a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software
2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

MCS 572 Lecture 16
Introduction to Supercomputing
Jan Verschelde, 2 October 2024

a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software
2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

general purpose graphics processing units

Thanks to the industrial success of video game development graphics processors became faster than general CPUs.

General Purpose Graphic Processing Units (GPGPUs) are available, capable of double floating point calculations.

Accelerations by a factor of 10 with one GPGPU are not uncommon.

Comparing electric power consumption is advantageous for GPGPUs.

Thanks to the popularity of the PC market, millions of GPUs are available – every PC has a GPU. This is the first time that massively parallel computing is feasible with a mass-market product.

Example: Actual clinical applications on magnetic resonance imaging (MRI) use some combination of PC and special hardware accelerators.

topics on accelerated parallelism

Following textbook and NVIDIA programming guide:

1. architecture, programming models, scalable GPUs
2. introduction to CUDA and data parallelism
3. CUDA thread organization, synchronization
4. CUDA memories, reducing memory traffic
5. coalescing and applications of GPU computing

We follow the book by David B. Kirk and Wen-mei W. Hwu:
Programming Massively Parallel Processors. A Hands-on Approach.
Elsevier 2010; 4th edition, 2023, with Izzat El Hajj as 3rd author.

expectations

Expectations?

1. design of massively parallel algorithms
2. understanding of architecture and programming
3. software libraries to accelerate applications

Key questions:

1. Which problems may benefit from GPU acceleration?
2. Rely on existing software or develop own code?
3. How to mix MPI, multicore, and GPU?

The textbook authors use the peach metaphor:

- much of the application code will remain sequential;
- GPUs can dramatically improve easy to parallelize code.

a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software
2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

equipment: hardware and software

Microway workstation `pascal` (acquired in 2016):

- NIVDIA P100 general purpose graphics processing unit
  1. number of CUDA cores: 3,584 (56 $\times$ 64)
  2. frequency of CUDA cores: 405MHz
  3. double precision floating point performance: 4.7 Tflops (peak)
  4. single precision floating point performance: 9.3 Tflops (peak)
  5. total global memory: 16275 MBytes
- CUDA programming model with `nvcc` compiler.

Two Intel E5-2699v4 (2.20 GHz 22 cores) CPUs:

- 2.20 GHz $\times$ 8 flops/cycle = 17.6 GFlops/core;
- 44 core $\times$ 17.6 GFlops/core = 774.4 GFlops.

$\Rightarrow$ 4700/774.4 = 6.07. One P100 is as strong as 6 $\times$ 44 = 264 cores.

CUDA stands for Compute Unified Device Architecture, is a general purpose parallel computing architecture introduced by NVIDIA.

pascal versus volta

NVIDIA P100 16GB “Pascal” Accelerator

- 3,586 CUDA cores, 3,586 = 56 SM $\times$ 64 cores/SM
- GPU max clock rate: 1329 MHz (1.33 GHz)
- 16GB Memory at 720GB/sec peak bandwidth
- peak performance: 4.7 TFLOPS double precision

NVIDIA V100 “Volta” Accelerator

- 5,120 CUDA cores, 5,120 = 80 SM $\times$ 64 cores/SM
- GPU max clock rate: 1912 MHz (1.91 GHz)
- 32GB Memory at 870GB/sec peak bandwidth
- peak performance: 7.9 TFLOPS double precision

volta versus ampere

NVIDIA V100 “Volta” Accelerator

- 5,120 CUDA cores, 5,120 = 80 SM $\times$ 64 cores/SM
- GPU max clock rate: 1912 MHz (1.91 GHz)
- Memory clock rate: 850 Mhz
- 32GB Memory at 870GB/sec peak bandwidth
- peak performance: 7.9 TFLOPS double precision

