CPSC/ECE 4780/6780

General-Purpose Computation on Graphical Processing Units (GPGPU)

Lecture 2: Introduction to GPGPU

Recaps of Last Lecture

1. What is parallel computing?
2. Why use parallel computing?
3. What is the computer architecture of parallel computing?
4. What is the memory architecture of parallel computing?
5. What are the classification of parallel computing models?
6. What kind of parallel computing languages are available?
7. How to design a parallel program?
8. How to evaluate the performance of a parallel program?

Contents of this Lecture

- GPU
- CPU vs. GPU
- GPGPU

Slides Materials

- Programming Massively Parallel Processors A Hands-on Approach, David B. Kirk, Wen-mai W. Hwu
- Introduction to GPU Hardware and to CUDA, Philip Blakely
- GPU Computing with CUDA, Christopher Cooper
- www.NVIDIA.com

What is GPU?

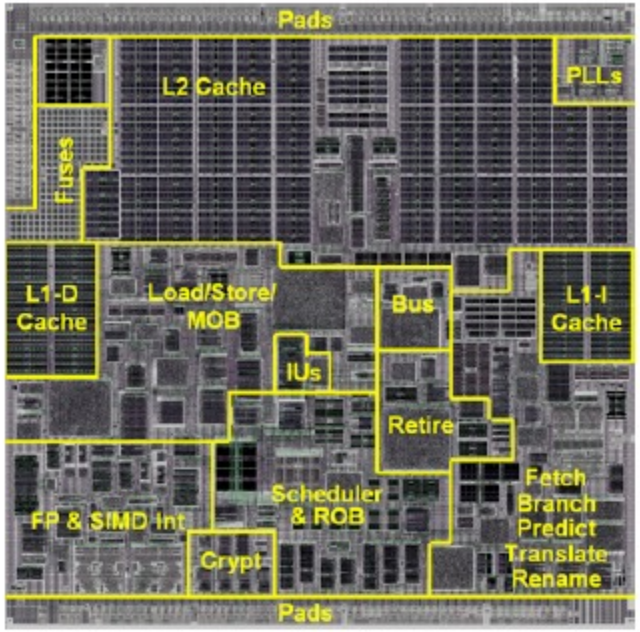
- The "Graphics Processing Unit"
  - A processor that initially specialized for processing graphics
  - Recently evolved towards a more flexible architecture capable of general computations

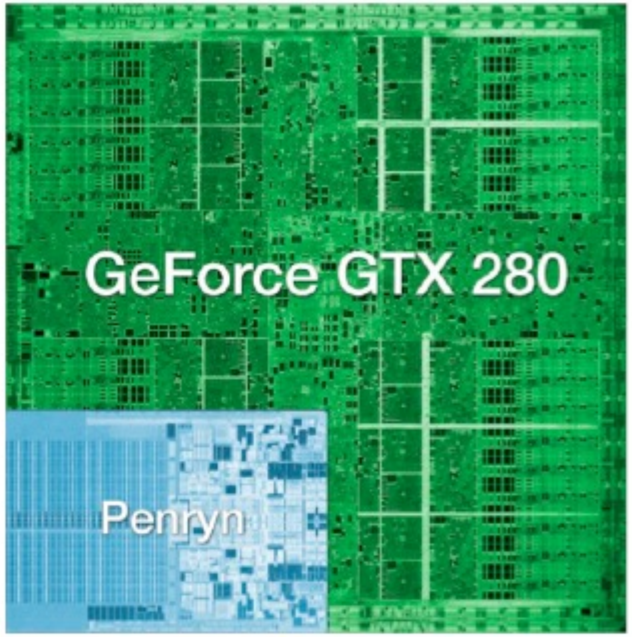
History of GPUs

- 1990s: Graphics cards for PCs
- 1999: NVIDIA's first GPU
- 2000: Advent of GPGPUs
- 2003: Brook – the first programing model to extend C with data-parallel constructs by Ian Buck
- 2007: CUDA (Compute Unified Device Architecture)
- 2010: NVIDIA release Fermi cards – more versatile
- 2012: NVIDIA release Kepler cards – even more scope for dynamic programming
- 2014: NVIDIA release Maxwell cards – more efficient, allowing more cores per GPU
- 2016: NVIDIA release Pascal cards – improved bandwidth and unified virtual addressing scheme across GPUs and between CPUs and GPUs

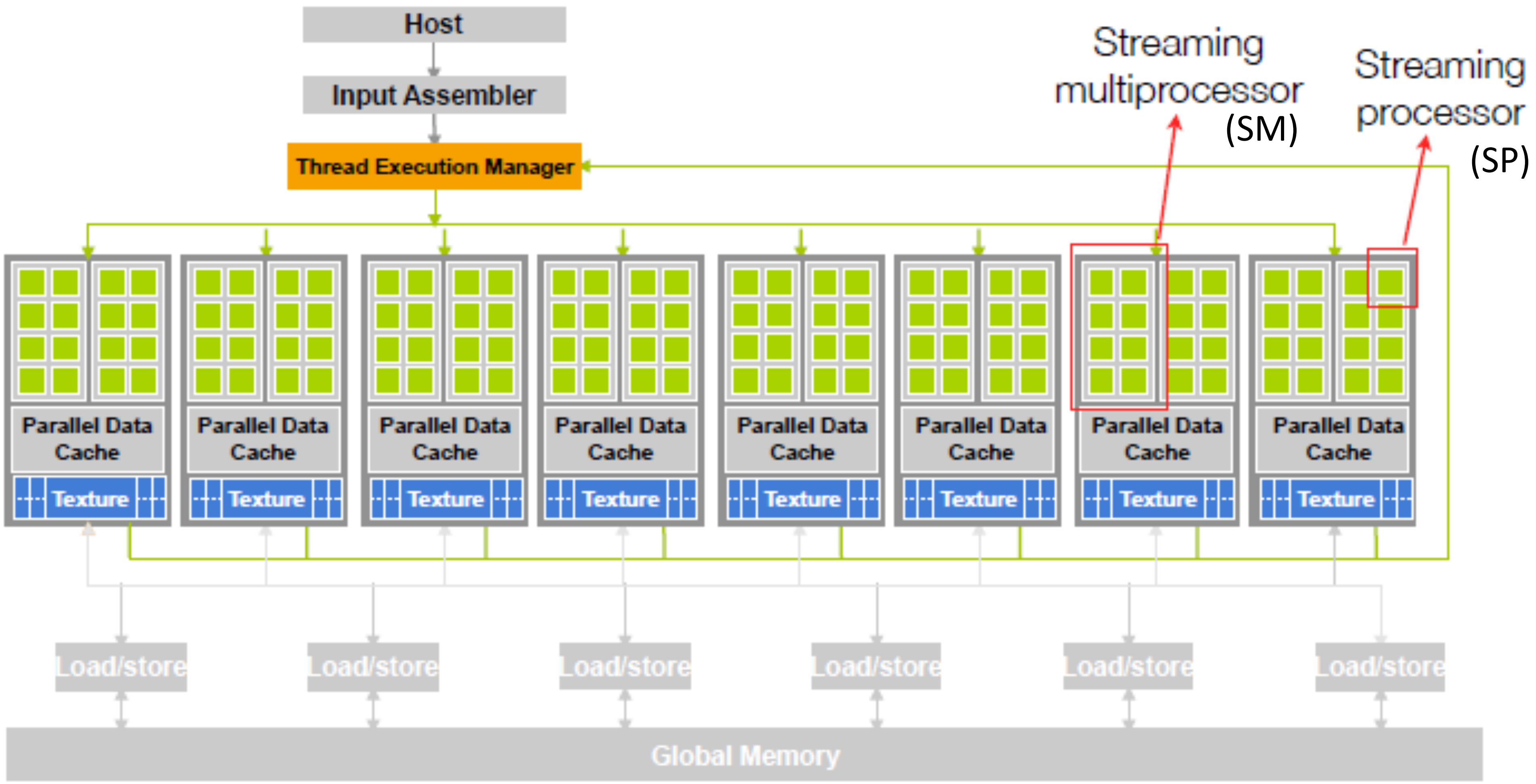
How Does a GPU Look Like?

- NVIDIA GeForce GTX 280 GPU
    - 240 parallel cores
        - Floating point unit
        - Logic unit
        - Move, compare unit
        - Branch unit
    - Heavily multithreaded
    - In-order
    - Single instruction



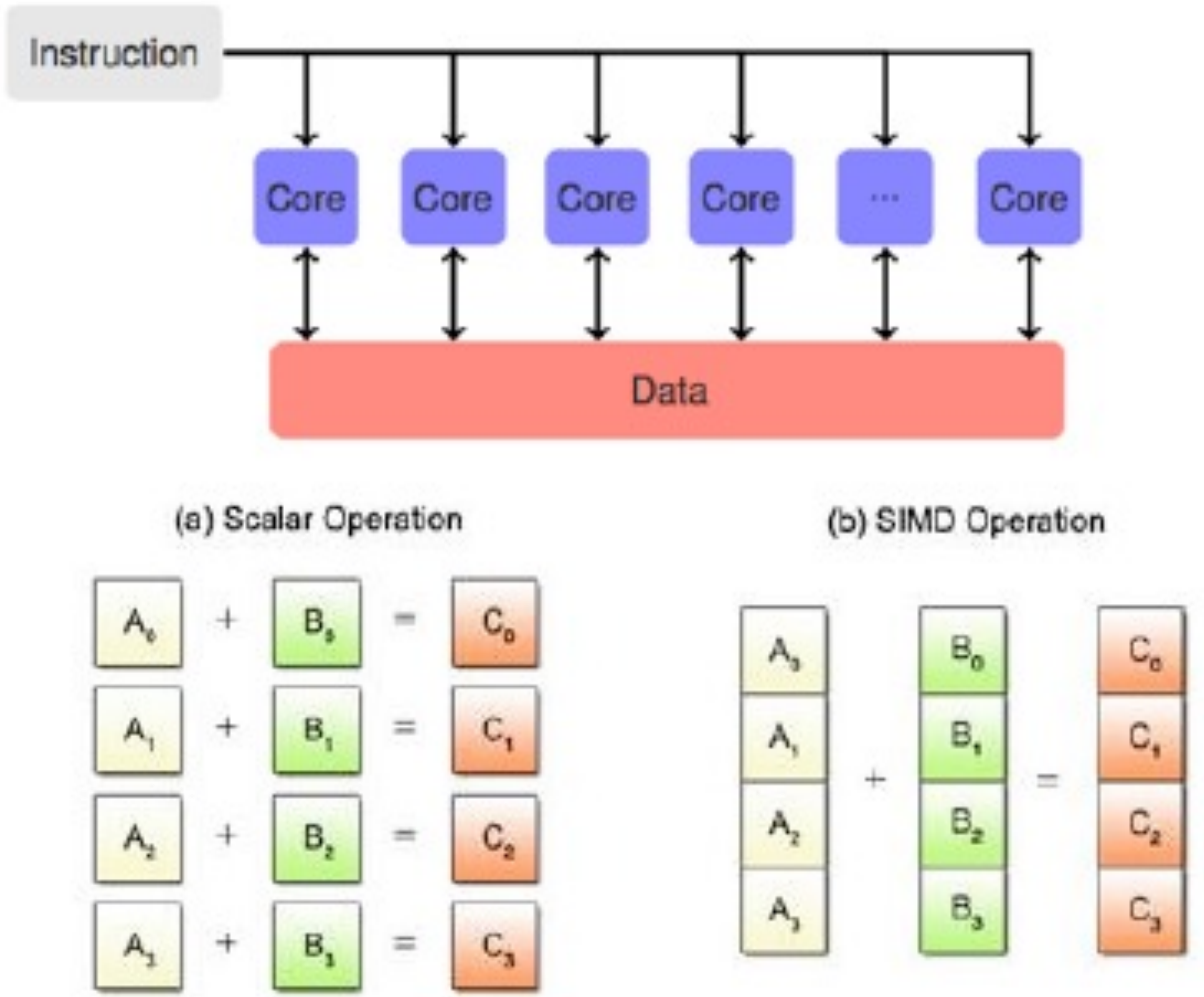


Architecture of a Modern GPU



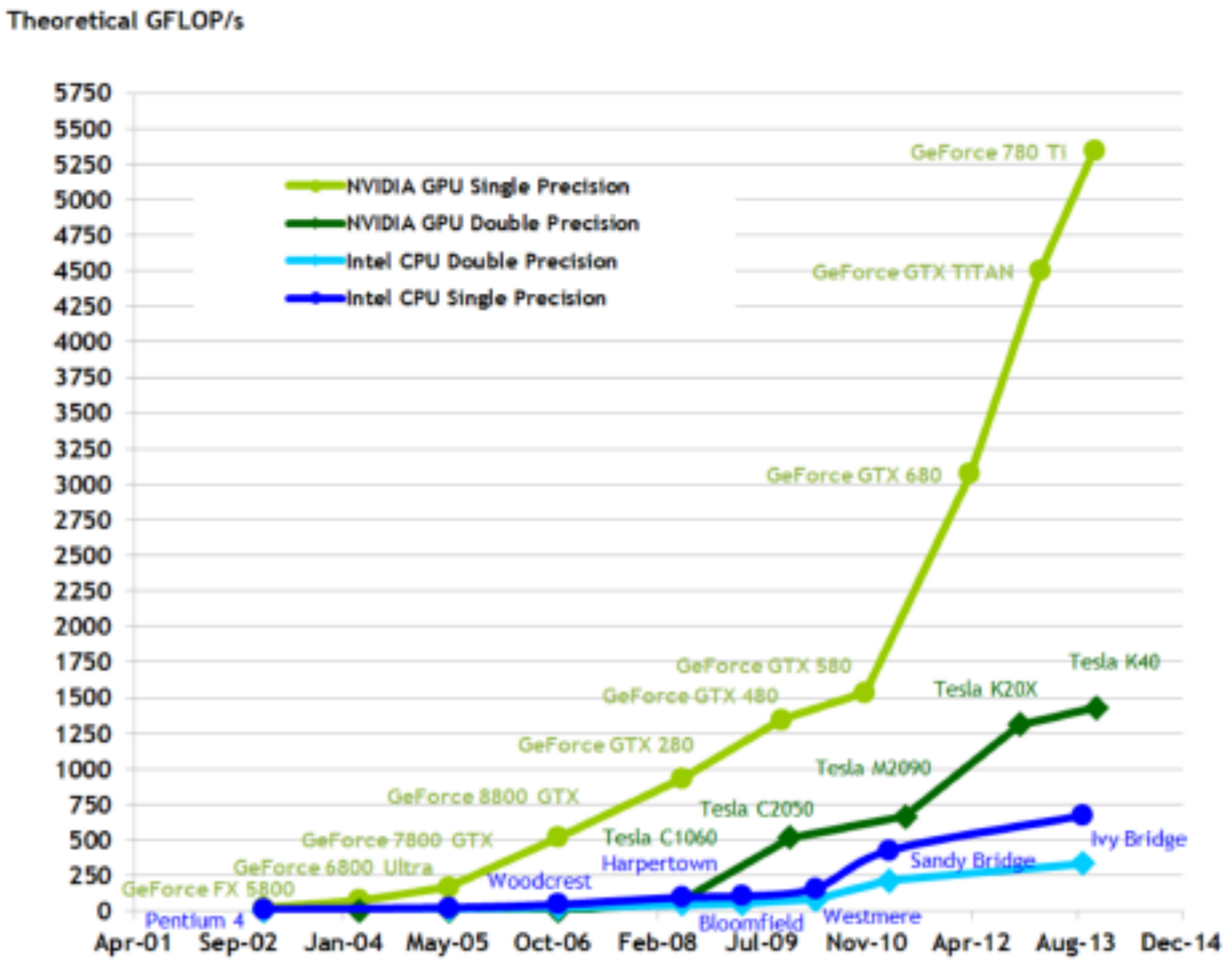
Architecture of a CUDA-capable GPU: GeForce 8800

Stream Multiprocessor (SM)



Mythbusters Demo GPU versus CPU

Why Should We Use GPUs?

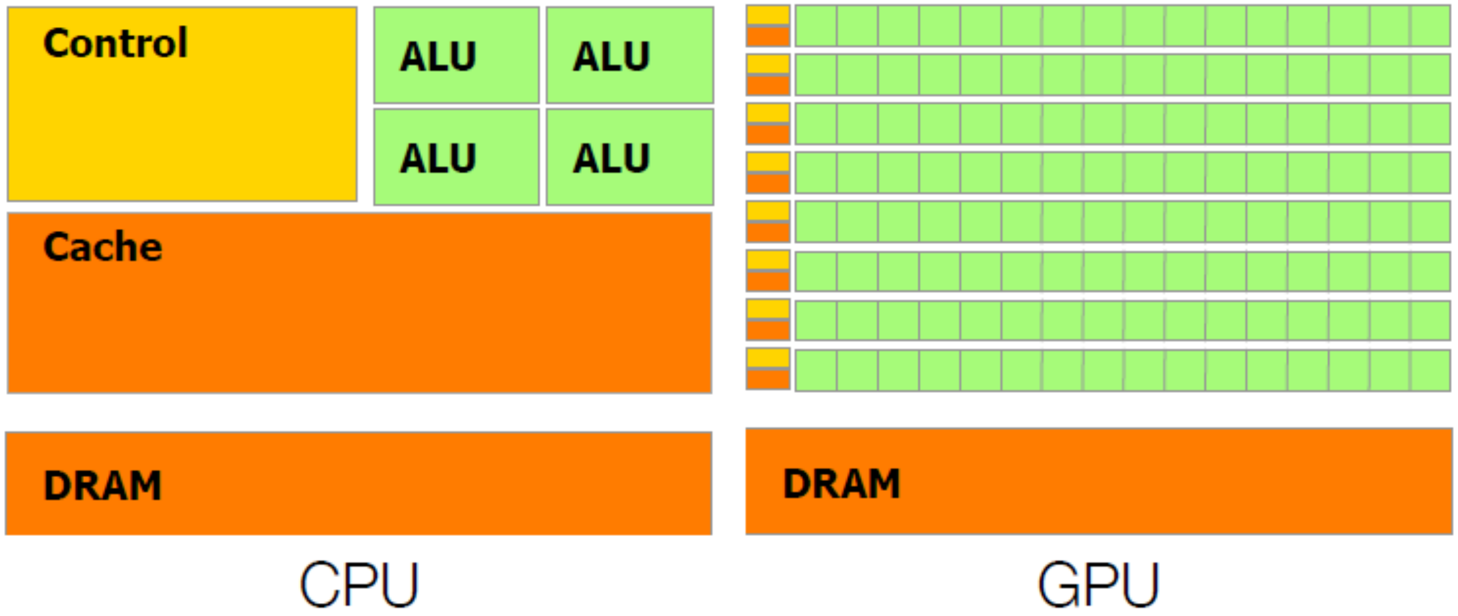


Floating-point operations per second for the CPU and GPU

Improved FLOPS!

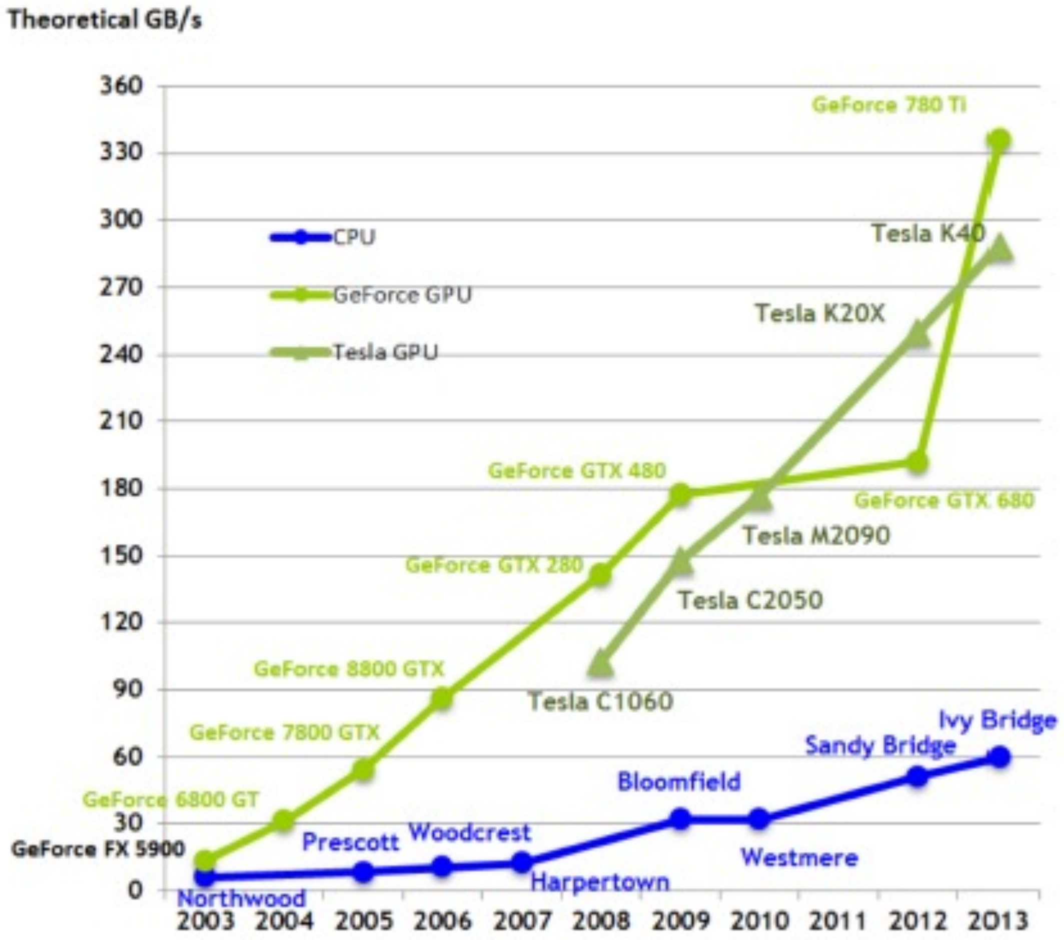
CPU/GPU Design Comparison

- CPU: large number of transistors associated with flow control and data caching
- GPU: more transistors are devoted to floating point operations for data processing



GPU devotes more transistors to data processing

CPU/GPU Bandwidth Comparison



Memory Bandwidth for the CPU and GPU

CPU/GPU Perf/Watt Comparison

2.3 PFlops

7.0 Megawatts

7000 homes

7.0 Megawatts

Traditional CPUs are not economically feasible

CPU/GPU More Comparisons

- Threading resources
  - CPU: 4 quad-core processors supports 16 threads
  - GPU: supports at least 768 threads per multiprocessor
- Threads
  - CPU: heavy contexts switch involving data transfer (slow)
  - GPU: registers are allocated for each thread (fast)
- RAM
  - CPU: memory is accessible by any threads at any time
  - GPU: different types of memory for different purposes with different access strategies

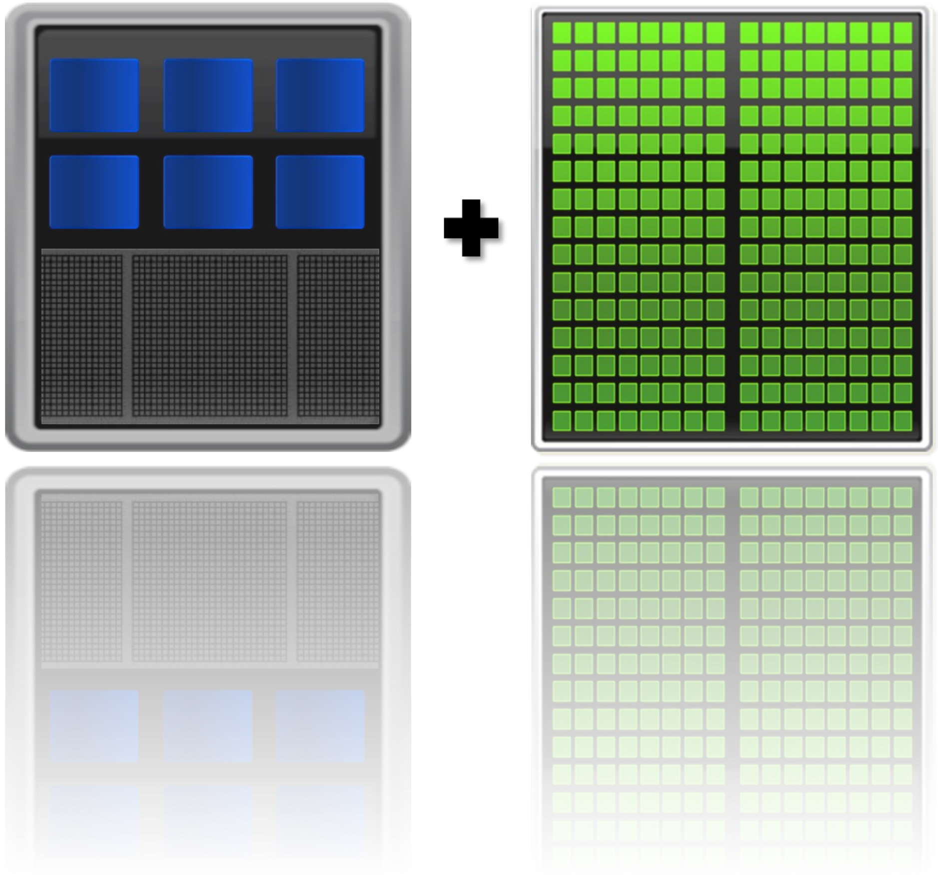
Why Should We Use GPUs? More Reasons

- Improved FLOPs
- Improved bandwidth
- Energy efficient
- Easy accessibility with large market presence
- Improved computing capability
  - Support IEEE floating-point standard
- Programmable using high-level languages
- Large speedup
  - Typically x10
- Relatively cheap

