https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff
Instructor: Hao Zhang
TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab
haozhang@ucsd.edu

Where We Are

Motivations, Economics, Ecosystems, Trends

Networking

Storage

Part3: Compute

~~Datacenter networking~~

~~Collective communication~~

(Distributed) File Systems / Database

Cloud storage

Distributed Computing

Big data processing

Parallelism

Central Issue: Workload takes too long for one processor!

Basic Idea: Split up workload across processors and perhaps also across machines/workers (aka "Divide and Conquer")

Key parallelism paradigms in data systems

- **assuming there will be coordination:**

func \ data	**Shared**	**Replicated**	**Partitioned**
Replicated	N/A (rare cases)		Data parallelism
Partitioned	Task parallelism		Hybrid parallelism

Terms are confusing

- Different domains term them differently in different contexts
- Architecture/parallel computing: single-node multi-cores
  - SIMD, MIMD, SIMT
- Distributed system: multiple-node multi-cores
  - SPMD vs. MPMD
- Machine learning community
  - Data parallelism vs. Model parallelism
  - Inter-operator parallelism vs. Intra-operator parallelism

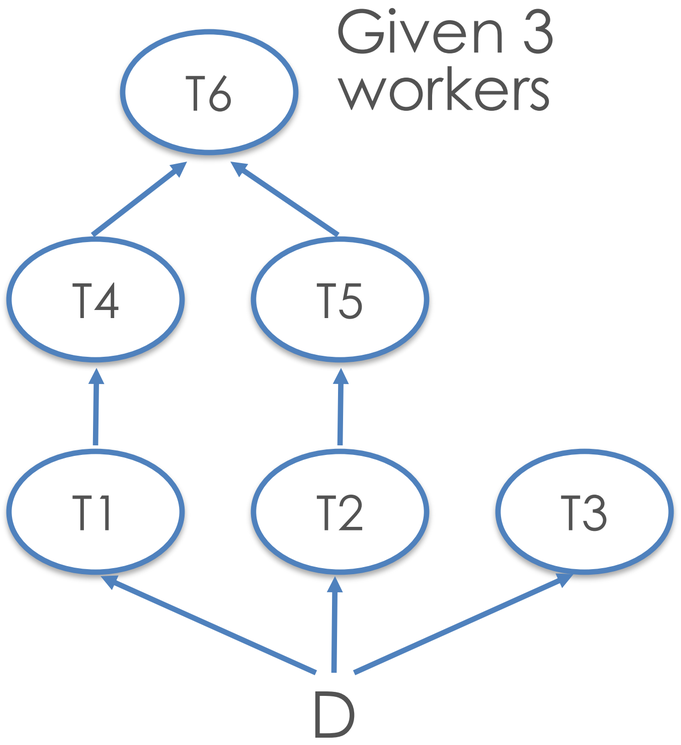
Today's topic: Parallelism

- Express data processing in abstraction
- Parallelisms
  - **Task parallelism**
  - Data parallelism
  - Terms: SIMD, SIMT, SPMD, MPMD

Task Parallelism

Basic Idea: Split up *tasks* across workers; if there is a common dataset that they read, just make copies of it (aka *replication*)

Example:



4) After T4 & T5 end, run T6 on W1; W2 is *idle*

3) After T1 ends, run T4 on W1; after T2 ends, run T5 on W2; after T3 ends, W3 is *idle*

2) Put T1 on worker 1 (W1), T2 on W2, T3 on W3; run all 3 in parallel

1) Copy whole D to all workers

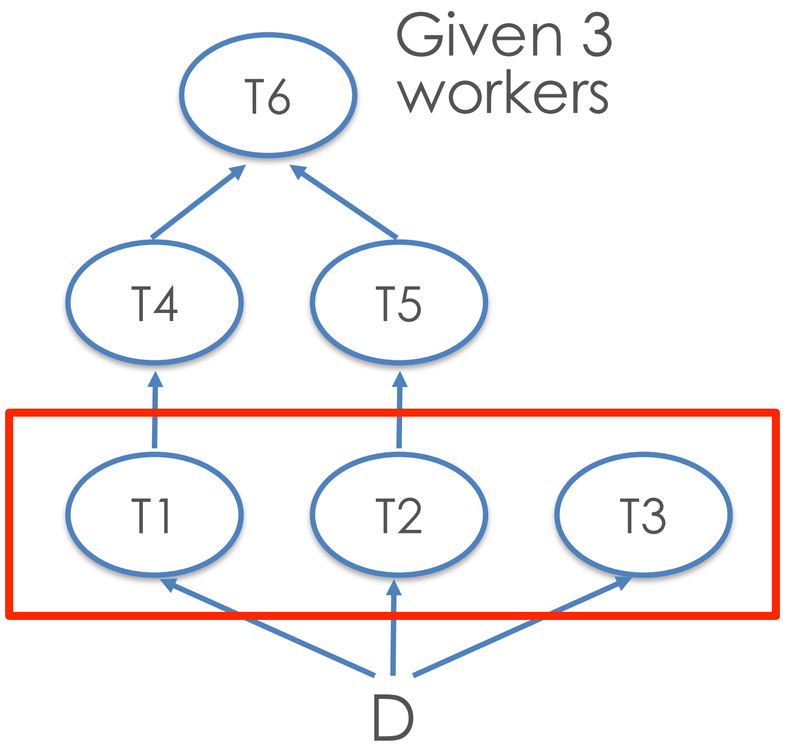
Task Parallelism

- Topological sort of tasks in task graph for scheduling
- Notion of a "worker" can be at processor/core level, not just at node/server level
  - Thread-level parallelism possible instead of process-level
  - E.g., Ray: 4 worker nodes x 4 cores = 16 workers total
- Main pros of task parallelism:
  - Simple to understand
  - Independence of workers => low software complexity
- Main cons of task parallelism:
  - Can be difficult to implement
  - Idle times possible on workers

Degree of Parallelism

- The largest amount of *concurrency* possible in the task graph, i.e., how many task can be run simultaneously

Example:



Q: *How do we quantify the runtime performance benefits of task parallelism?*

But over time, degree of parallelism keeps dropping in this example

Degree of parallelism is only 3

So, more than 3 workers is not useful for this workload!

Quantifying Benefit of Parallelism: Speedup

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

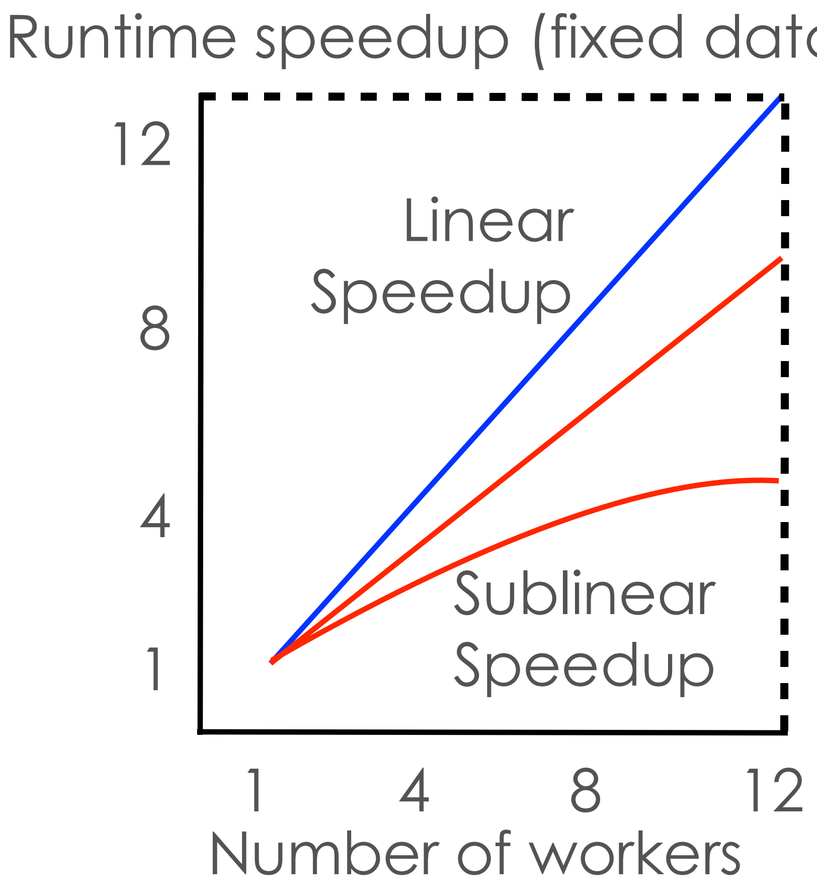
Q: *But given n workers, can we get a speedup of n?*

It depends!

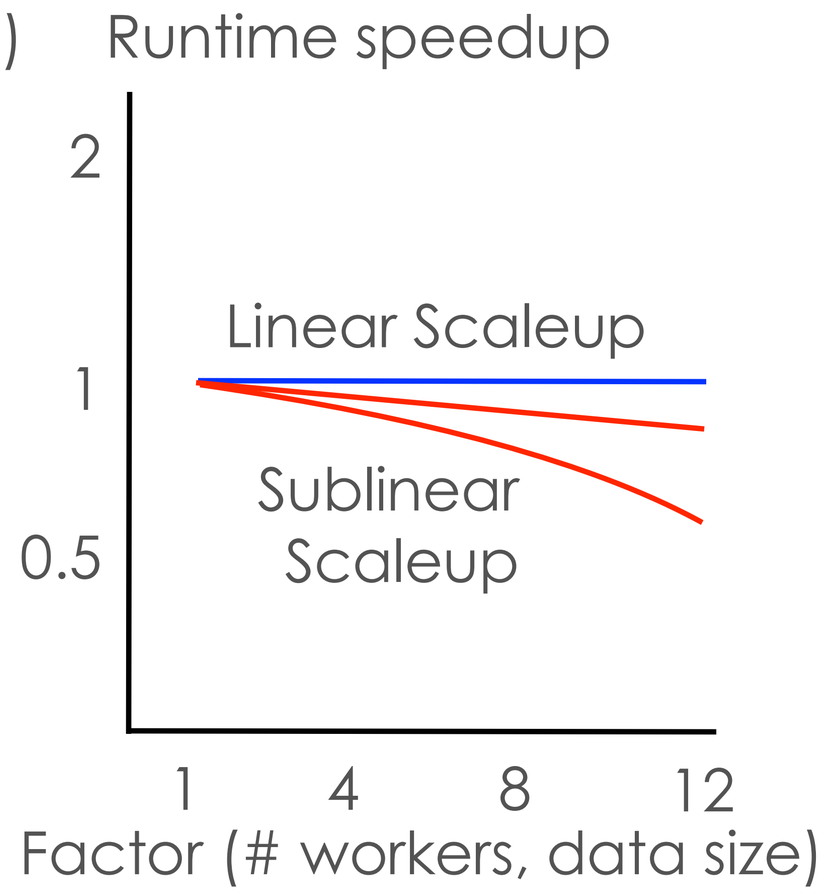
(On degree of parallelism, task dependency graph structure, intermediate data sizes, etc.)

Q: what kind of graphs can give a speedup of n?

Weak and Strong Scaling

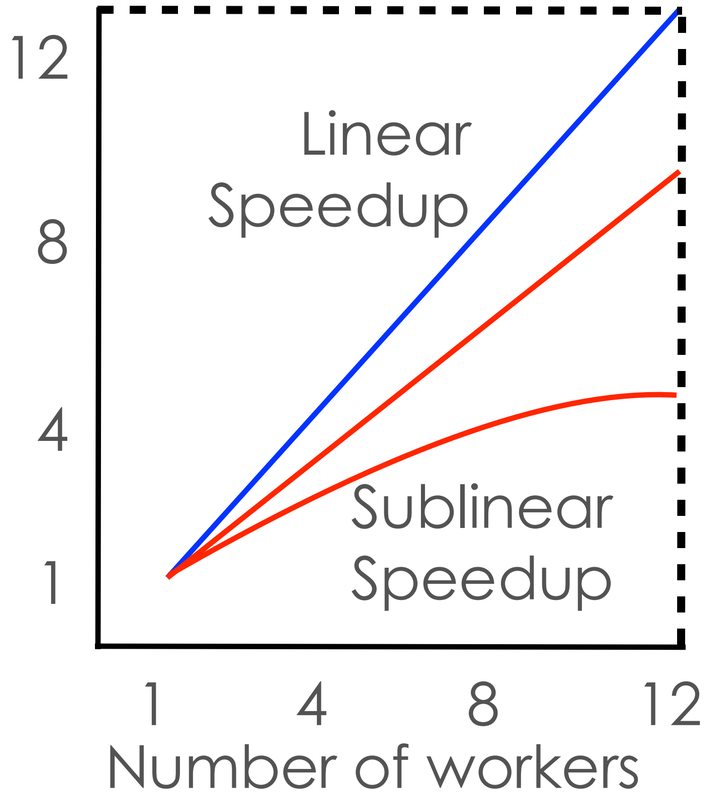


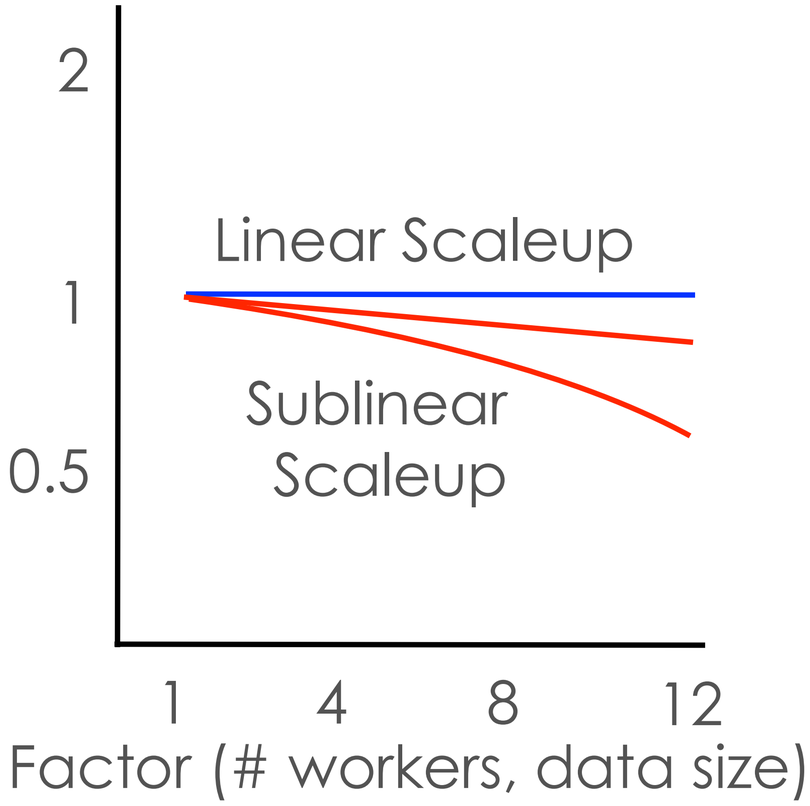
Speedup plot / Strong scaling



Scaleup plot / Weak scaling

Discussion: Is *superlinear* speedup/scaleup possible?





Some Clarifications on Terms

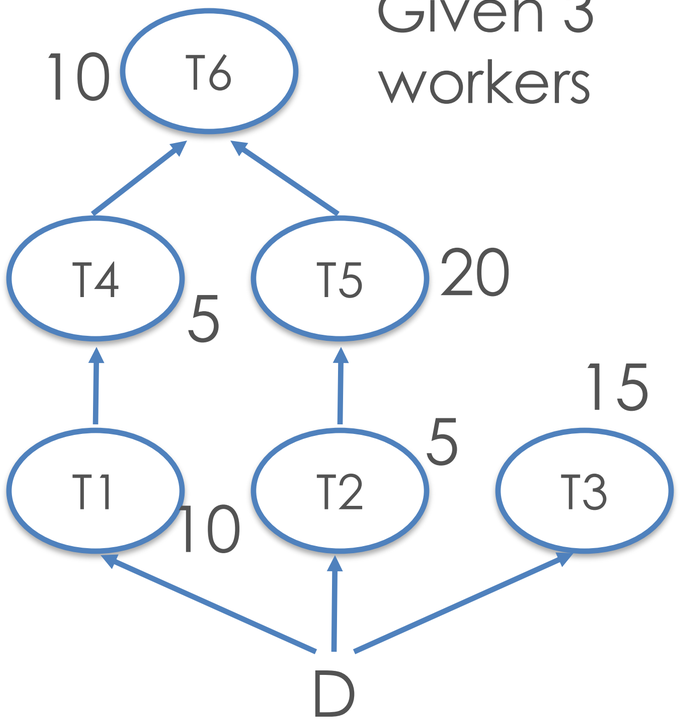
$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

- These terms almost all refer to the above, but they are slightly different
  - Speedup, acceleration -> strong scaling
  - Scaling, scale-up -> weak scaling
  - Scalability -> both
- “system A is very scalable”
  - When you add 1 more workers, the speedup increase by ~1
- “system A is more scalable than system B”
  - When you add 1 more worker, the speedup of system A is larger than that of system B

