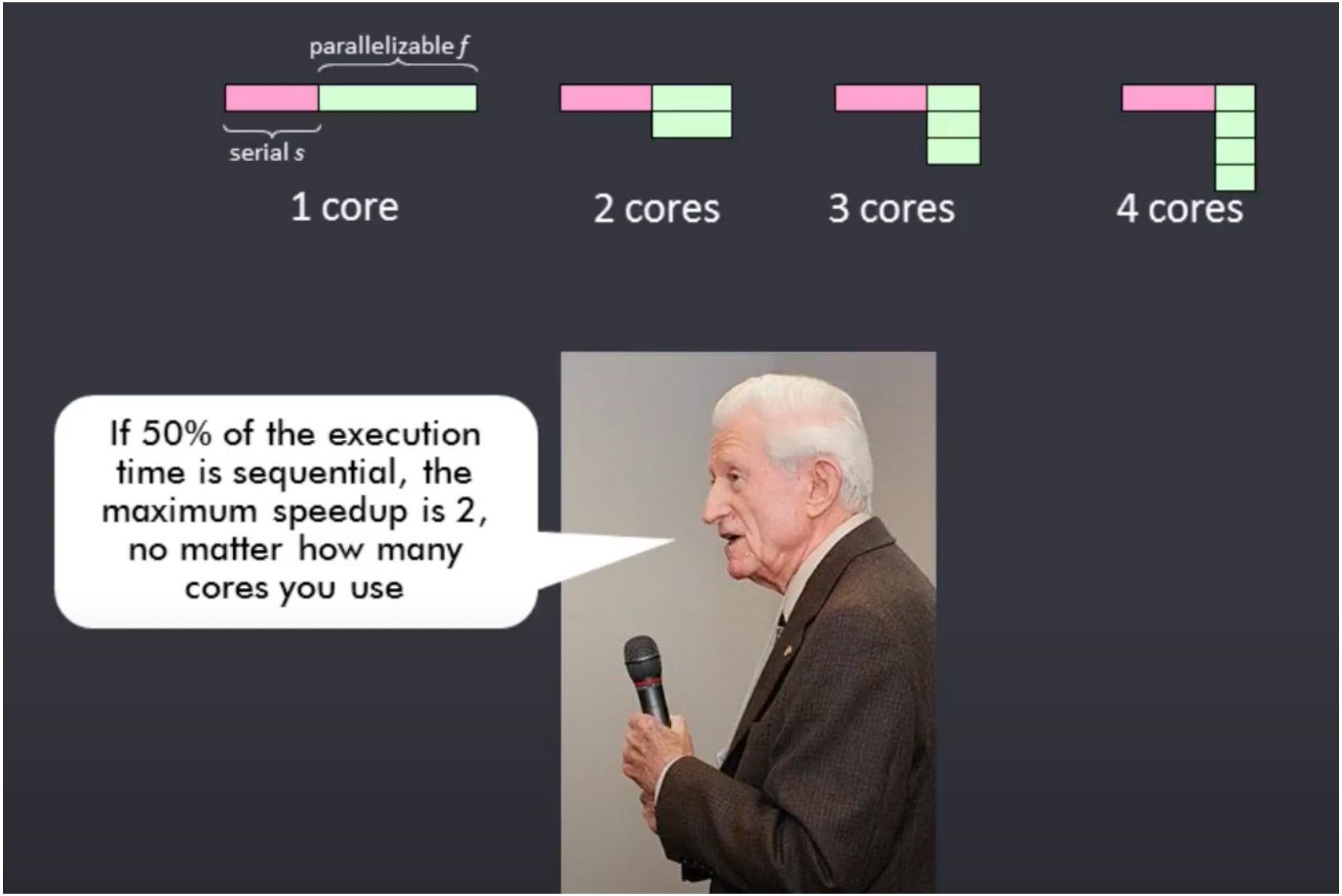
$T_{yes} \longrightarrow T_{yes}/n$

$T_{no} \longrightarrow T_{no}$

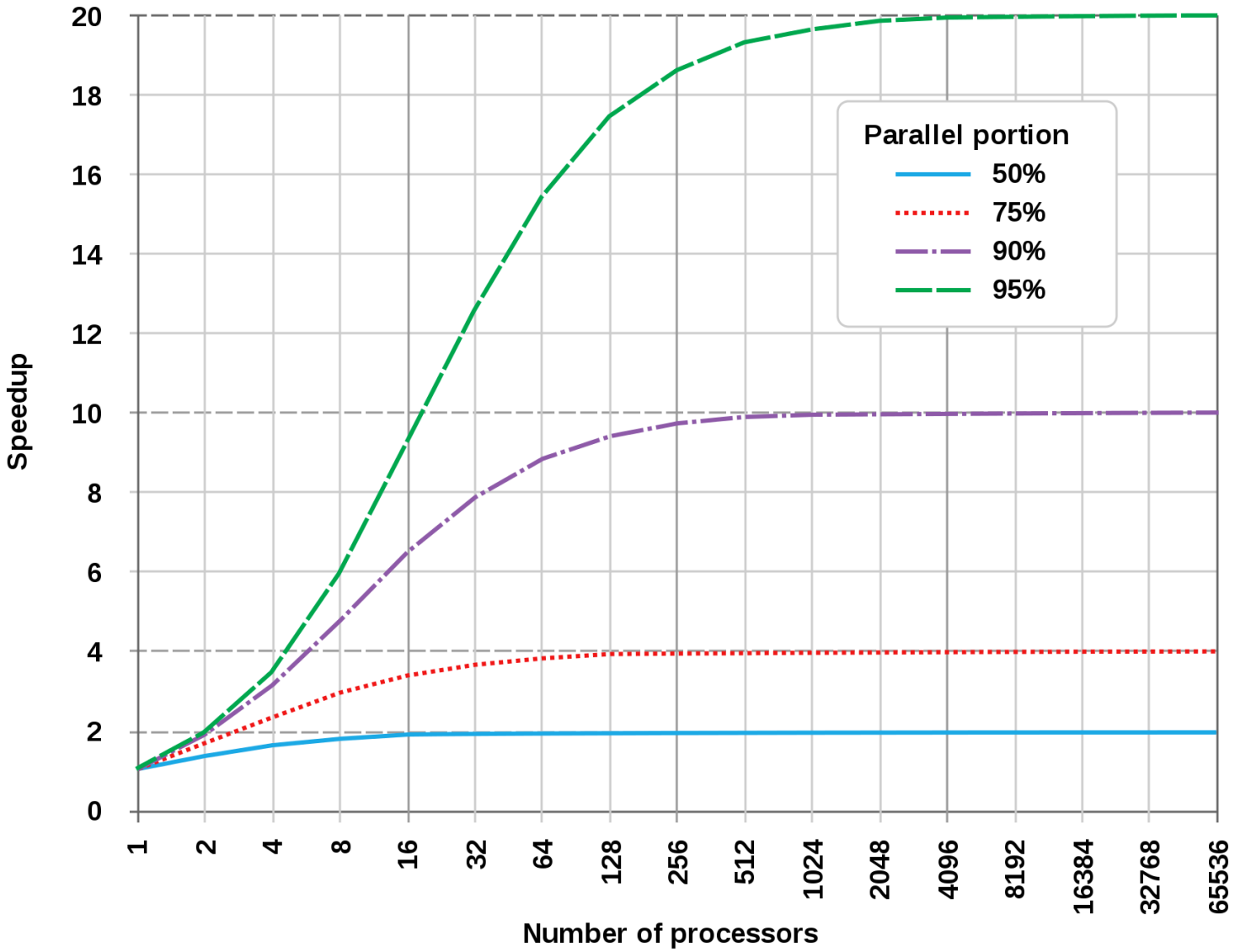
$$\text{Speedup} = \frac{T_{yes} + T_{no}}{T_{yes}/n + T_{no}} = \frac{n(1 + f)}{n + f}$$

Denote $T_{yes}/T_{no} = f$

Amdahl's Law


parallelizable f
serial s
1 core
2 cores
3 cores
4 cores
If 50% of the execution time is sequential, the maximum speedup is 2, no matter how many cores you use

Amdahl's Law



$$\text{Speedup} = \frac{n(1+f)}{n+f}$$

$$f = T_{yes}/T_{no}$$

$$\text{Parallel portion} = f / (1+f)$$

Hardware Trends on Parallelism

- Multi-core processors grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues

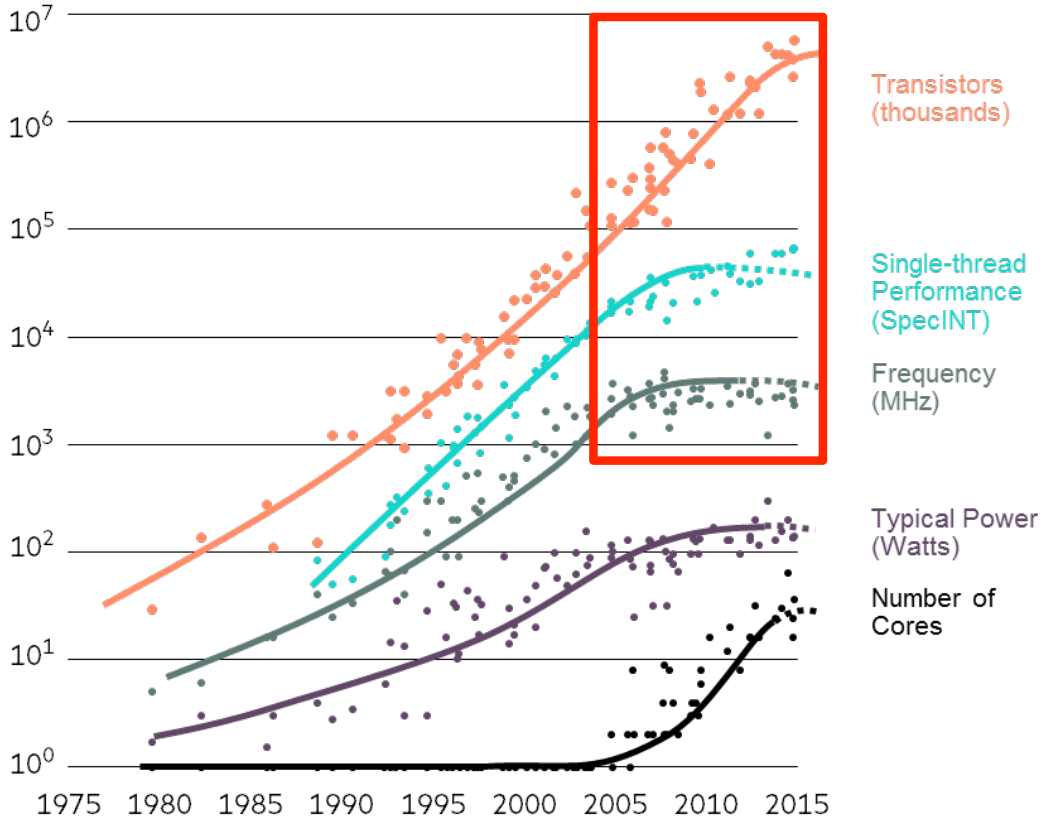
Moore's Law: The number of transistors in a dense integrated circuit doubles about every two years. Has not held up due to physical limits of miniaturization, increasing costs, and emergence of alternative innovations.

Dennard Scaling: As transistors get smaller, their power density stays constant, so that the power use stays in proportion with area. Has not held up because transistors have become so small that their power consumption has started to increase again, and accordingly, heat.

- End of “Moore’s Law” and end of “Dennard Scaling”.

Hardware Trends on Parallelism

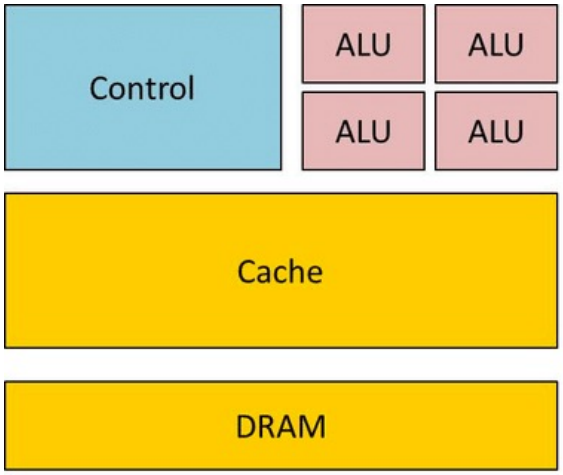
- Multi-core processors grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues.



- Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOP-heavy programs like deep nets
- Motivated the rise of “accelerators” for some classes of programs

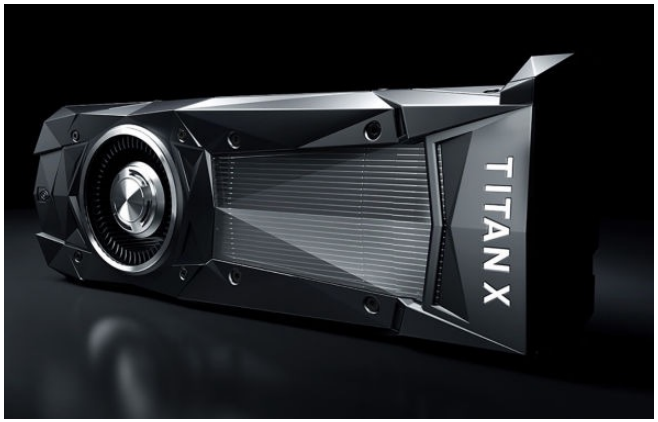
Hardware Accelerators: GPUs

- **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- Basic idea: use tons of ALUs; massive data parallelism (SIMD on steroids); Titan X offers ~11 TFLOP/s!
- Popularized by NVIDIA in early 2000s for video games, graphics, video/multimedia, cryptomining; now ubiquitous in deep learning
- CUDA released in 2007; later wrapper APIs on top: CuDNN, CuSparse, CuDF (RapidsAI), etc.