NVIDIA A100 “Ampere” Accelerator

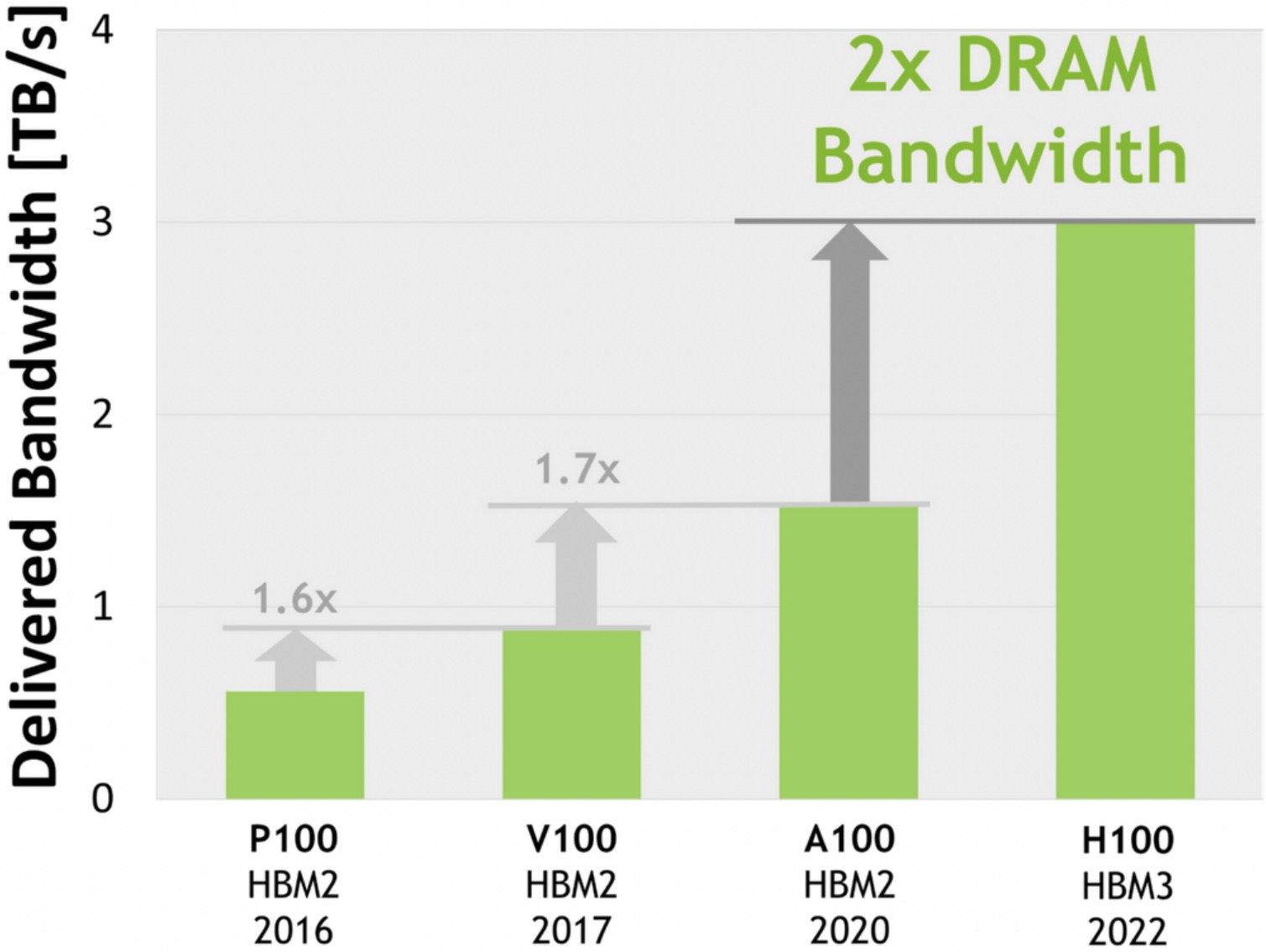
- 6,912 CUDA cores, 6,912 = 108 SM $\times$ 64 cores/SM
- GPU max clock rate: 1410 MHz (1.41 GHz)
- Memory clock rate: 1512 Mhz
- 80GB Memory at 1.94TB/sec peak bandwidth
- peak performance: 9.7 TFLOPS double precision
- **peak FP64 Tensor Core performance: 19.5 TFLOPS**

NVIDIA whitepapers

The full specifications are described in whitepapers available at the web site of NVIDIA:

- NVIDIA GeForce GTX 680, 2012.
- NVIDIA Tesla P100, 2016.
- NVIDIA Tesla V100 GPU Architecture, August 2017.
- NVIDIA A100 Tensor Core GPU Architecture, 2020.
- NVIDIA Ampere GA102 GPU Architecture, 2021.
- NVIDIA H100 Tensor Core GPU Architecture, 2022.

evolution of bandwidth



evolution of core counts

GPU Features	NVIDIA A100	NVIDIA H100 SXM5	NVIDIA H100 PCIe
GPU Architecture	NVIDIA Ampere	NVIDIA Hopper	NVIDIA Hopper
GPU Board Form Factor	SXM4	SXM5	PCIe Gen 5
SMs	108	132	114
TPCs	54	66	57
FP32 Cores / SM	64	128	128
FP32 Cores / GPU	6912	16896	14592
FP64 Cores / SM (excl. Tensor)	32	64	64
FP64 Cores / GPU (excl. Tensor)	3456	8448	7296
INT32 Cores / SM	64	64	64
INT32 Cores / GPU	6912	8448	7296
Tensor Cores / SM	4	4	4
Tensor Cores / GPU	432	528	456
GPU Boost Clock[2] for FP8, FP16, BF16, TF32 Tensor Core Ops	1410 MHz	1830 MHz	1620 MHz
GPU Boost Clock[2] for FP64 Tensor Core Ops, FP32 and FP64 non-Tensor Core Ops	1410 MHz	1980 MHz	1755 MHz

evolution of peak floating-point performance

non-Tensor

GPU Features	NVIDIA A100	NVIDIA H100 SXM5	NVIDIA H100 PCIe
GPU Architecture	NVIDIA Ampere	NVIDIA Hopper	NVIDIA Hopper
Peak FP16 TFLOPS (non-Tensor)	78	133.8	102.4
Peak BF16 TFLOPS (non-Tensor)	39	133.8	102.4
Peak FP32 TFLOPS (non-Tensor)	19.5	66.9	51.2
Peak FP64 TFLOPS (non-Tensor)	9.7	33.5	25.6

Pascal: 4.7, Volta: 7.9, Ampere: 9.7, Hopper: 25.6 TFLOPS.

a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software

2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

comparing flops on GPU and CPU

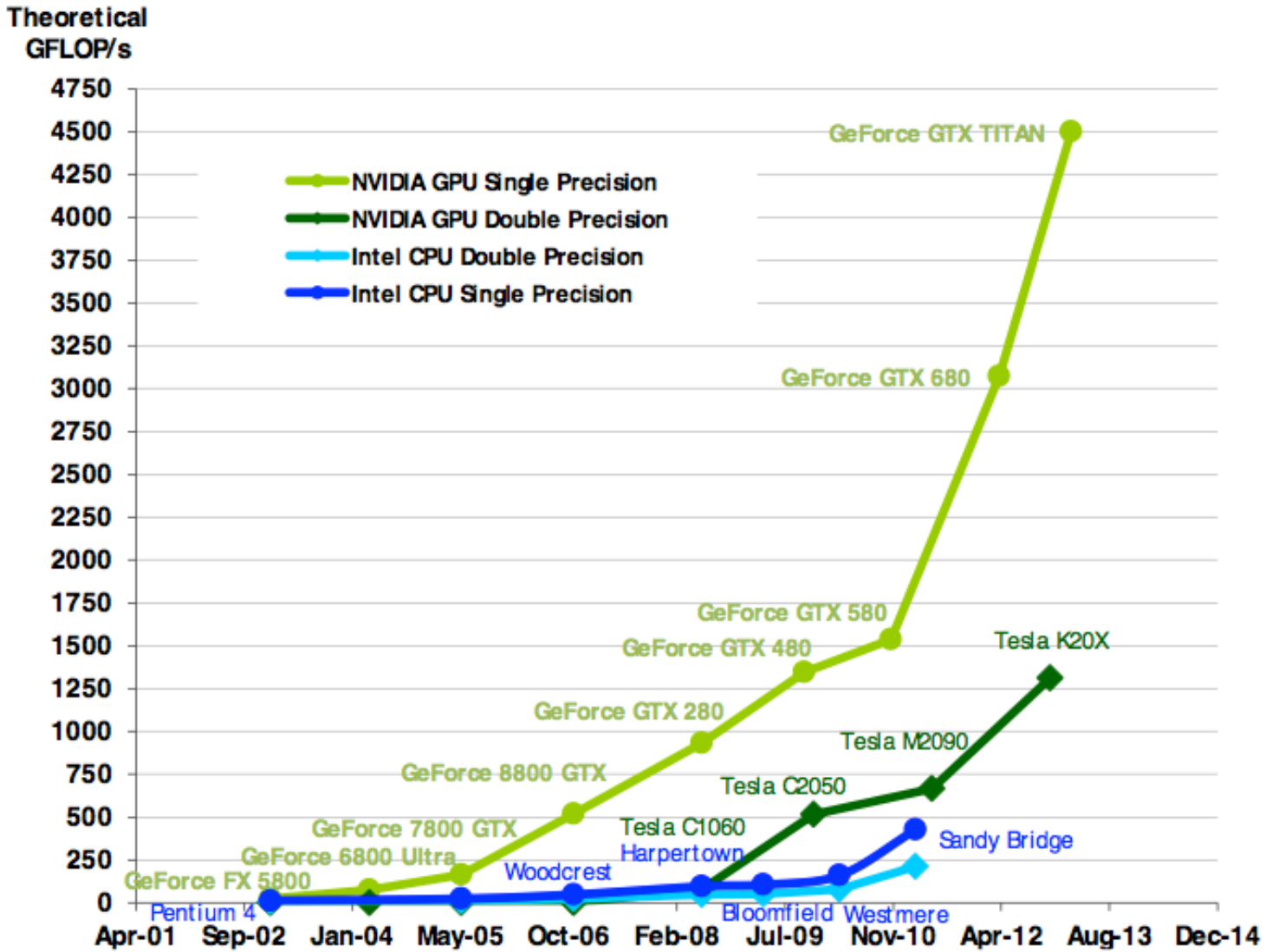
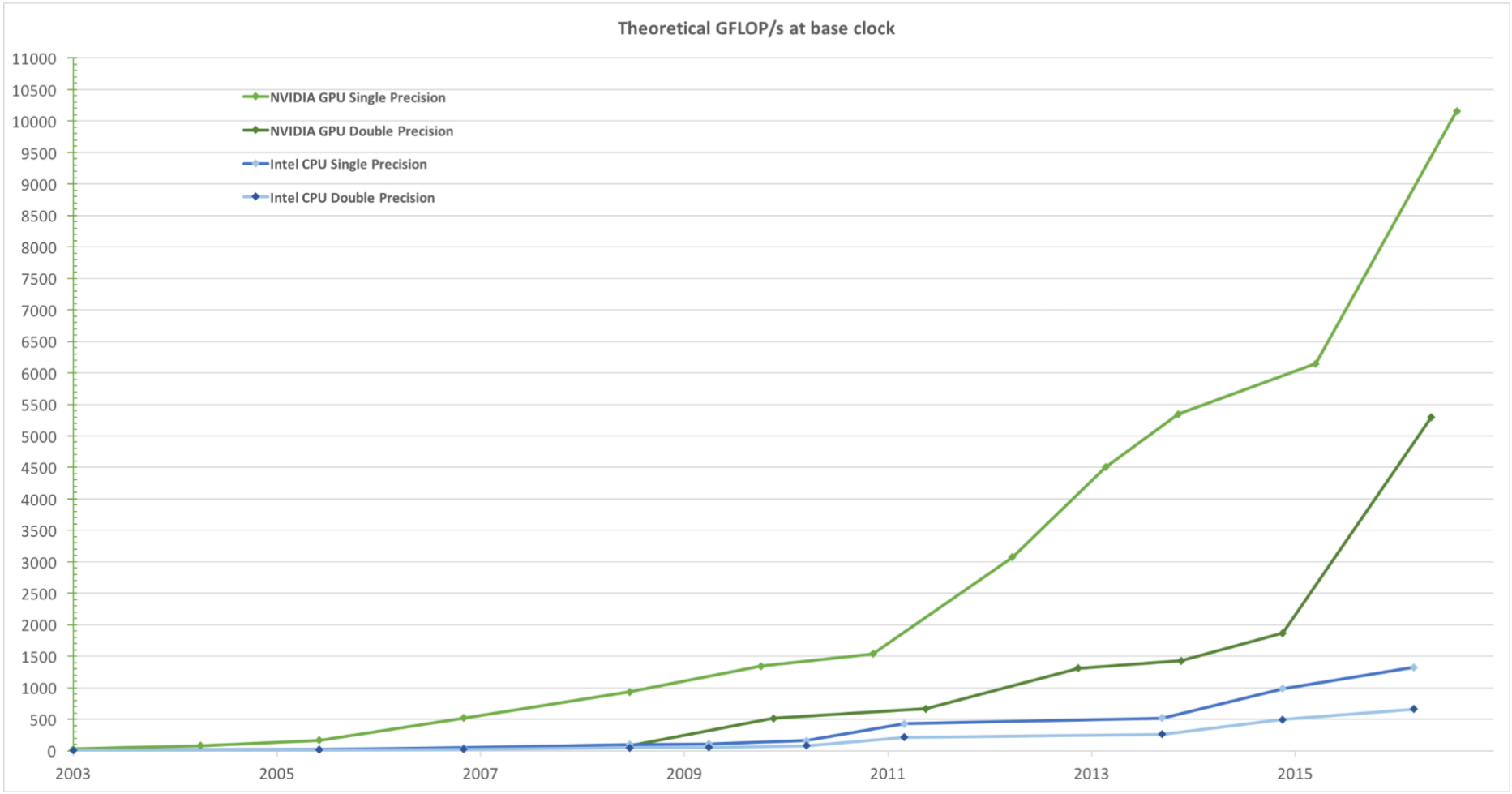


Figure 1 Floating-Point Operations per Second for the CPU and GPU

from the NVIDIA CUDA programming Guide

2016 comparison of flops between CPU and GPU

Figure 1. Floating-Point Operations per Second for the CPU and GPU



comparing memory bandwidths of CPU and GPU

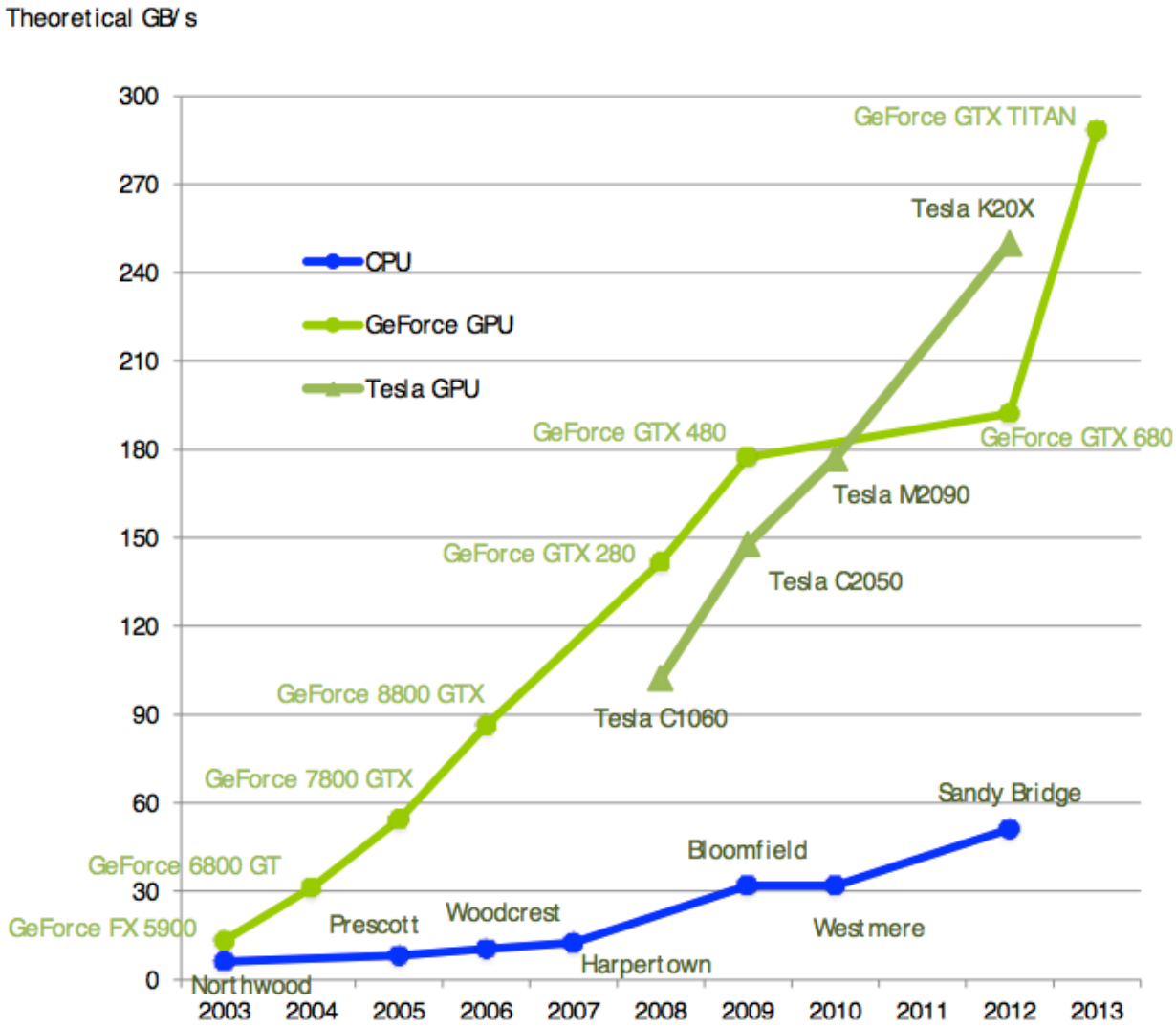
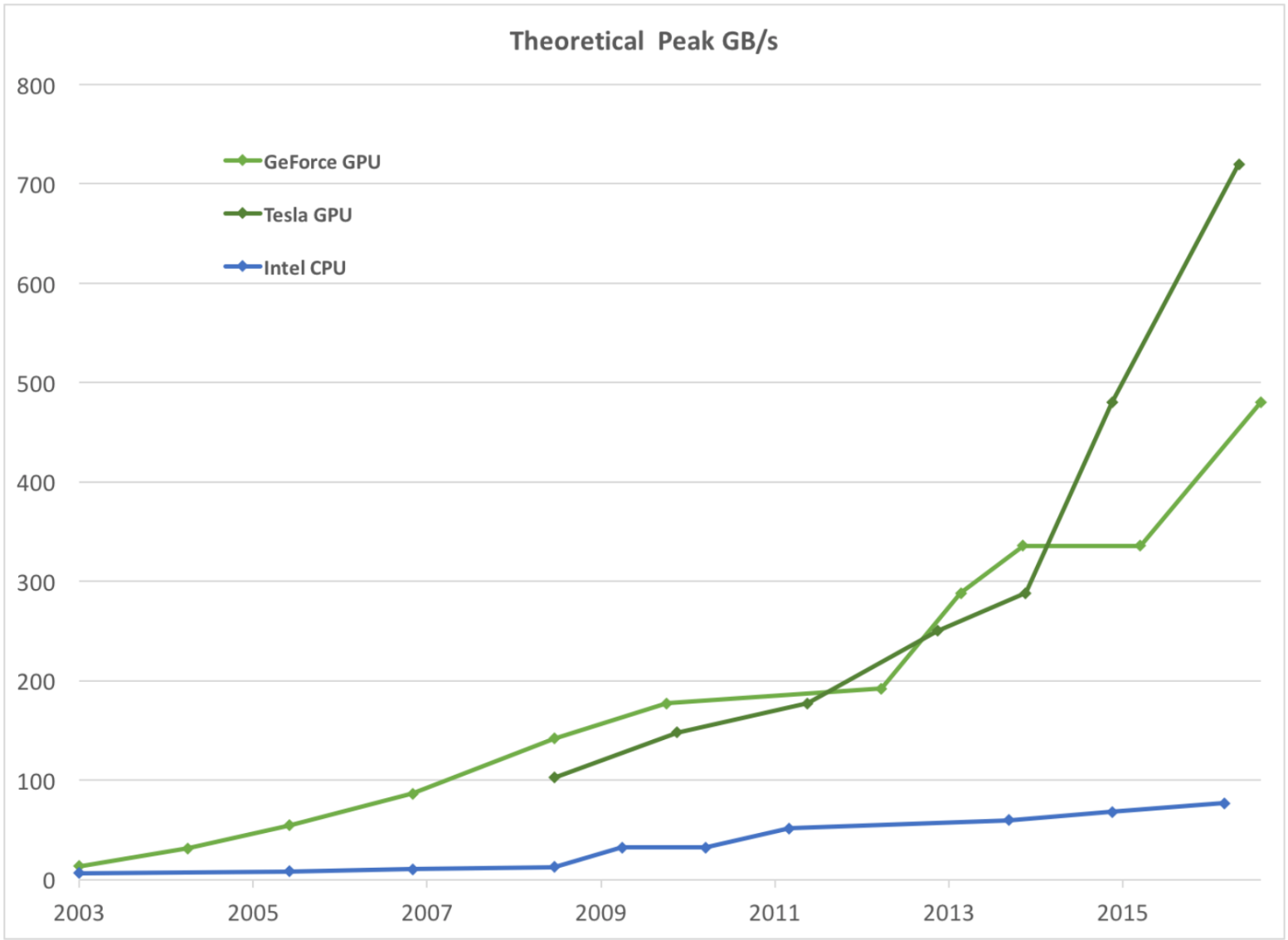


Figure 2 Memory Bandwidth for the CPU and GPU

from the NVIDIA CUDA programming Guide

comparison of bandwidth between CPU and GPU

Figure 2. Memory Bandwidth for the CPU and GPU



memory bandwidth

Graphics chips operate at approximately 10 times the memory bandwidth of CPUs.

Memory bandwidth is the rate at which data can be read from/stored into memory, expressed in bytes per second.

For pascal:

- Intel Xeon E5-2699v4: the memory bandwidth is 76.8GB/s.
- NVIDIA P100: 720GB/s is peak bandwidth.

Straightforward parallel implementations on GPGPUs often achieve directly a speedup of 10, saturating the memory bandwidth.

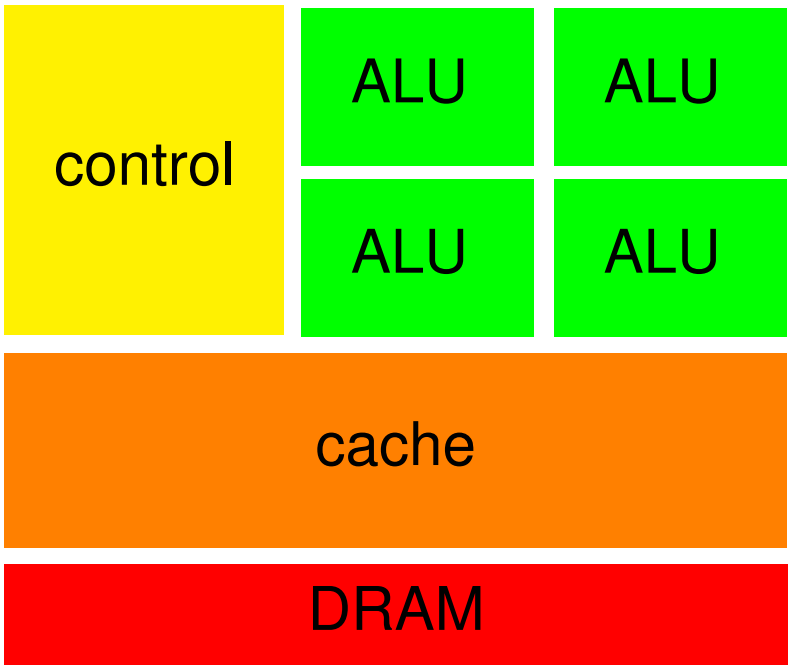
a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software
2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