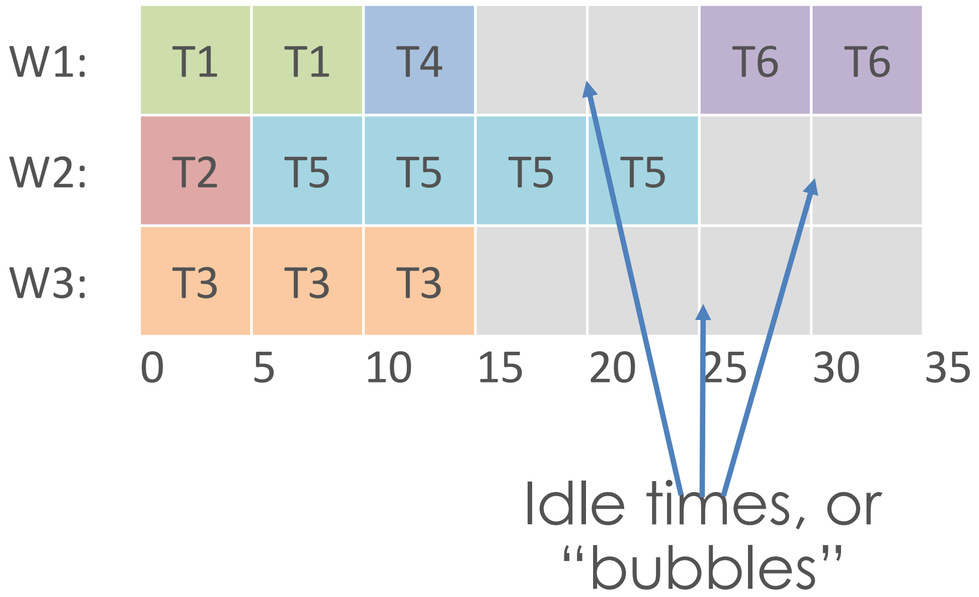
Idle Times in Task Parallelism

Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:



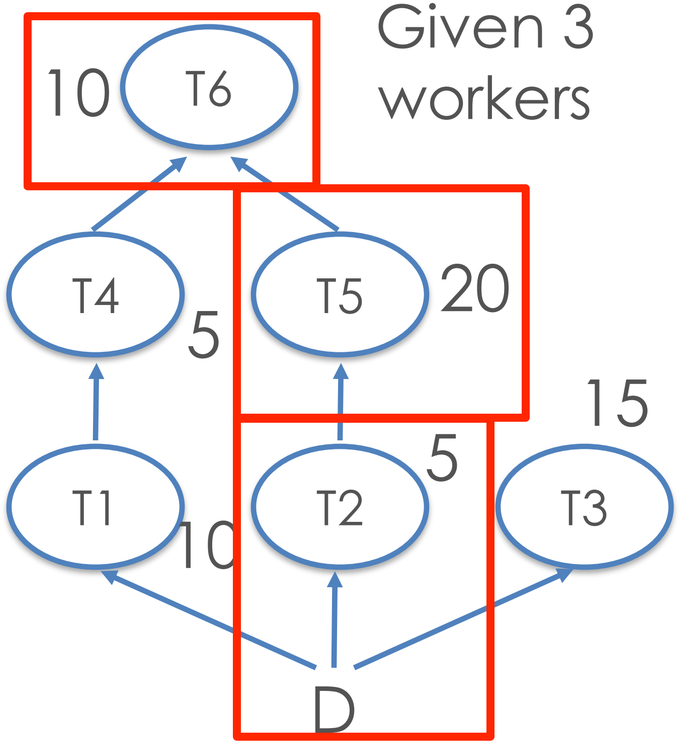
Gantt Chart visualization of schedule:



Idle Times in Task Parallelism

Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:

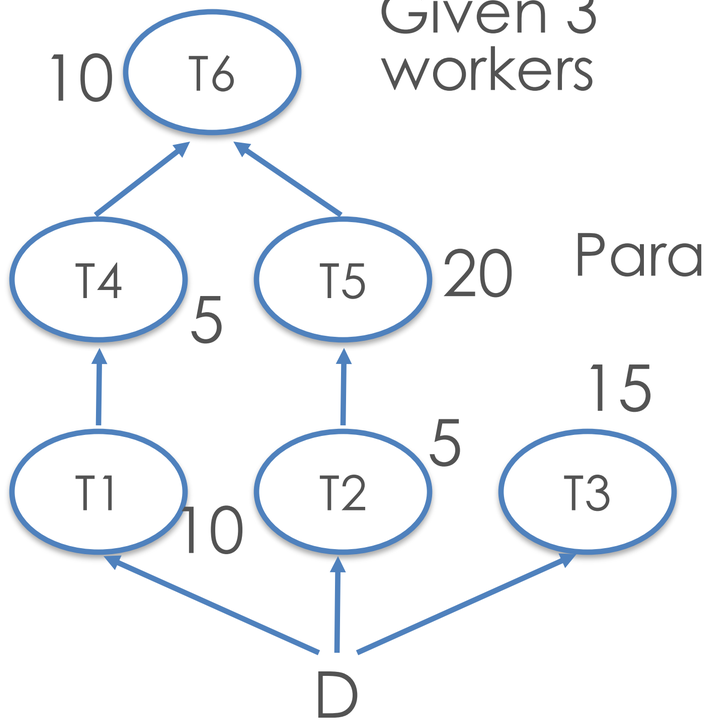


- In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- Possibility: A task-parallel scheduler can "release" a worker if it knows that will be idle till the end
- Can saves costs in cloud

Calculating Task Parallelism Speedup

Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:



Given 3 workers

Completion time with 1 worker: 10+5+15+5+20+10 = 65

Parallel completion time: 35

Speedup = 65/35 = 1.9x

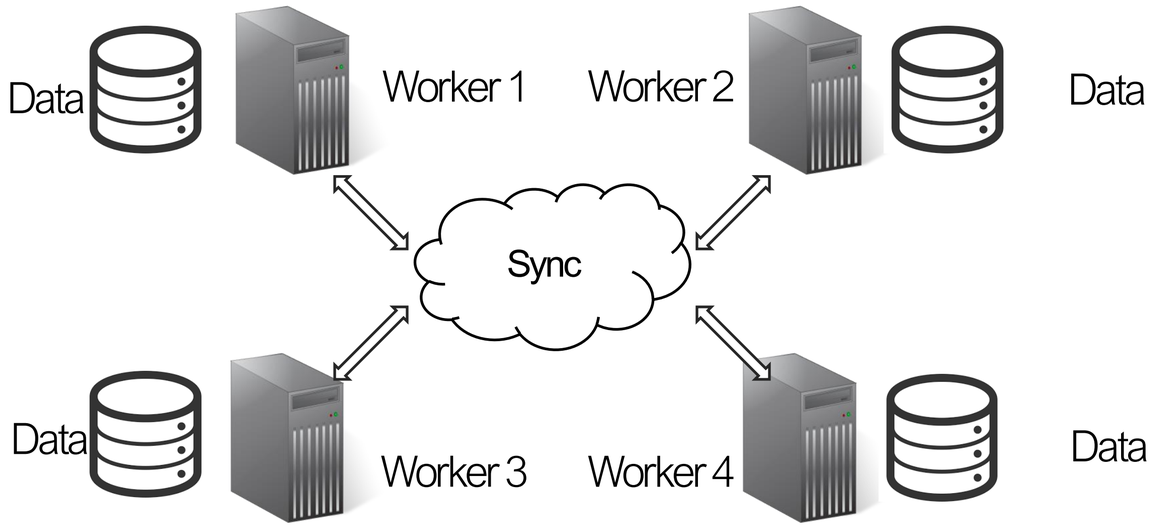
Ideal/linear speedup is 3x

Q: *Why is it only 1.9x?*

Today's topic: Parallelism

- Express data processing in abstraction
- Parallelisms
  - Task parallelism
  - **Data parallelism**
  - Parallel Processing Chips