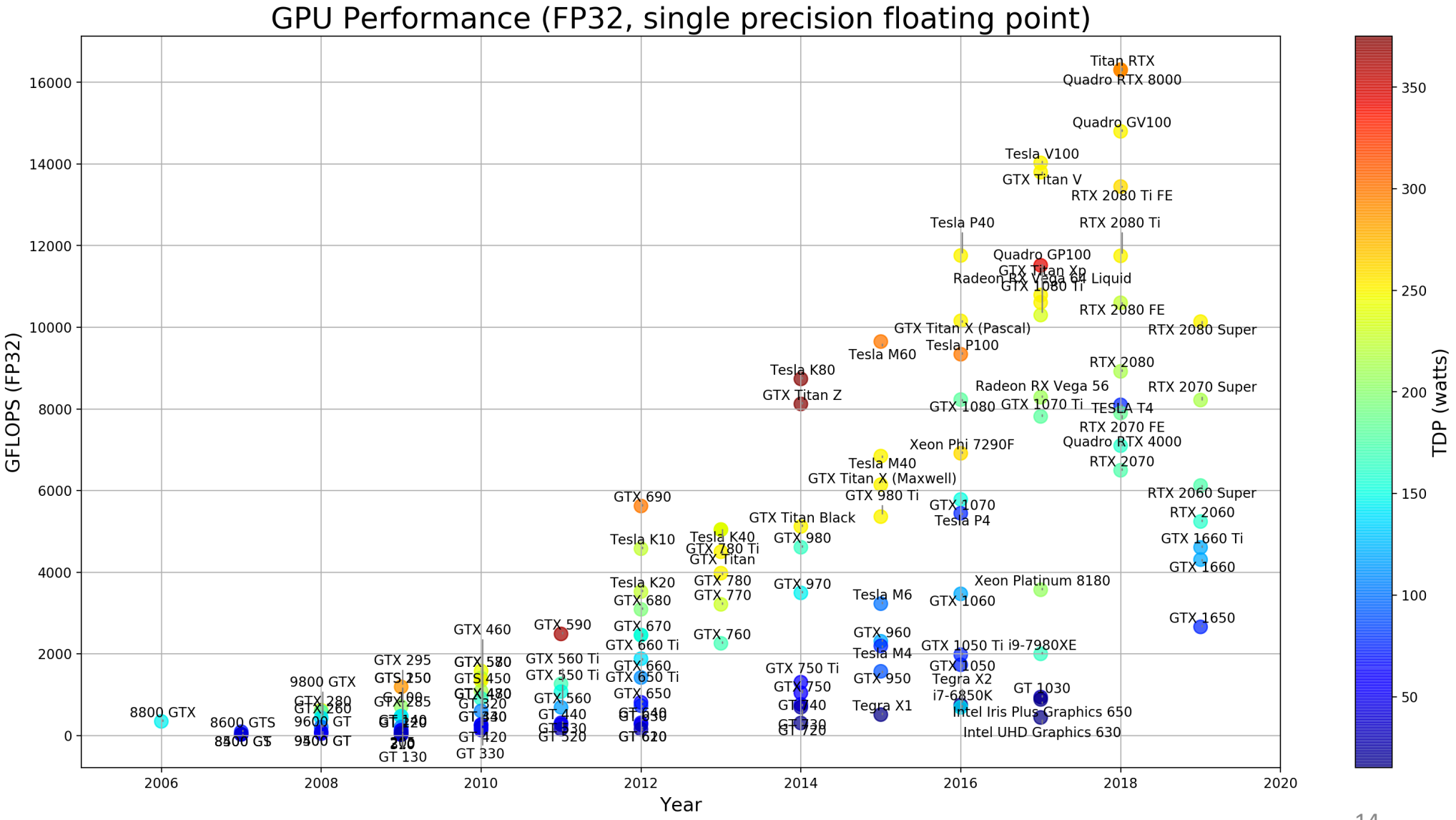


CPU

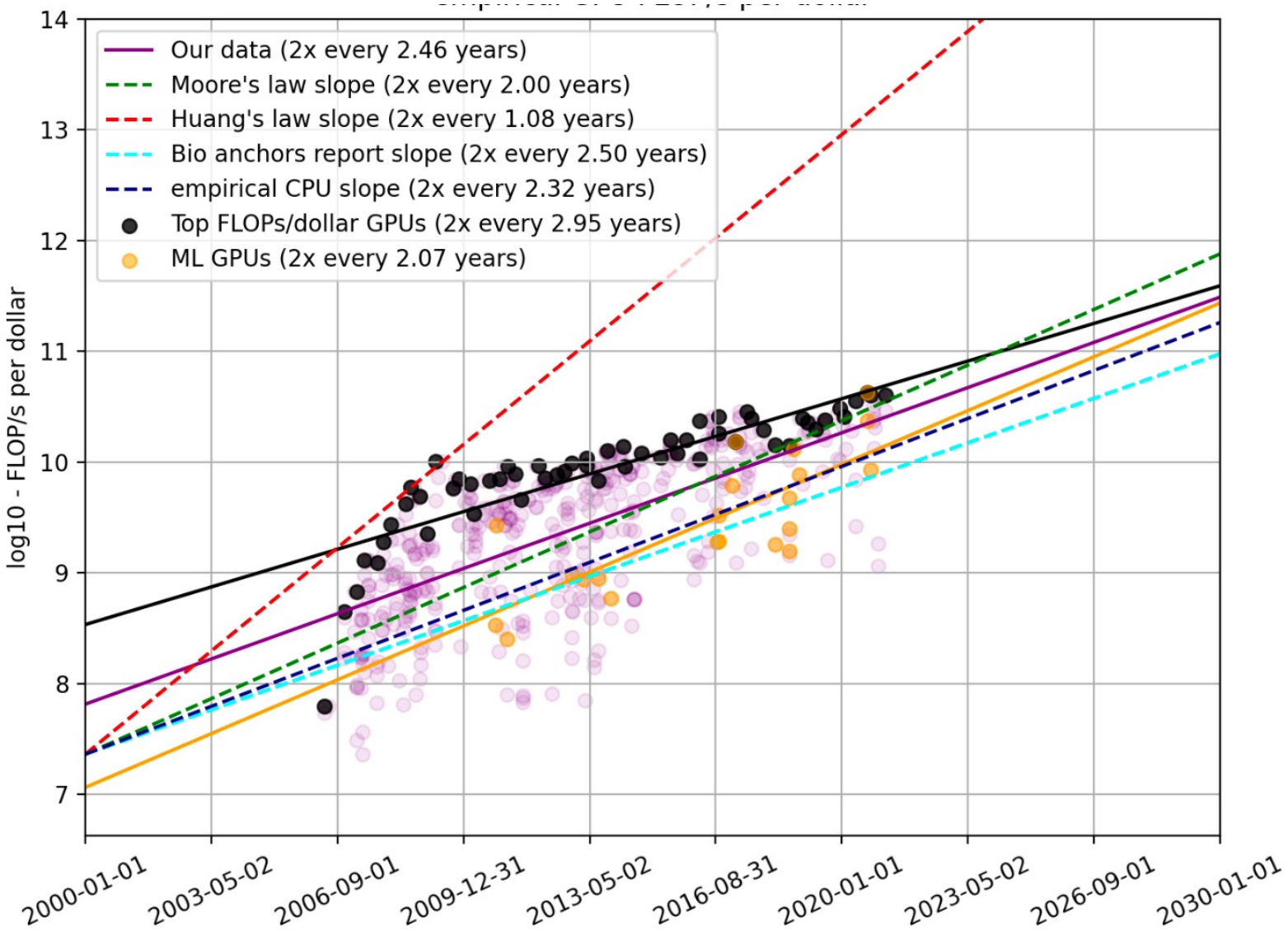
GPU



GPUs on the Market

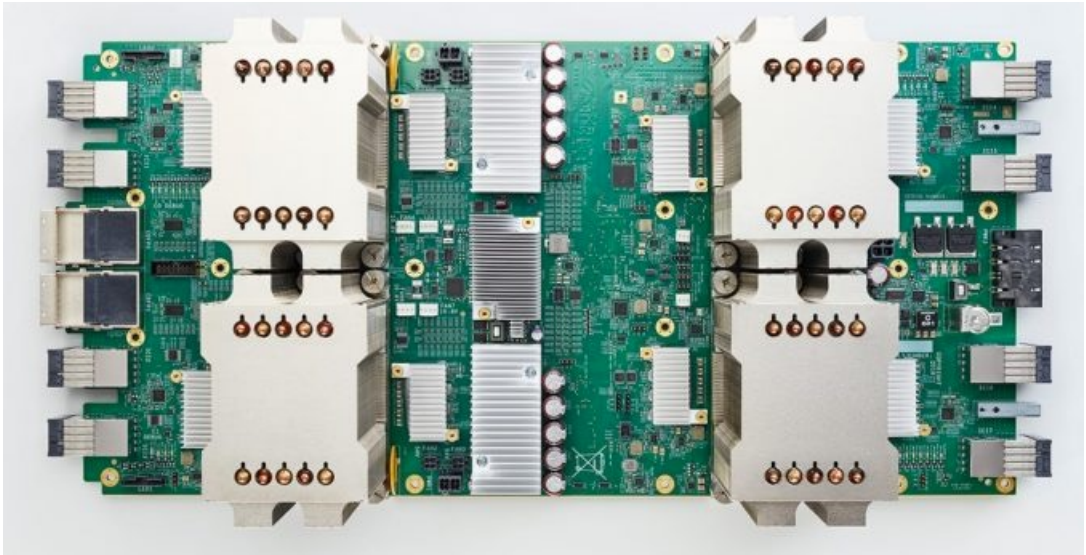


Empirical GPU FLOP/s per dollar



Other Hardware Accelerators

- **Tensor Processing Unit (TPU)**: Even more specialized tensor ops in DL inference; ~45 TFLOP/s!
  - An “application-specific integrated circuit” (ASIC) created by Google in mid 2010s; used for AlphaGo!
- **Field-Programmable Gate Array (FPGA)**: Configurable for any class of programs; ~0.5-3 TFLOP/s
  - Cheaper; new h/w-s/w stacks for ML/DL; Azure/AWS support



Comparing Modern Parallel Hardware

	Multi-core CPU	GPU	FPGA	ASICs (e.g., TPUs)
Peak FLOPS/s	Moderate	High	High	Very High
Power Consumption	High	Very High	Very Low	Low-Very Low
Cost	Low	High	Very High	Highest
Generality / Flexibility	Highest	Medium	Very High	Lowest
"Fitness" for DL Training?	Poor Fit	Best Fit	Low Fit	Potential exists but yet unrealized
"Fitness" for DL Inference?	Moderate	Moderate	Good Fit	Best Fit
Cloud Vendor Support	All	All	AWS, Azure	AWS, GCP

https://www.embedded.com/leveraging-fpgas-for-deep-learning/

Review Questions

1. Briefly explain 3 benefits of large-scale data in Data Science.
2. What is a dataflow graph? Give an example from a data system.
3. How does a task graph differ from a dataflow graph?
4. Briefly explain 1 benefit and 1 drawback of task parallelism.
5. What is the lower bound on completion time when running a task graph in a task-parallel manner?
6. What is the degree of parallelism of a task graph?
7. Is linear speedup always possible with task parallelism?
8. What is SIMD? Why is “vectorized” data processing critical?
9. What is the point of Amdahl’s Law?
10. Briefly 1 pro and 1 con of TPU vs GPU.

Outline

- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism