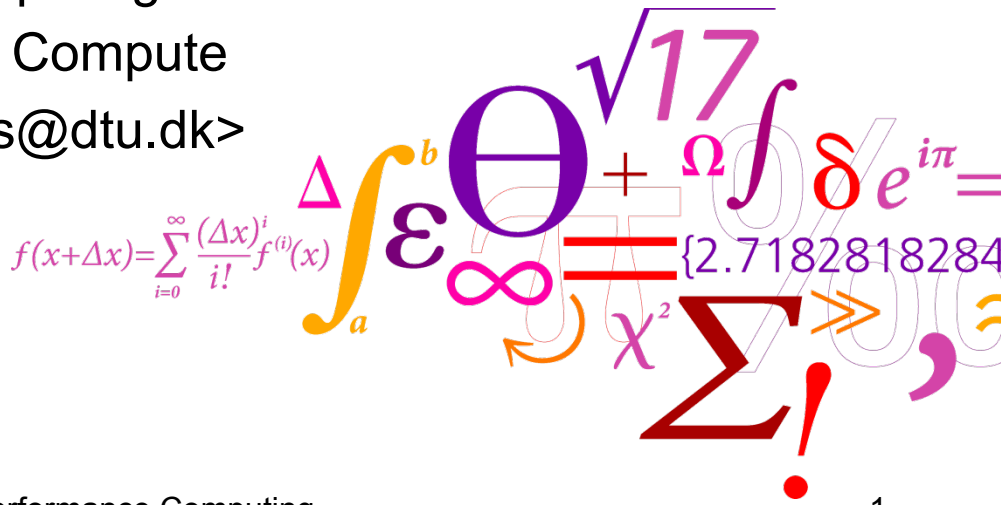
Introduction to GPU Computing

Hans Henrik Brandenborg Sørensen
DTU Computing Center
DTU Compute
<hhbs@dtu.dk>



Practicalities

- Teacher(s) - Week 3
  - Hans Henrik B. Sørensen
  - Nicolai
- Lectures / GPU Exercises – Mon-Tue
  - Learning by doing!
  - Complementary to Assignment 3 (should not be handed in)
- Assignment #3: GPU Matrix Multiplication and GPU Poisson Problem – Wed-Fri
  - Collaborate on developing the code
  - Write “individual reports”

Learning objectives for week 3

- This week we focus on learning
  - Interplay of computer components like CPU, GPU, caches, and memory
  - Choose the optimal hardware platform for a given problem
  - Write parallel programs with OpenMP and CUDA
  - Write efficient programs for multi-core processors and many-core GPUs / Multi-GPUs
- Pre-requisites: access to a CUDA enabled GPU (e.g. DCC or home-pc with Nvidia cc.>=7.0).

Overview for week 3

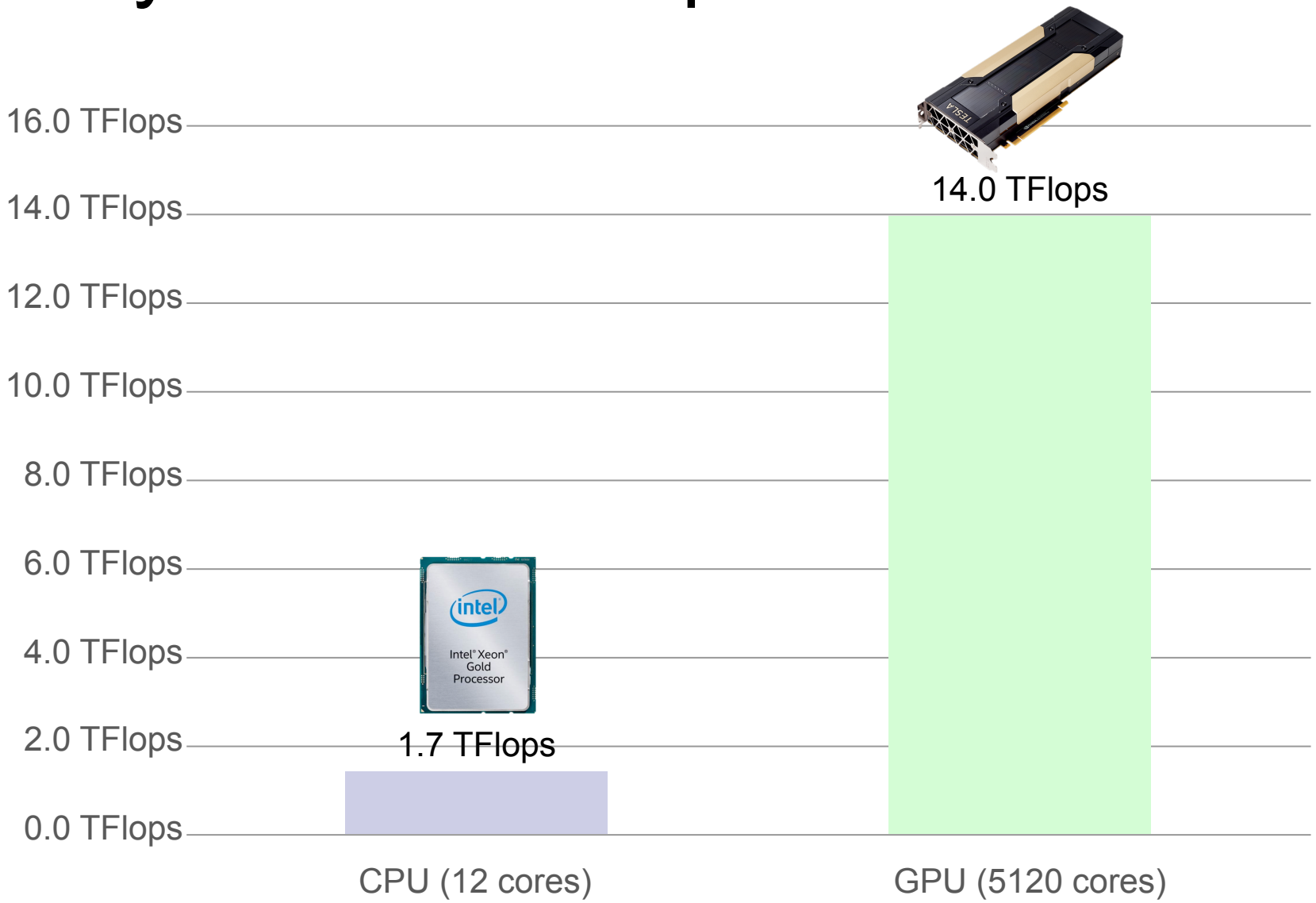
- GPU computing – Mon
  - Introduction to GPU computing
  - CUDA programming model
  - CUDA memory model
- Performance tuning of CUDA applications – Tue
  - Basic tuning techniques
  - Memory access optimization
  - Control flow and instruction optimizations
  - Profiling tools / demo
- Assignment 3 – Wed

Introduction to GPU computing

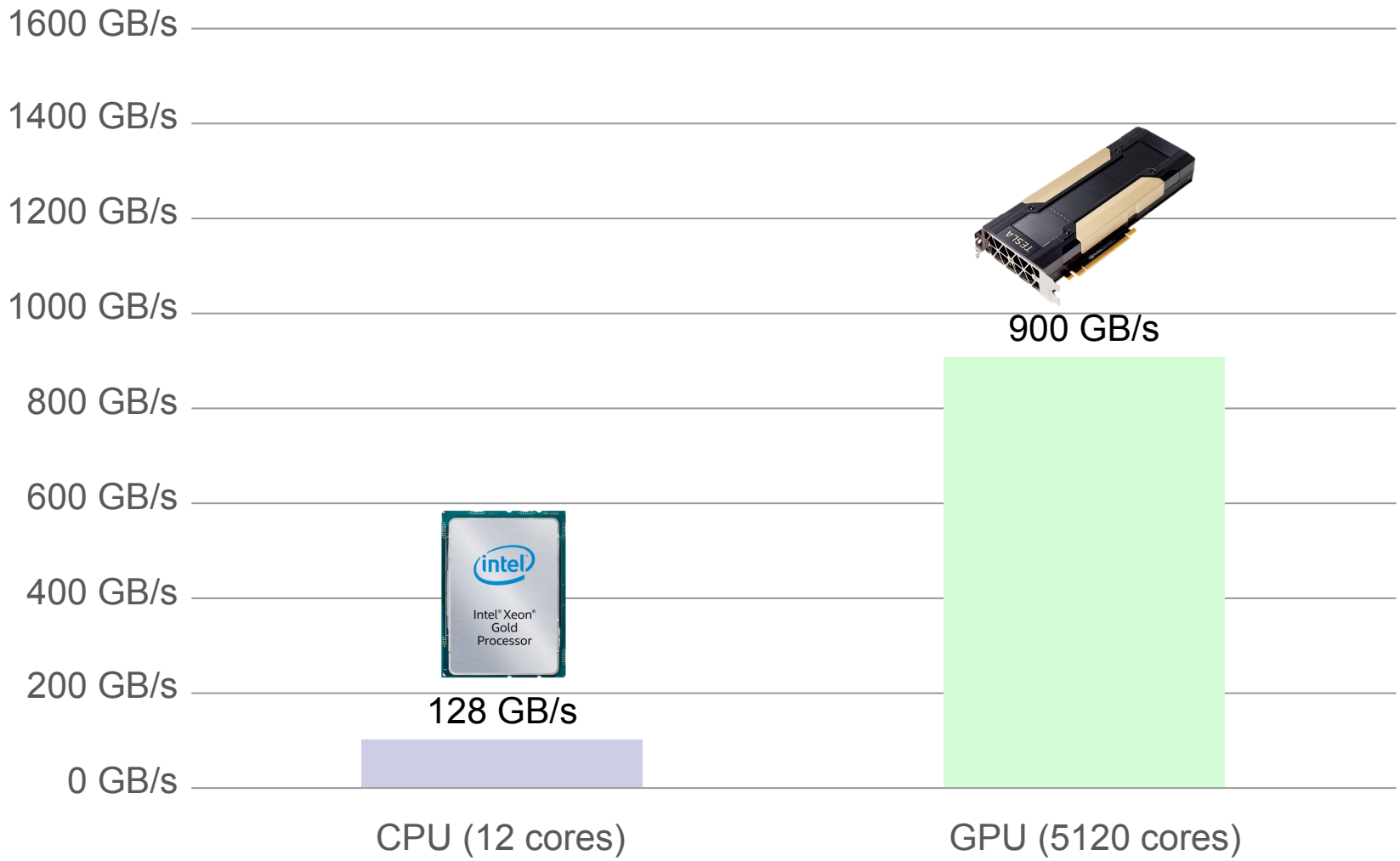
Outline

- Why are GPUs important in HPC?
- Why are GPUs different from CPUs?
- GPUs as accelerators
- GPU hardware for 02614 course

Why are GPUs important in HPC?



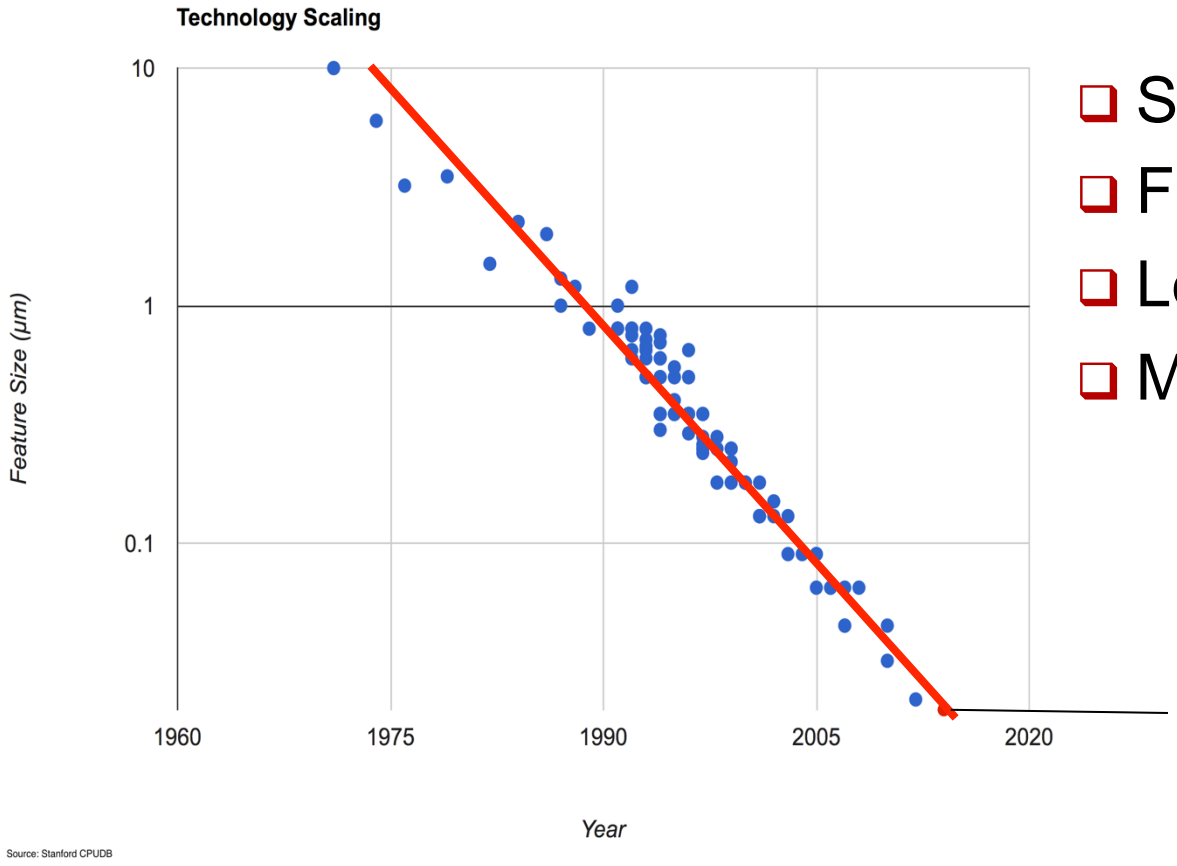
Why are GPUs important in HPC?



Why are GPUs different from CPUs?

Recap from week 1: Good news

- Transistor size over the years; still decreases

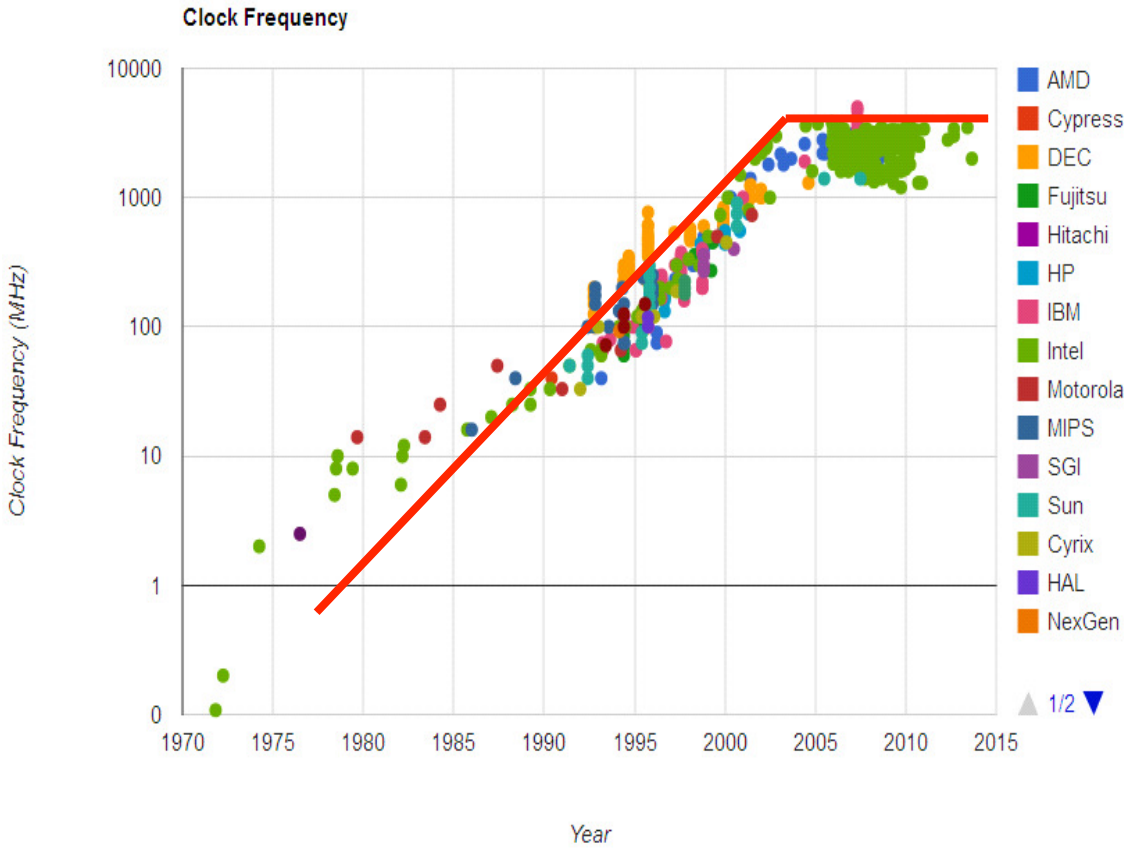


- Smaller
- Faster
- Less power
- More on chip

Currently best: 12 nm

Recap from week 1: Bad news

- Clock speed over the years; stagnated since 2005



- Increasing over many years
- However, over the last decade the clocks speeds have essentially remained constant

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power
- “Nard scaling”
  - Reduce transistor voltage to counter higher clock speed
  - Broke down in 2005 because of weakened current in the wires (need to distinguish 0 and 1)

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power
- "Nard scaling"
  - Reduce transistor voltage to counter higher clock speed
  - Broke down in 2005 because of weakened current in the wires (need to distinguish 0 and 1)
- What matters today: # operations per Watt!
- Trade-off favors slower simpler processors
  - More operations per watt
  - Frequency kept low

Building a modern processor

- What is the goal?

Building a modern processor

- What is the goal? Roughly two choices...

A.	B.
Latency (time to complete a task) [e.g. seconds]	Throughput (tasks completed per unit time) [e.g. jobs/hour]

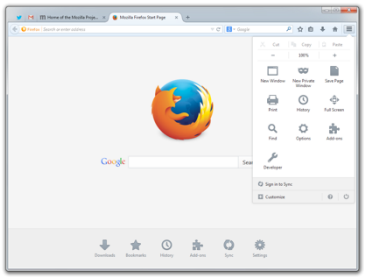
Building a modern processor

- What is the goal? Roughly two choices...

A.	B.
Latency (time to complete a task) [e.g. seconds]	Throughput (tasks completed per unit time) [e.g. jobs/hour]

- Unfortunately these goals are not always aligned

CPUs target latency (traditionally)



- Usual task of traditional CPUs
  - Desktop applications / OS
    - Lightly threaded
    - Lots of branches
    - Lots of (indirect) memory accesses

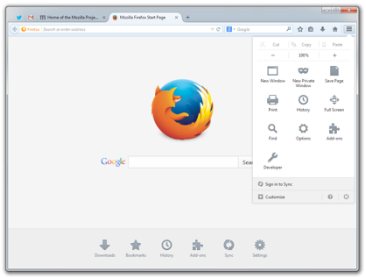
Mac OS

Windows 10

Microsoft Office

- CPUs try to minimize the time to complete a particular task – often to support user interaction!

CPUs target latency (traditionally)



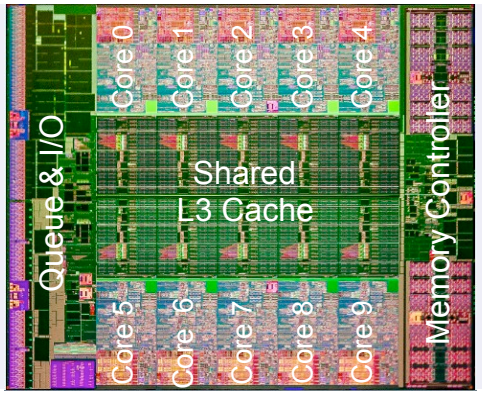
- Usual task of traditional CPUs
  - Desktop applications / OS
    - Lightly threaded
    - Lots of branches
    - Lots of (indirect) memory accesses

Mac OS

Windows 10

Microsoft Office

- CPUs try to minimize the time to complete a particular task – often to support user interaction!
- Complex control hardware
  - \+ Flexibility in performance
  - \+ Lightly parallel
  - − Expensive in terms of power



GPUs target throughput

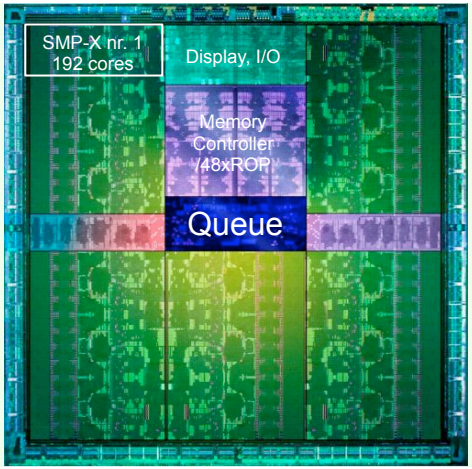
- GPUs are designed to compute pixels – fast!
  - Rendering video games in real-time
  - Play HD movies on smart phones
  - Render visual effects for movies…

- More concerned about the number of pixels per second than the latency of any particular pixel!

GPUs target throughput

- GPUs are designed to compute pixels – fast!
  - Rendering video games in real-time
  - Play HD movies on smart phones
  - Render visual effects for movies…

- More concerned about the number of pixels per second than the latency of any particular pixel!
- Simpler control hardware
  - + More transistors for computation
  - + Power efficient
  - − Highly parallel
  - − More restrictive in performance



Nvidia GPU architectures

- Four generations of high-performance GPUs:

	# GPUs	Name	Year	Architecture	CUDA cap.	CUDA cores	Clock MHz	Mem GiB	SP peak GFlops	DP peak GFlops	Peak GB/s
Fermi	8	Tesla M2050	2012	GF100 (Fermi)	2.0	448	575	2.62	1030	515	148.4
	6	Tesla M2070Q	2012	GF100 (Fermi)	2.0	448	575	5.25	1030	515	150.3
	2	*GeForce GTX 680	2012	GK104-400 (Kepler)	3.0	1536	1058	1.95	3090	128	192.2
Kepler	3	Tesla K20c	2013	GK110 (Kepler)	3.5	2496	745	4.63	3524	1175	208
	5	Tesla K40c	2013	GK110B (Kepler)	3.5	2880	745 / 875	11.17	4291 / 5040	1430 / 1680	288
	8	Tesla K80c (dual)	2014	GK210 (Kepler)	3.7	2496	562 / 875	11.17	2796 / 4368	932 / 1456	240
Maxwell	1	*GeForce GTX TITAN X	2015	GM200-400 (Maxwell)	5.2	3072	1076	11.92	6144	192	336
Pascal	8	*TITAN X	2016	GP102 (Pascal)	6.1	3584	1417 / 1531	11.90	10157 / 10974	317.4 / 342.9	480

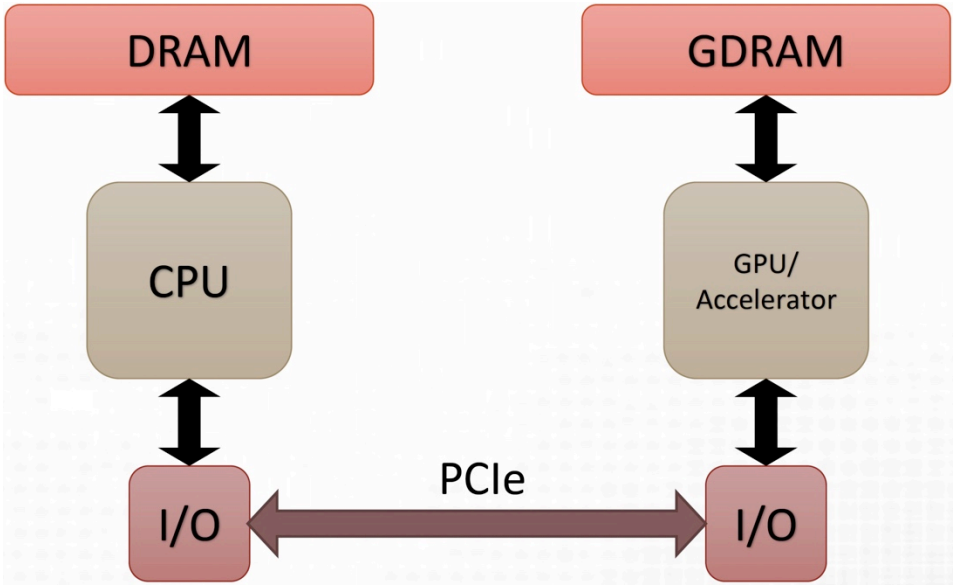
Source: http://www.hpc.dtu.dk/?page_id=2129

- More cores, similar clock rate, more bandwidth.

GPUs as accelerators

GPUs as accelerators

- Problem: Still require OS, I/O, and scheduling
- Solution: “Hybrid system”
  - CPU provides management
  - Accelerators such as GPUs provide compute power

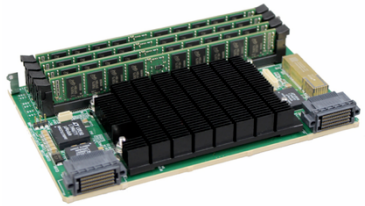


Types of accelerators

- GPUs
  - HPC high-end versions – Tesla branch
  - DP downgraded versions – Titan branch

- Intel Xeon Phis
  - Many Integrated Cores (MIC) architecture
  - Based om Pentium 4 with wide vectors
  - Closer to traditional CPU / same compiler

- Custom many-core processors
  - Japan / China



Accelerators in Top500

DTU

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P, NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209

RISC processor with 260 cores

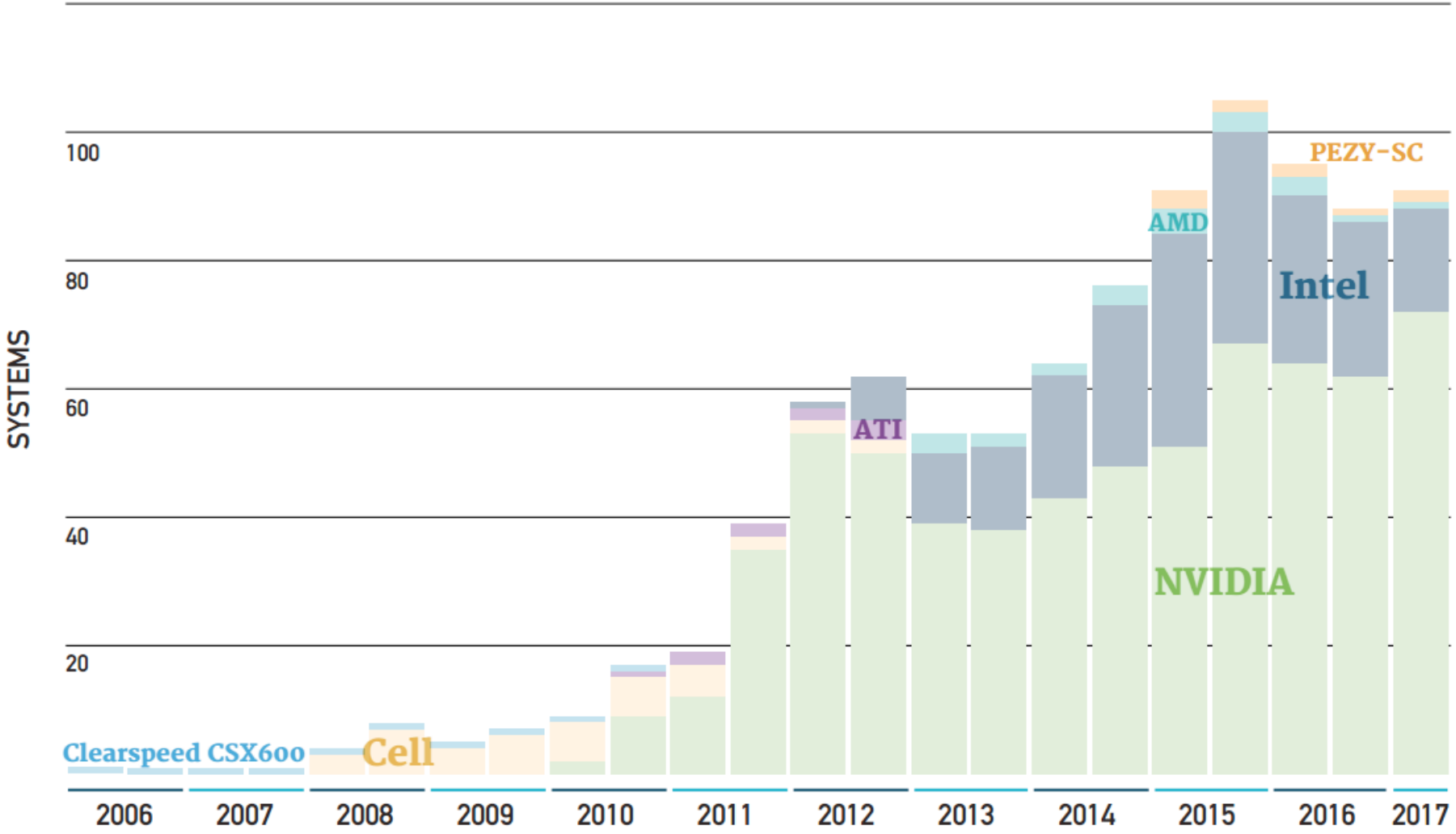
48,000 Xeon Phi cards

Nvidia Tesla (Pascal)

2048 cores / 8.2 TFlops SP

Nvidia Tesla (Kepler)

Accelerators in Top500

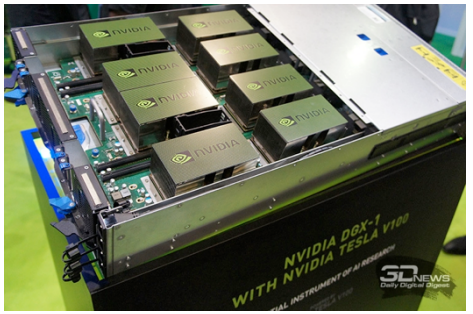


Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI

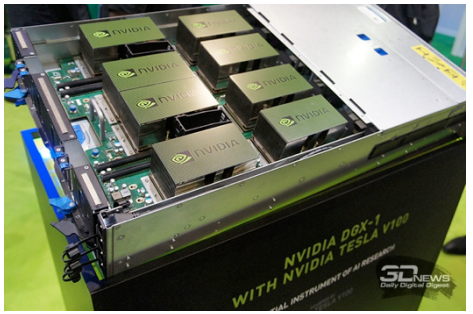
Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI
- "Super-nodes": Nvidia DGX-1
  - Eight tightly linked high-end GPUs
  - 40,960 cores / 960 TFlops in 1 node



Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI
- "Super-nodes": Nvidia DGX-1
  - Eight tightly linked high-end GPUs
  - 40,960 cores / 960 TFlops in 1 node
- Communication costs are increasing
  - Synchronization-reducing algorithms
  - Communication lower-bound algorithms

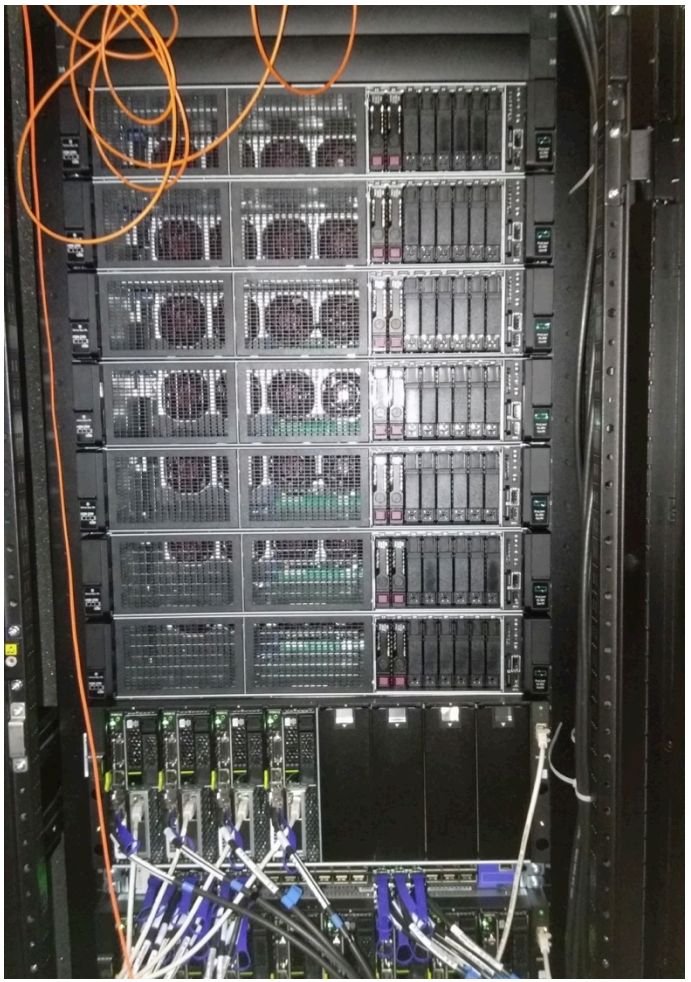


GPU hardware for this course

Volta cluster for this course

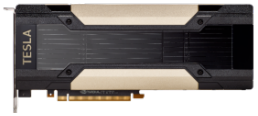
Delivered: December 19, 2017

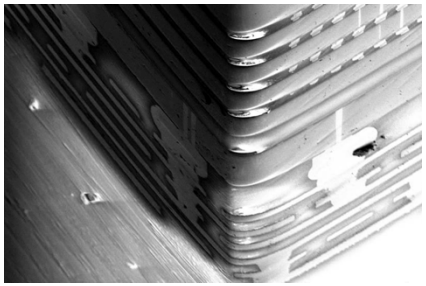
1 HPE node:
2 x Intel Xeon Gold 6126 CPU @ 2.60GHz (12 cores)
2 x Tesla V100-PCIE-16GB (5120 cores)
192 GB DDR4 @ 2666MHz



7 HPE nodes:
Theoretical peak: 109,9 TFlops (FP64)

Tesla V100



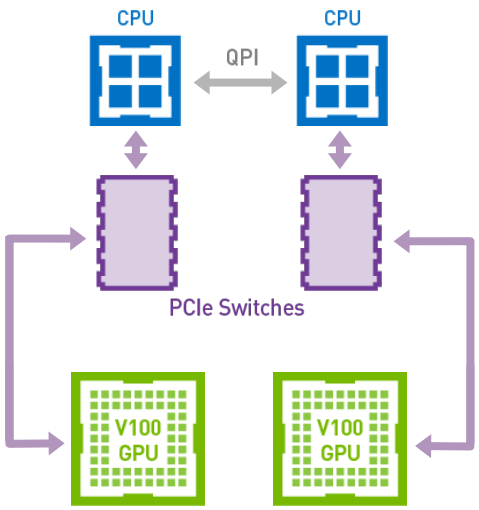


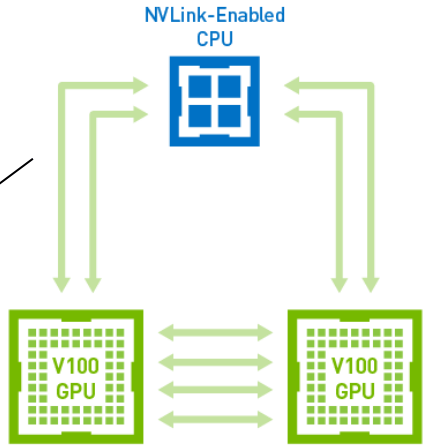
HBM2 memory (Samsung)

	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	16 GB HBM2	
Memory Bandwidth	900 GB/sec	
ECC	Yes	
Interconnect Bandwidth	32 GB/sec	300 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

79.436,- DKK

Læg i kurv





Volta GV100 GPU

Hardware hierarchy

- Nvidia GPU
  - Processing Unit (L2 cache)
  - GPC – Graphics Processing Cluster
  - TPC – Texture Processing Cluster
  - SM – Streaming Multiprocessor (L1 cache)
  - Processing block (Instr. cache)
  - Core

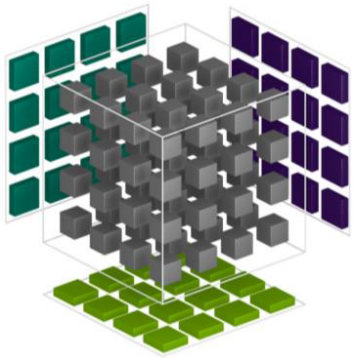
Hardware hierarchy

- Nvidia GPU
  - Processing Unit (L2 cache)
  - GPC – Graphics Processing Cluster
  - TPC – Texture Processing Cluster
  - SM – Streaming Multiprocessor (L1 cache)
  - Processing block (Instr. cache)
  - Core
- CPU (typical)
  - Processing Unit (L3 cache)
  - Core (L2 cache / L1 cache / Instr. cache)

Volta GV100 GPU

- GV100 = 80 SMs
- Each SM has
  - 64 FP32 cores
  - 64 INT32 cores
  - 32 FP64 cores
  - 8 Tensor cores
  - 256 KB register file
  - 128 KB L1 cache
  - Up to 2048 active threads

Tensor cores

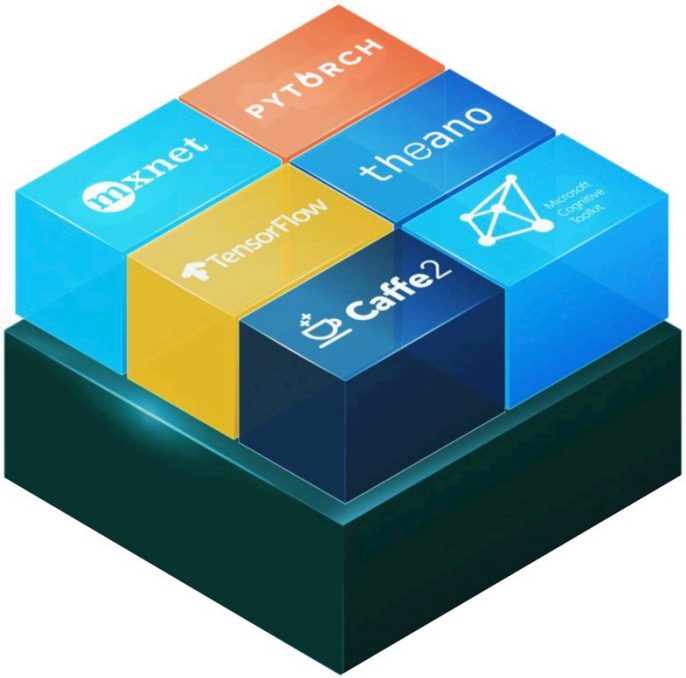


- For matrix operations
  - 4x4 matrices
  - Mixed precision (FP16 input data, FP32 accumulation)
  - Each does 64 floating-point FMA per clock
  - Data-paths custom-crafted to reduce area and power