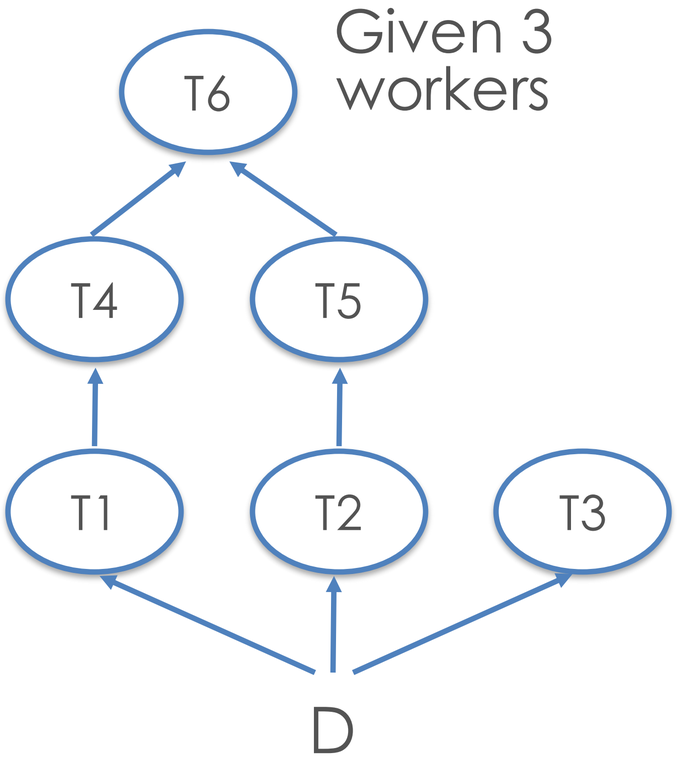
Recall: Data parallelism in ML

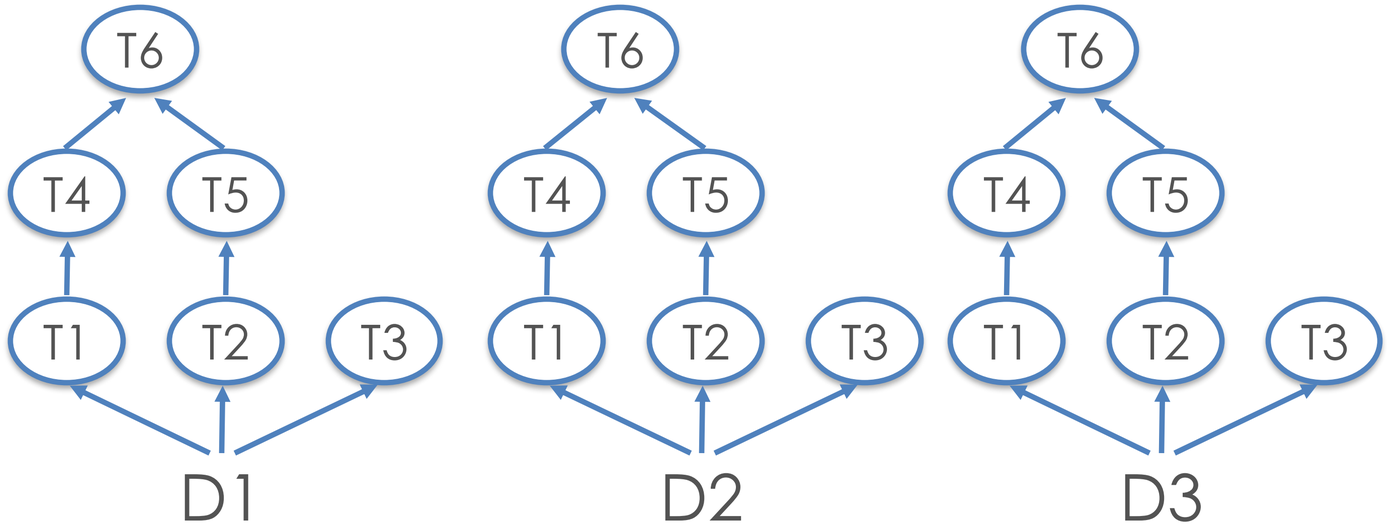


$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

Data parallelism: abstraction of SIMD/SIMT/SPMD

Q: How to represent data parallelism in dataflow graph notions?





Quantifying Efficiency of Data Parallelism

- As with task parallelism, we measure the speedup:

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

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, etc.

1 core: n cores:

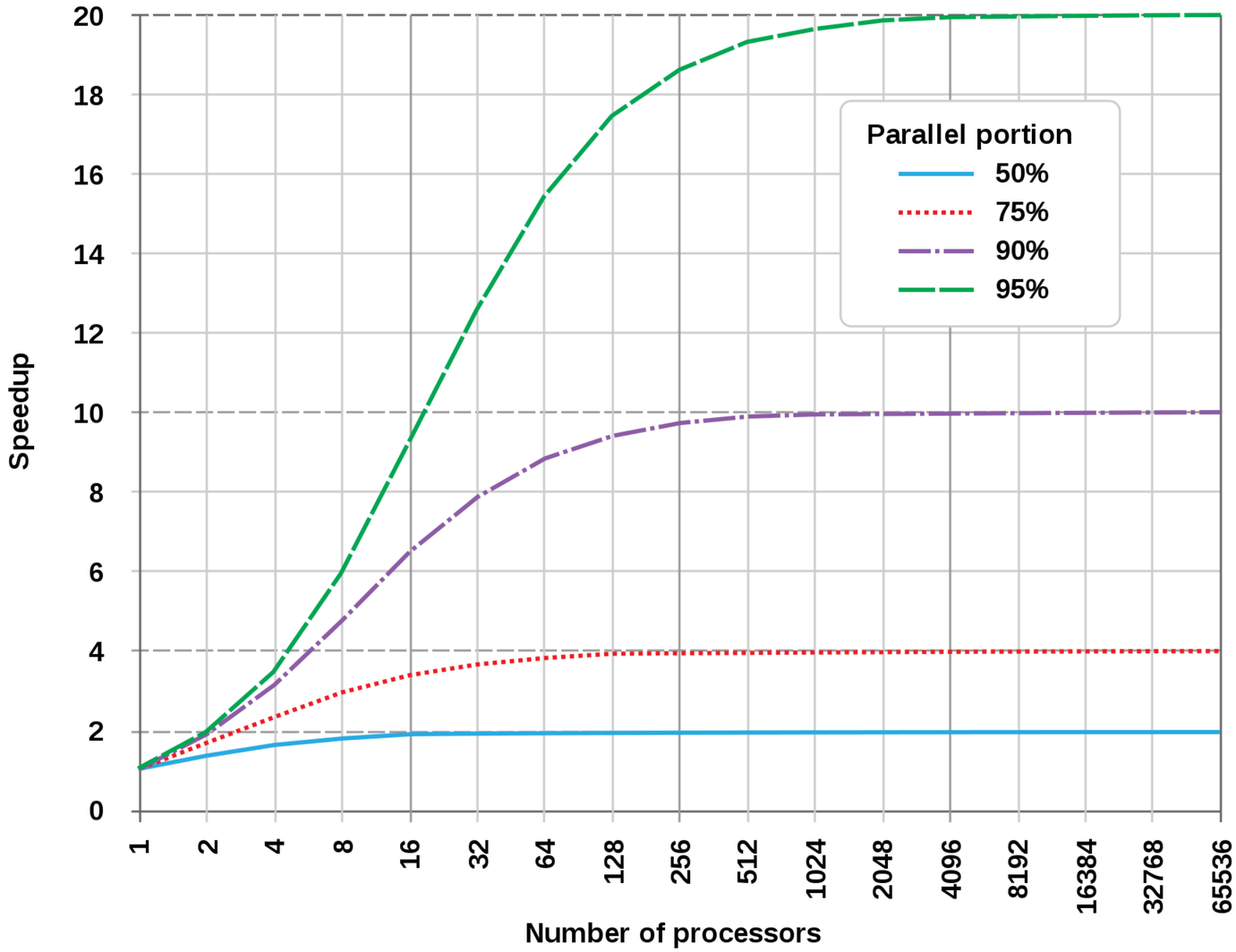
$T_{yes} \longrightarrow T_{yes}/n$

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

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

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

Amdahl's Law:



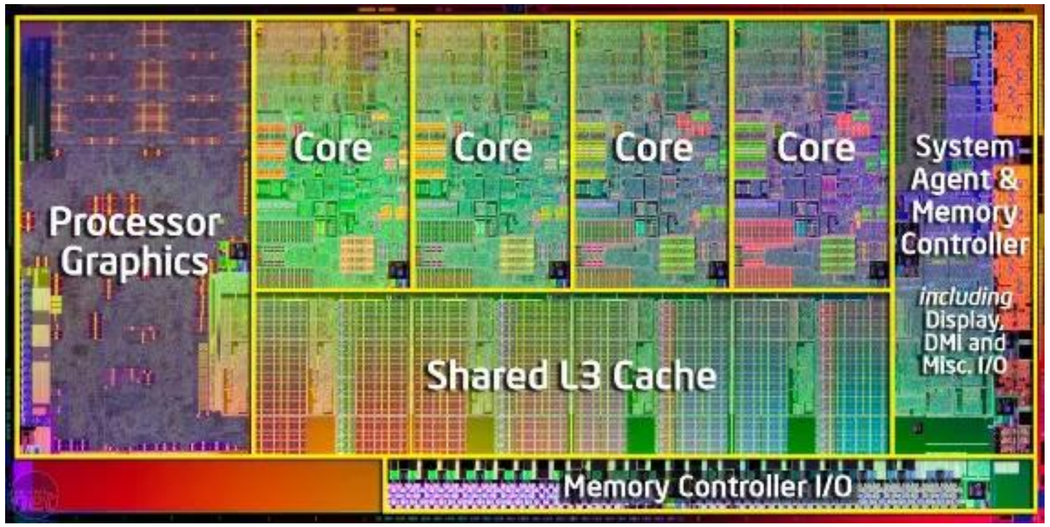
$f = T_{yes}/T_{no}$

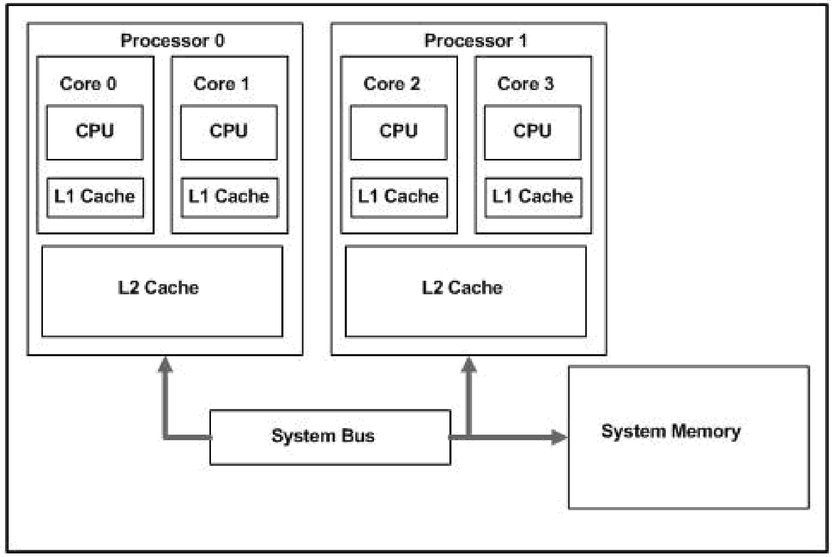
Parallel portion = $f / (1 + f)$

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

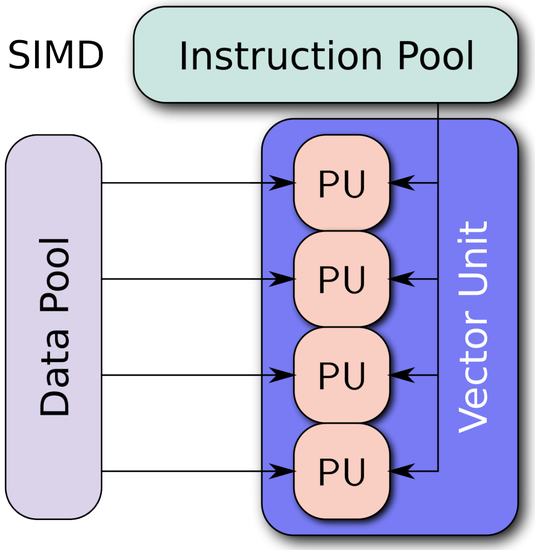
Data parallelism is built in with today's Processors

- Modern computers often have multiple processors and multiple *cores* per processor, with a hierarchy of shared caches

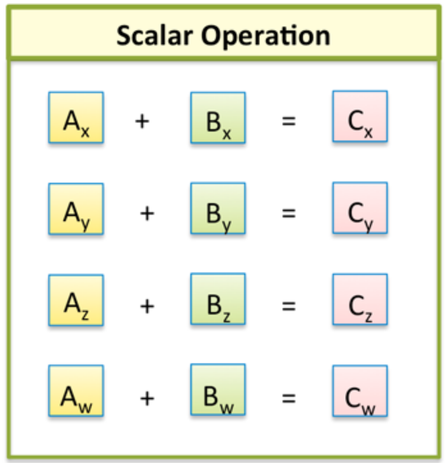


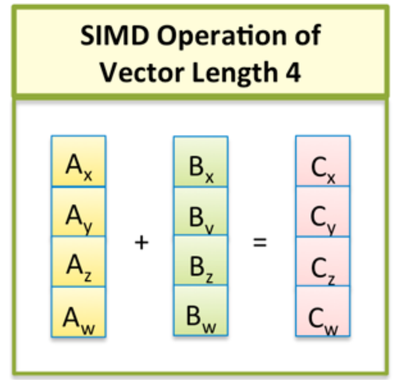


Single-Instruction Multiple-Data



Example for SIMD in data science:





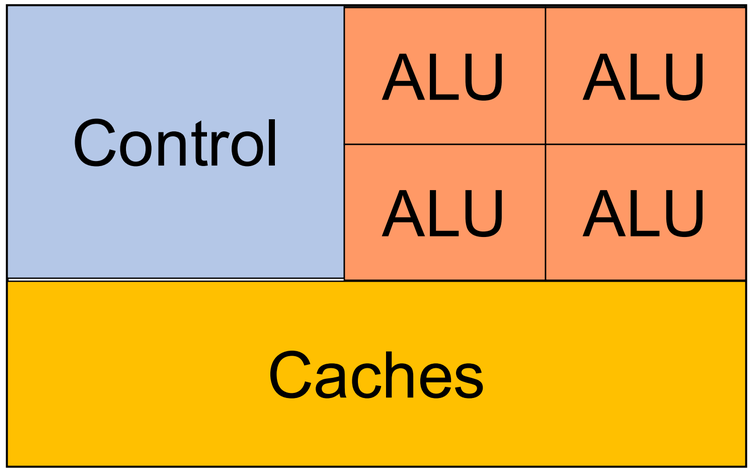
Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

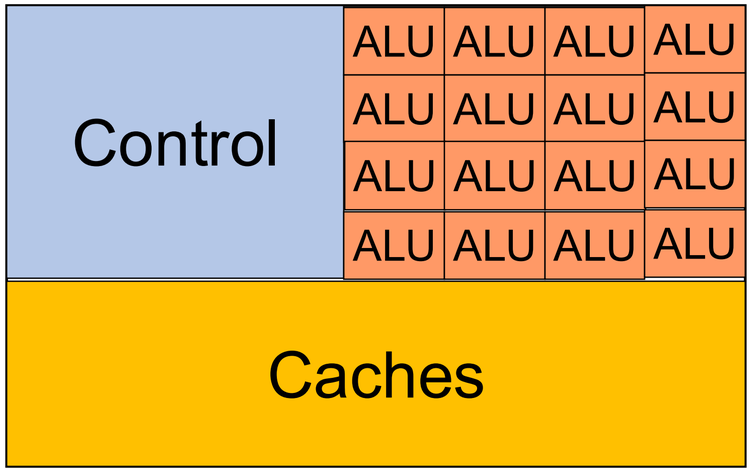
SIMD Generalizations

- 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 data processing at scale

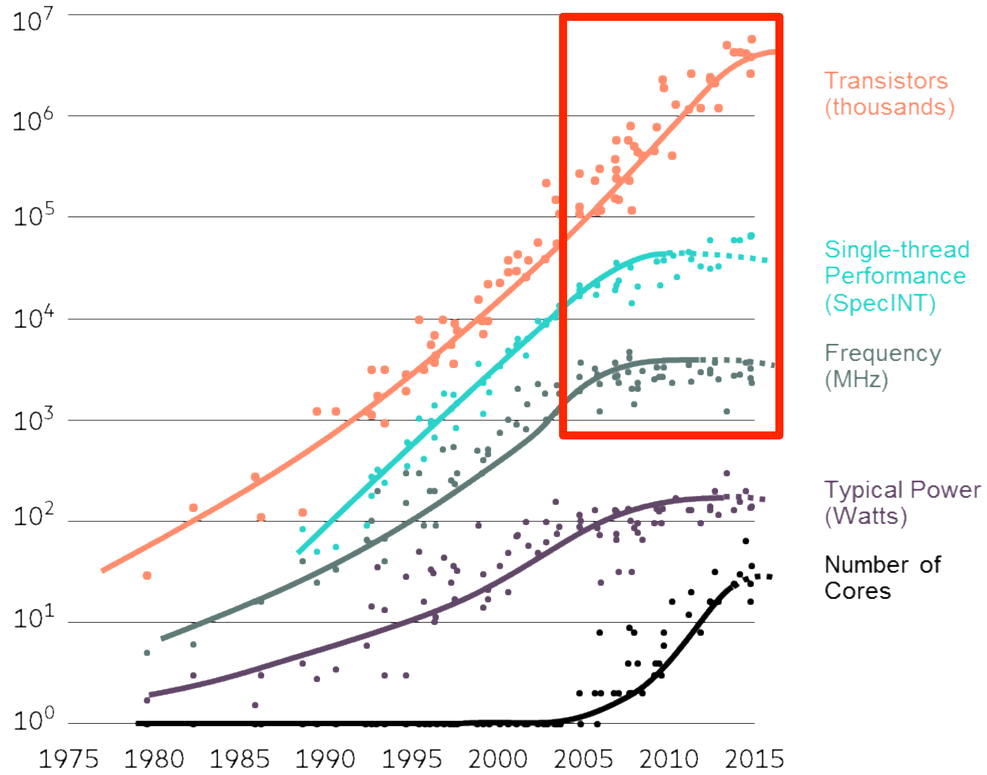
The Problem of Chip Design

If we're able to reduce to size of ALU while keeping its power





- That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?

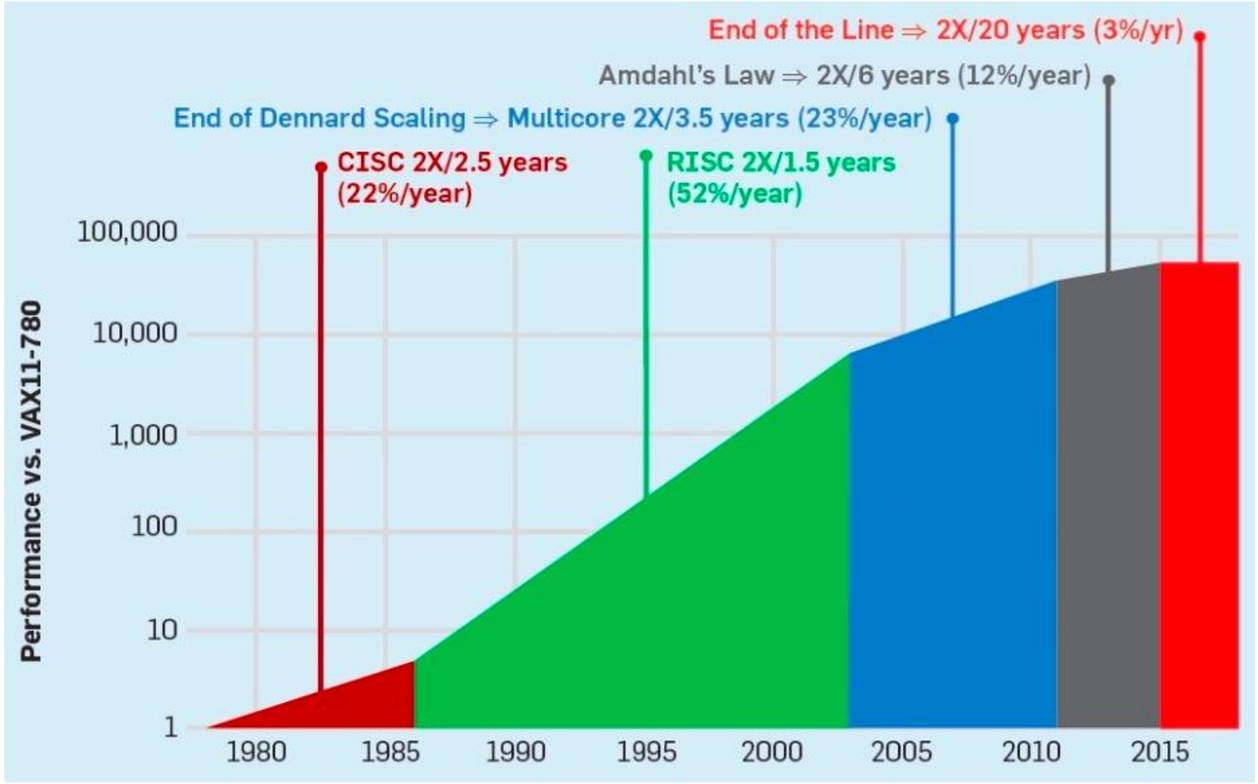


- That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?
- Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOPs-heavy programs like deep nets
- Motivated the rise of "accelerators" for some classes of programs

Chip Industry: 70nm -> 60nm -> 50nm -> … ->?

- Problem: this is not substantiable; there are also power/heat issues when you put more ALUs in a limited area (s.t. physics limitations)

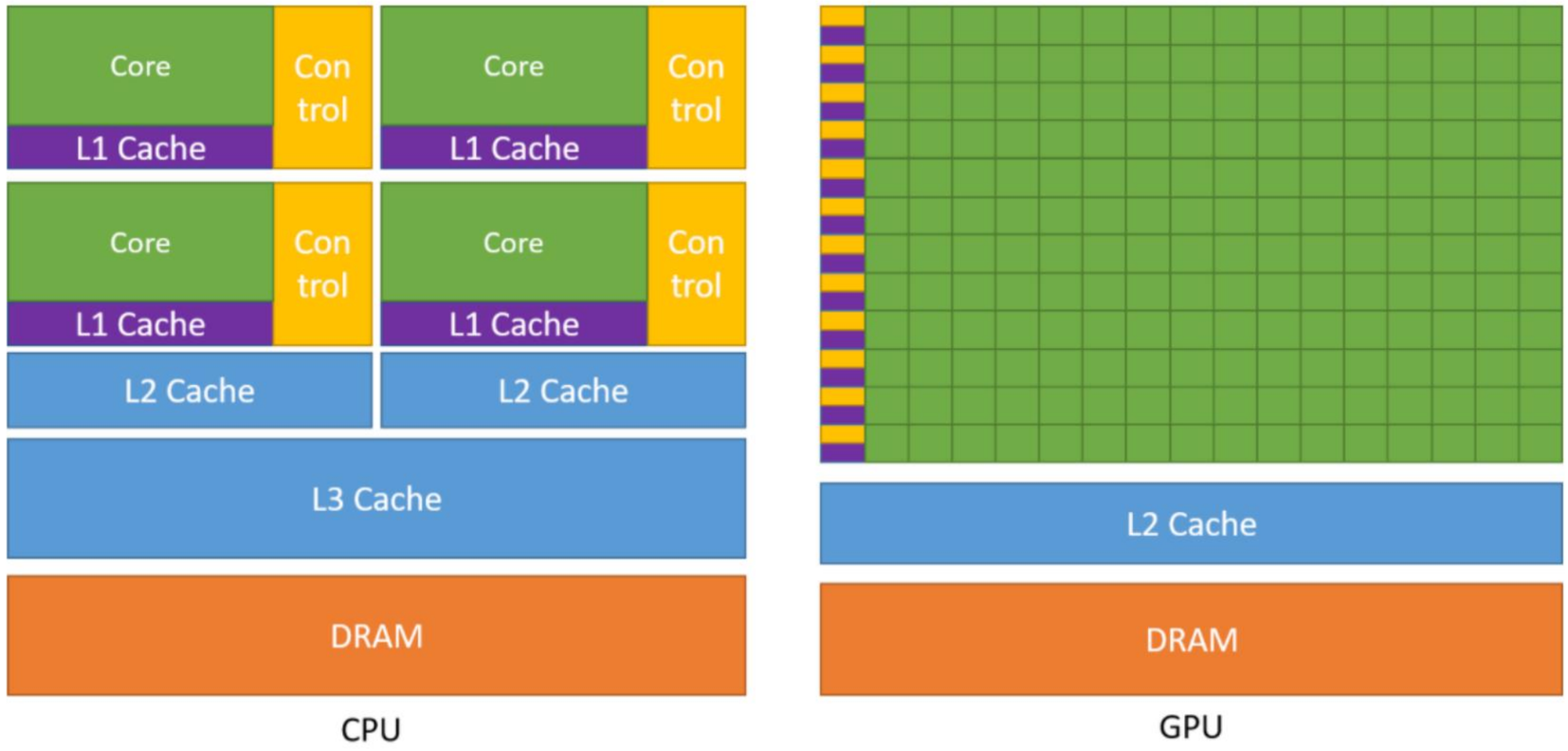
Chip Industry: Moore's Law Comes to an End



Option 1: Go to the quantum world

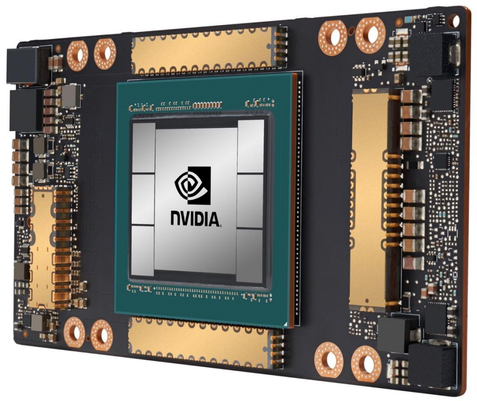
Option 2: Specialized hardware

Idea: How about we use a lot of weak/specialized cores



Hardware Accelerators: GPUs

- **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- Basic idea: Use tons of ALUs (but weak and more specialized); massive data parallelism (SIMD on steroids); now H100 offers ~980 TFLOPS for FP16!
- Popularized by NVIDIA in early 2000s for video games, graphics, and multimedia; now ubiquitous in DL
- CUDA released in 2007; later wrapper APIs on top: CuDNN, CuSparse, CuDF (RapidsAI), NCCL, etc.



Other Hardware Accelerators

- E.g.
  - Tensor Processing Unit (TPU)
  - An “application-specific integrated circuit” (ASIC) created by Google in mid 2010s; used for AlphaGo
- E.g.
  - B200 (projected release 2025): fp4 / fp8 Tensorcore
- E.g.
  - M3 max: mixing tensorcore and normal core

What Does It Mean by “Specialized” In accelerator world

In General:

- Functionality-specialized:
  - Can only compute certain computations: matmul, w/ sparsity
  - Mixing specialized cores with versatile cores
- Reduce precision
  - Floating point operations: fp32, fp16, fp8, int8, int4, …
- Tune the distribution of different components for specific workloads
  - SRAM, cache, registers, etc.

Comparing Modern Parallel Hardware

	Multi-core CPU	GPU	FPGA	ASICs (e.g., TPUs)
Peak FLOPS	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	Poor Fit	Potential exists but not mass market
Fitness for DL Inference?	Moderate	Moderate	Good Fit	Best Fit
Cloud Vendor Support	All	All	All	GCP

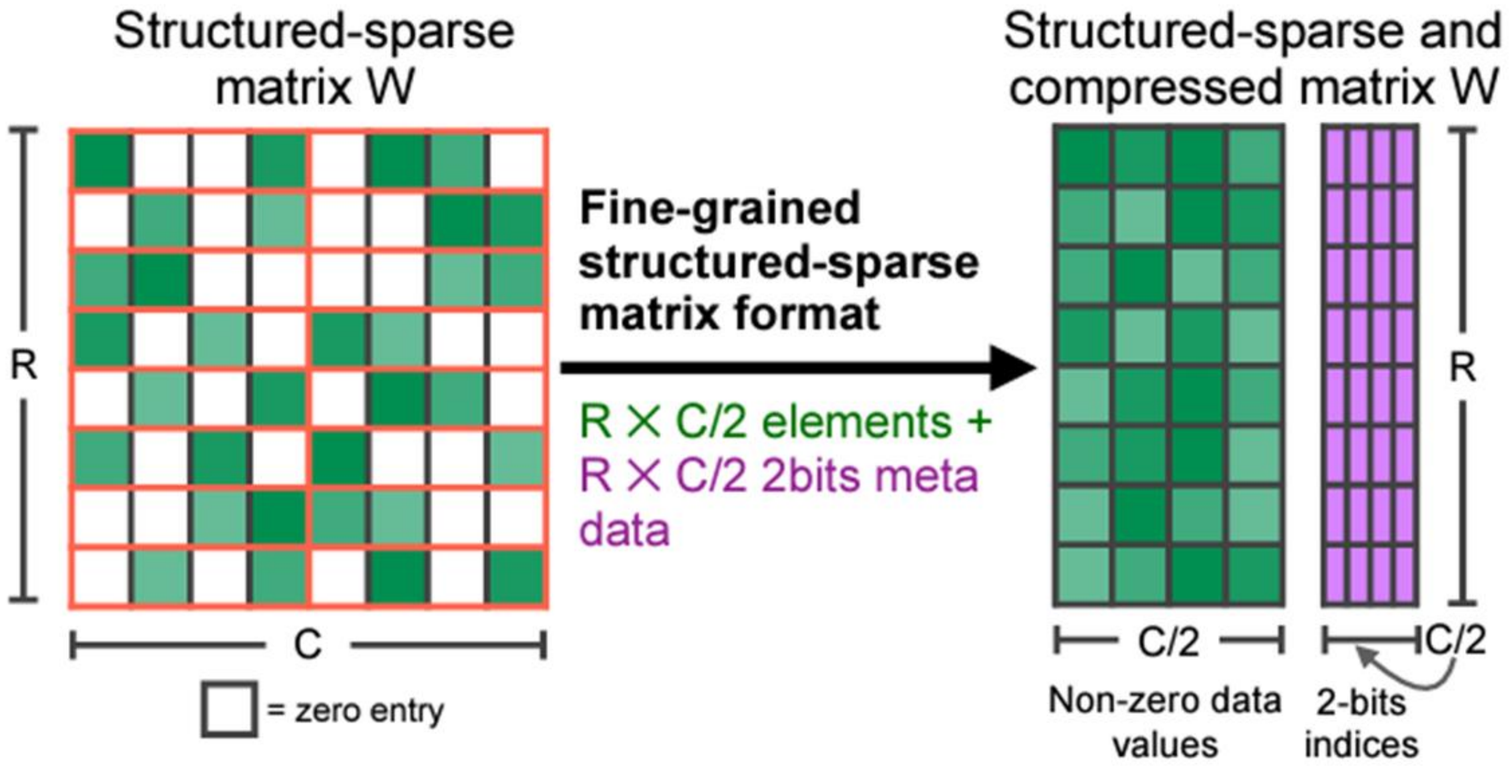
https://www.embedded.com/leveraging-fpgas-for-deep-learning/

Case Study 1: Nvidia GPU Specification

FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGS @ 10GB each
Form Factor	SXM
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA AI Enterprise	Add-on