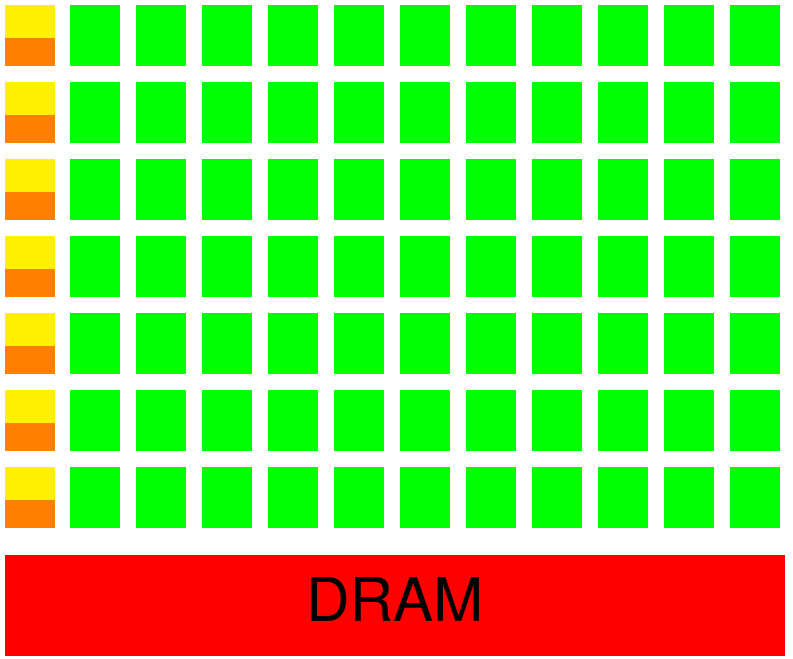
CPU and GPU design

CPU: multicore processors have large cores and large caches using control for optimal serial performance.

GPU: optimizing execution throughput of massive number of threads with small caches and minimized control units.



CPU

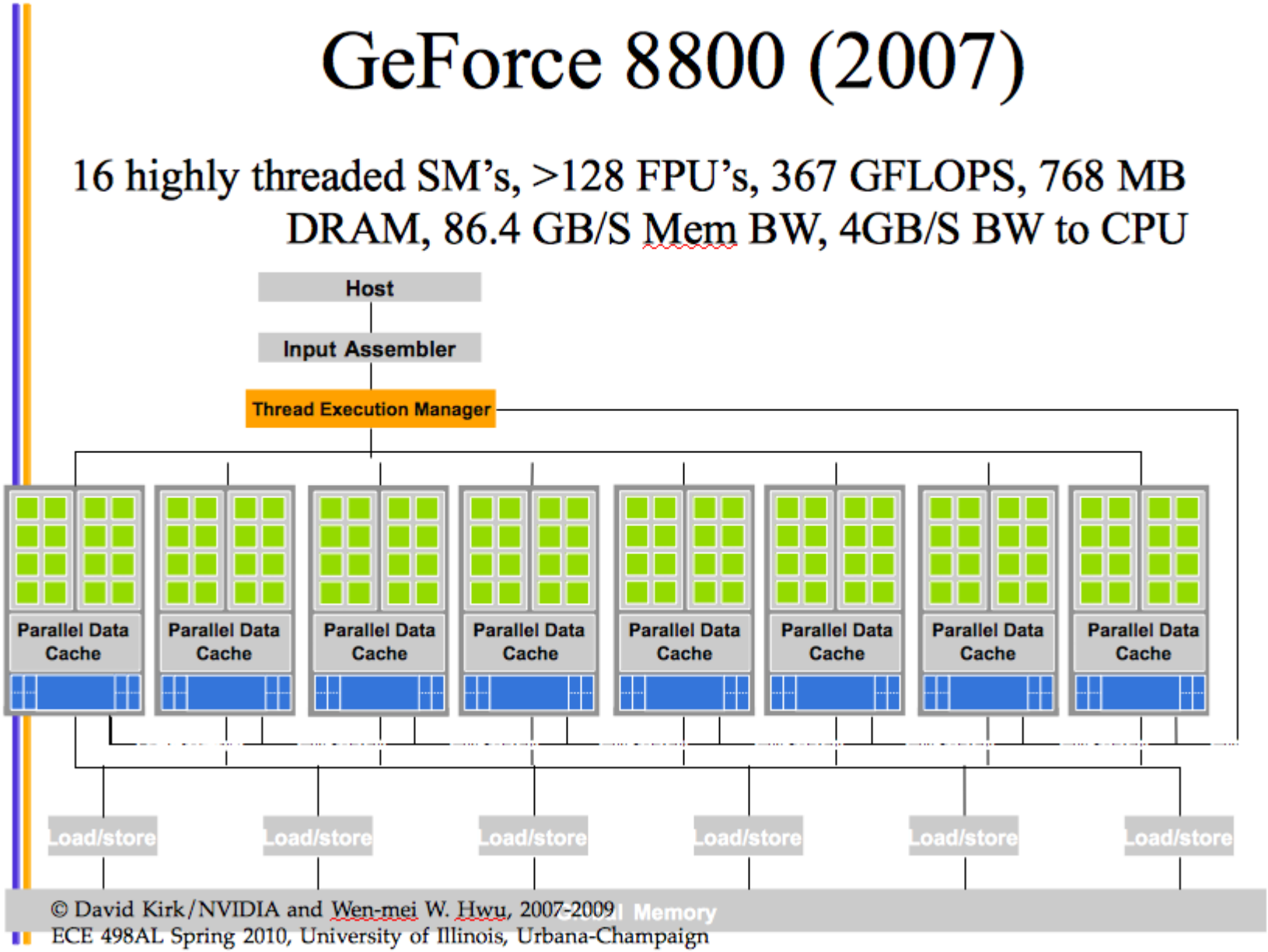


GPU

architecture of a modern GPU

- A CUDA-capable GPU is organized into an array of highly threaded Streaming Multiprocessors (SMs).
- Each SM has a number of Streaming Processors (SPs) that share control logic and an instruction cache.
- Global memory of a GPU consists of multiple gigabytes of Graphic Double Data Rate (GDDR) DRAM.
- Higher bandwidth makes up for longer latency.
- The growing size of global memory allows to keep data longer in global memory, with only occasional transfers to the CPU.
- A good application runs 10,000 threads simultaneously.

GPU architecture



the NVIDIA P100 and the A100 GPU

The P100 GPU has

- 56 streaming multiprocessors (SM),
- each SM has 64 streaming processors (SP),
- $56 \times 64 = 3,586$ cores.

The A100 GPU has

- 108 streaming multiprocessors (SM),
- each SM has 64 streaming processors (SP),
- $108 \times 64 = 6,912$ cores.

Streaming multiprocessors support up to 2,048 threads.

The multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called *warps*.

Unlike CPU cores, threads are executed in order and there is no branch prediction, although instructions are pipelined.

a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - massively parallel computing
   - our equipment: hardware and software
2. Graphics Processors as Parallel Computers
   - the performance gap
   - CPU and GPU design
   - programming models and data parallelism

programming models

According to David Kirk and Wen-mei Hwu (page 14, 1st Edition):

"*Developers who are experienced with MPI and OpenMP will find CUDA easy to learn.*"

CUDA (Compute Unified Device Architecture) is a programming model that focuses on data parallelism.

Programming model: Single Instruction Multiple Data (SIMD).

- Data parallelism: blocks of threads read from memory, execute the same instruction(s), write to memory.
- Massively parallel: need 10,000 threads for full occupancy.

data parallelism

Data parallelism involves

1. huge amounts of data on which
2. the arithmetical operations are applied in parallel.

With MPI we applied the SPMD (Single Program Multiple Data) model.

With GPGPU, the architecture is

SIMT = Single Instruction Multiple Thread

An example with large amount of data parallelism is matrix-matrix multiplication in large dimensions.

Available Software Development Tools (SDK), e.g.: BLAS, FFT from the NVIDIA web site.

Alternatives and Extensions

Alternatives to CUDA:

- OpenCL (chapter 14) for heterogeneous computing;
- OpenACC (chapter 15) uses directives like OpenMP;
- C++ Accelerated Massive Parallelism (chapter 18).

Extensions:

- Thrust: productivity-oriented library for CUDA (chapter 16);
- CUDA FORTRAN (chapter 17);
- MPI/CUDA (chapter 19).

And then, of course, there is Julia, which provides packages for *vendor agnostic GPU computing*.

suggested reading

We covered chapter 1 of the book by Kirk and Hwu.

- NVIDIA CUDA Programming Guide.
  Available at `developer.nvdia.com`
- Victor W. Lee et al: **Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU**.
  In *Proceedings of the 37th annual International Symposium on Computer Architecture (ISCA'10)*, ACM 2010.
- W.W. Hwu (editor). **GPU Computing Gems: Emerald Edition**.
  Morgan Kaufmann, 2011.

Notes and audio of a ECE 498 at UIUC, Spring 2009, are at
`https://nanohub.org/resources/7225/supportingdocs`.

Exercise: How strong is the graphics card in your computer?