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

Using Tensor cores



cuDNN, cuFFT, cuBLAS

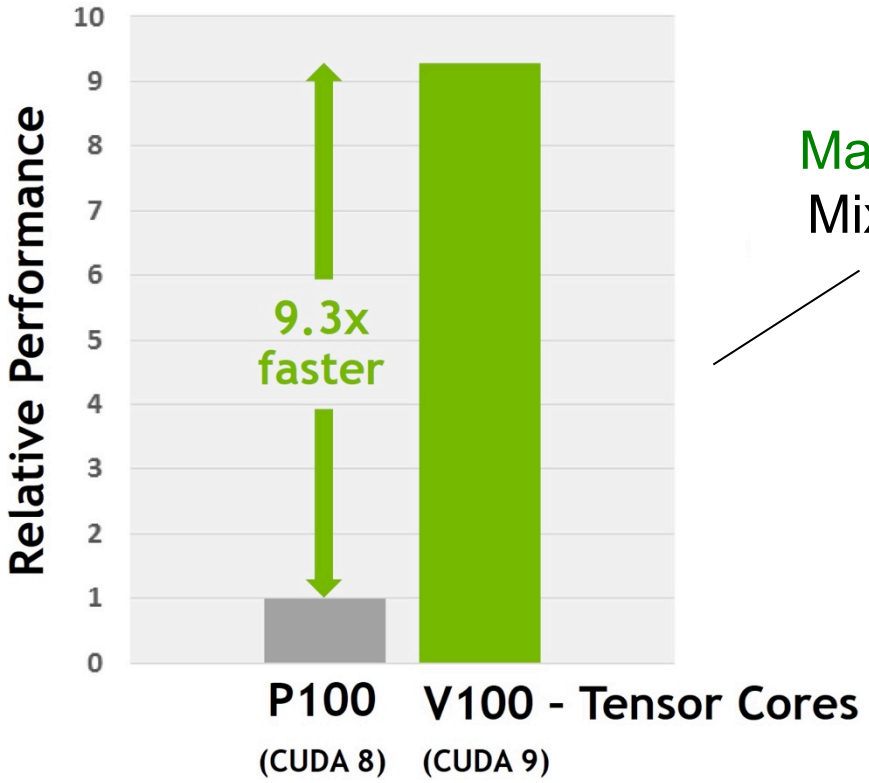
Volta optimized frameworks and libraries

```
__device__ void tensor_op_16_16_16(
float *d, half *a, half *b, float *c)
{
  wmma::fragment<matrix_a, …> Amat;
  wmma::fragment<matrix_b, …> Bmat;
  wmma::fragment<matrix_c, …> Cmat;
  wmma::load_matrix_sync(Amat, a, 16);
  wmma::load_matrix_sync(Bmat, b, 16);
  wmma::fill_fragment(Cmat, 0.0f);
  wmma::mma_sync(Cmat, Amat, Bmat,
Cmat);
  wmma::store_matrix_sync(d, Cmat, 16,
                          wmma::row_major);
}
```

CUDA C++ API (16x16 matrices)

A giant leap for Deep Learning

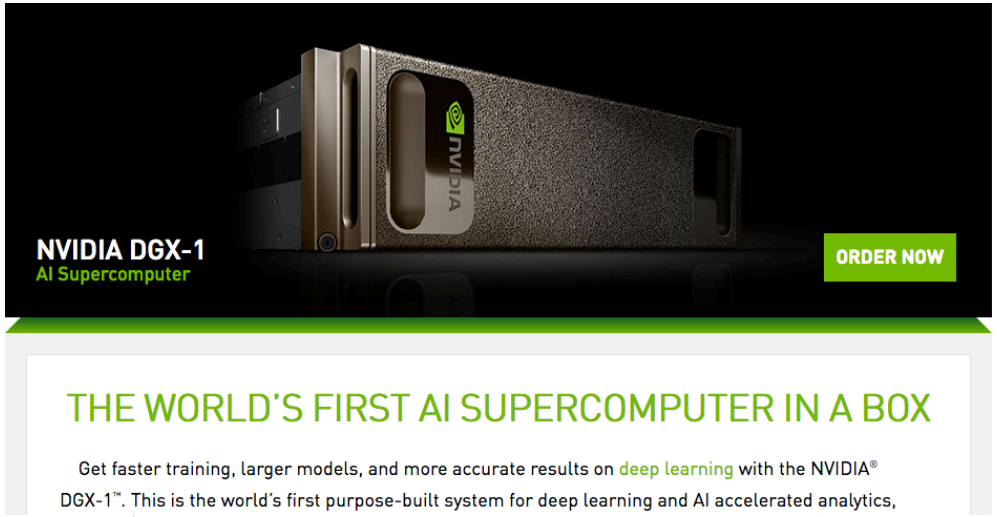
- In practice: Used to perform much larger GEMM matrix operations, built from the 4x4 operations

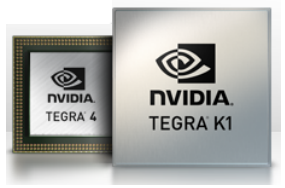


cuBLAS
Matrix multiply (M=N=K=2048)
Mixed precision (FP16 input, FP32 compute)

At the core of neural network training and inferencing

Google trends: GPUs still going up





End of lecture