* With sparsity

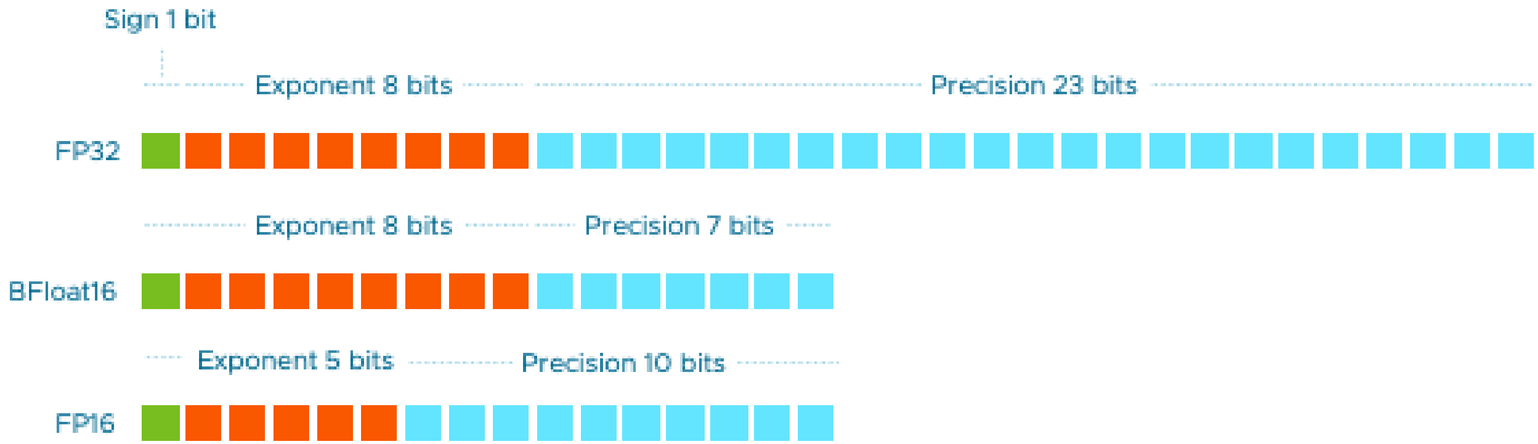
Case Study 1: Nvidia GPU Specification



Case Study 1: Nvidia GPU Specification

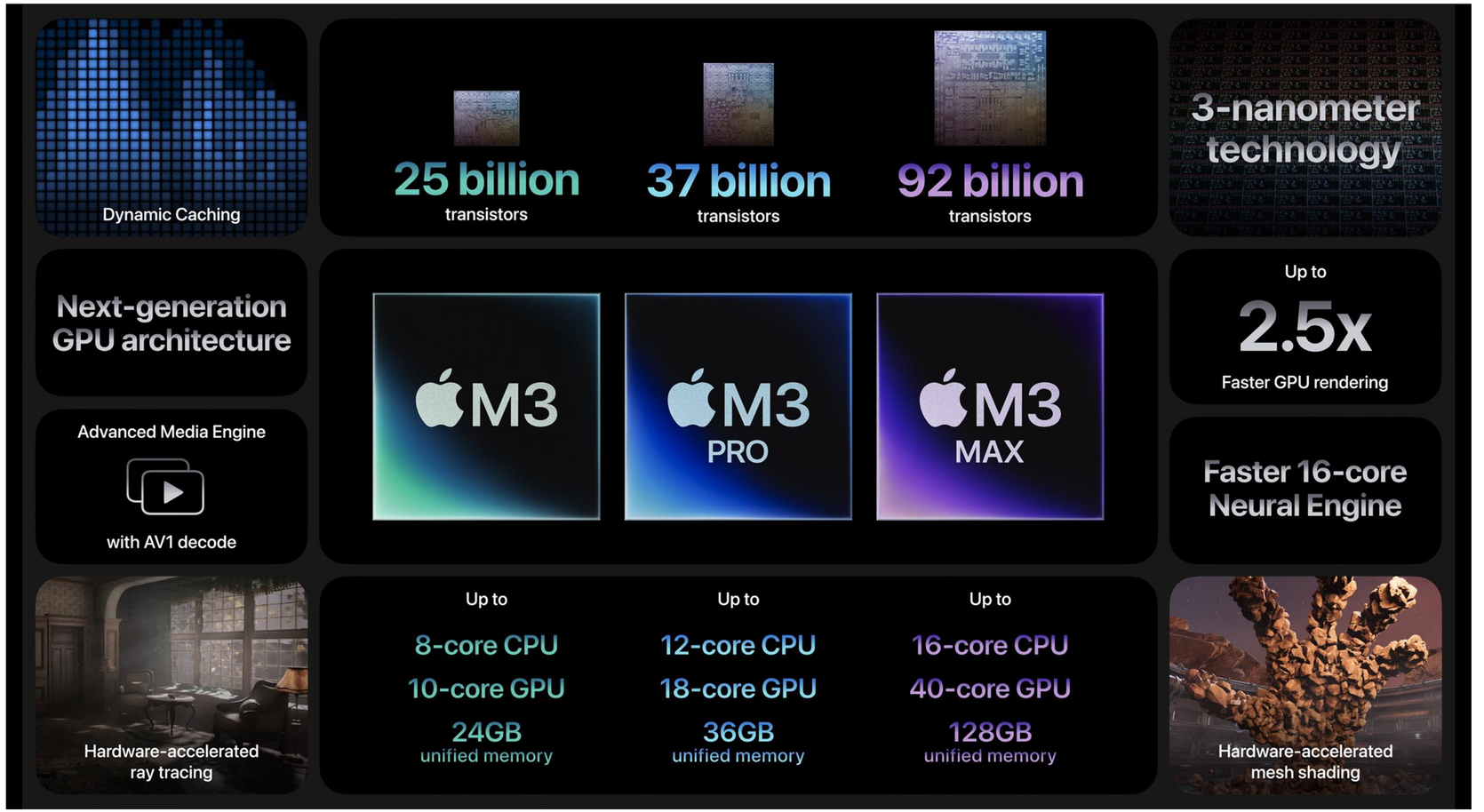
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGS @ 10GB each
Form Factor	SXM
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA AI Enterprise	Add-on

* With sparsity

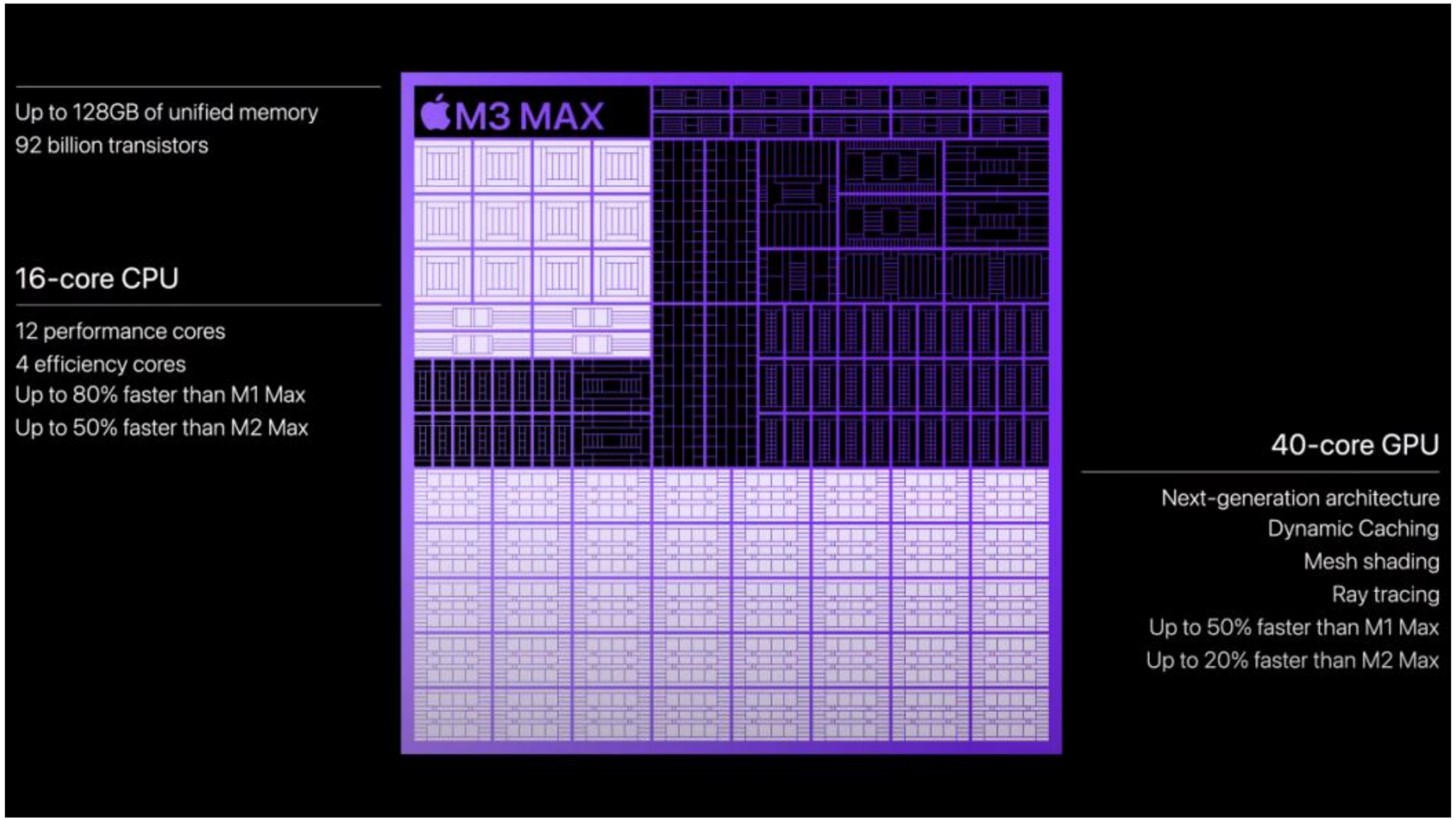


Question: why this could work in ML programs?

Case Study 2: Apple Silicon



Case Study 2: Apple Silicon Revealed



Specialized Hardware for DS/ML is a really good business

Market Summary > NVIDIA Corp

180.28 USD

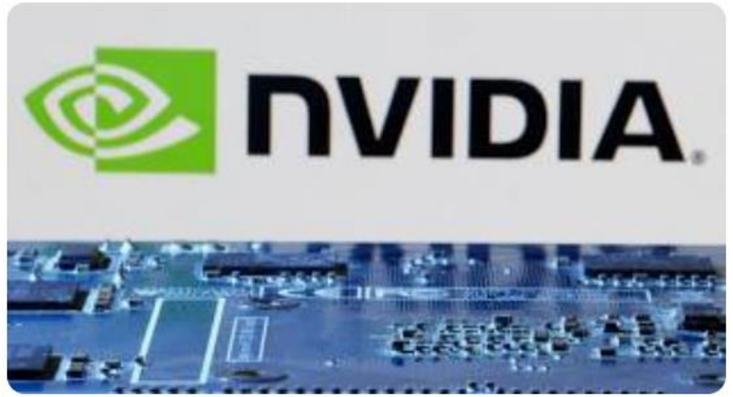
+180.24 (450,600.00%) ↑ all time

Closed: Oct 22, 7:59 PM EDT • Disclaimer

After hours 179.70 −0.58 (0.32%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	181.14	Mkt cap	4.38T	52-wk high	195.62
High	183.44	P/E ratio	51.31	52-wk low	86.63
Low	176.76	Div yield	0.022%	Qtrly Div Amt	0.010



Reuters

Nvidia outstrips Alphabet as third largest US company by market value

1 hour ago

Case Study 3: Leading Chip Startups

Summary

- Dataflow Graph
- Two major parallelisms:
  - Task parallelism -> partitioning the dataflow graph
  - Data parallel -> partitioning the data
- Data parallelism is ubiquitous, built in modern chips and everywhere.

Take-home Exercise

- Study B100 specification and compare it to H100
  - How nvidia claims another 2x from H100 -> B100?
- Study Apple M5 and compare it to M3