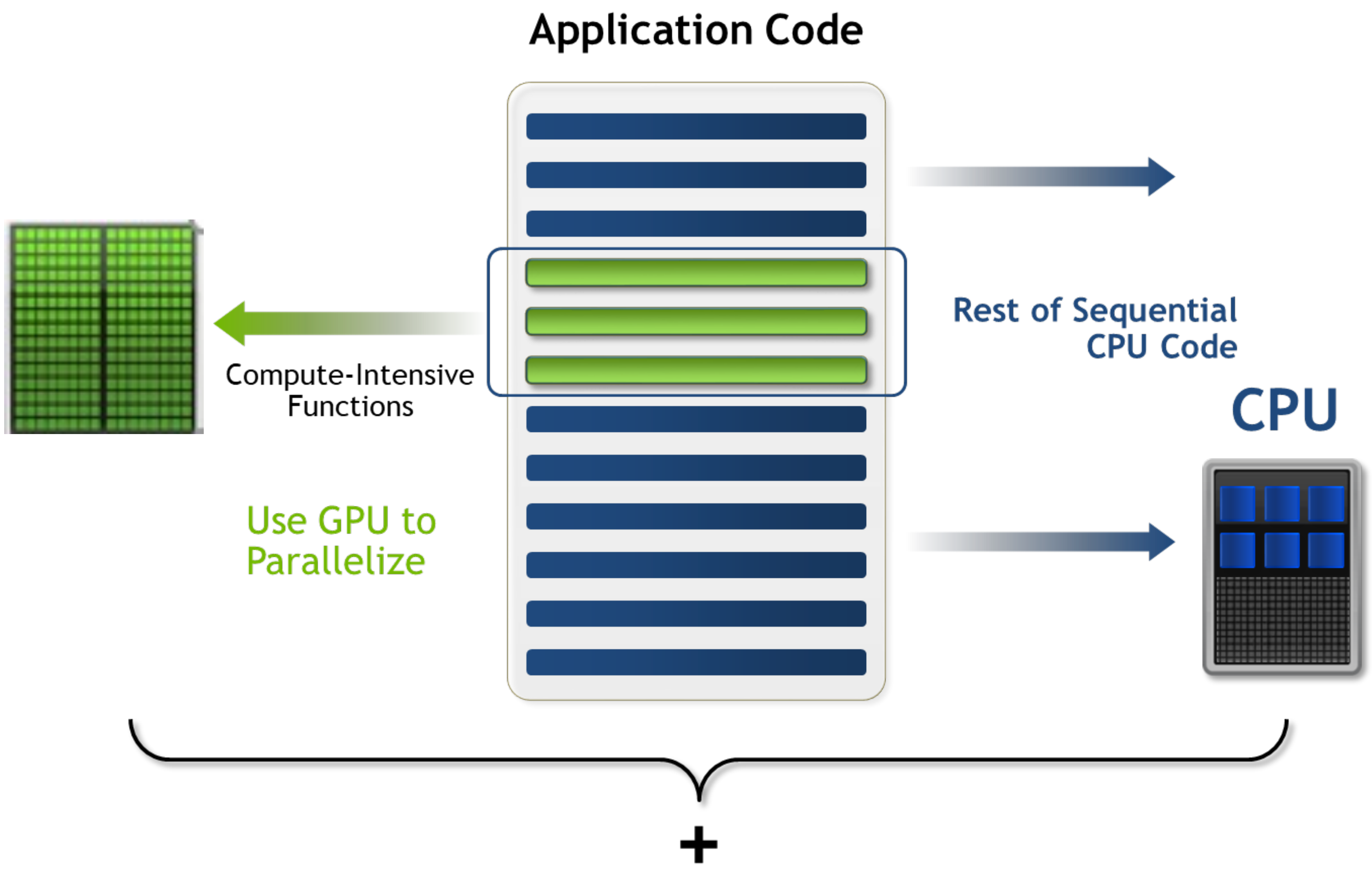
GPU Computing Features

- Fast GPU cycle: New hardware every ~ 18 months
- Requires special programming but similar to C
- CUDA code is forward compatible with future hardware
- Cheap and available hardware
- Important factors to consider: power and cooling!

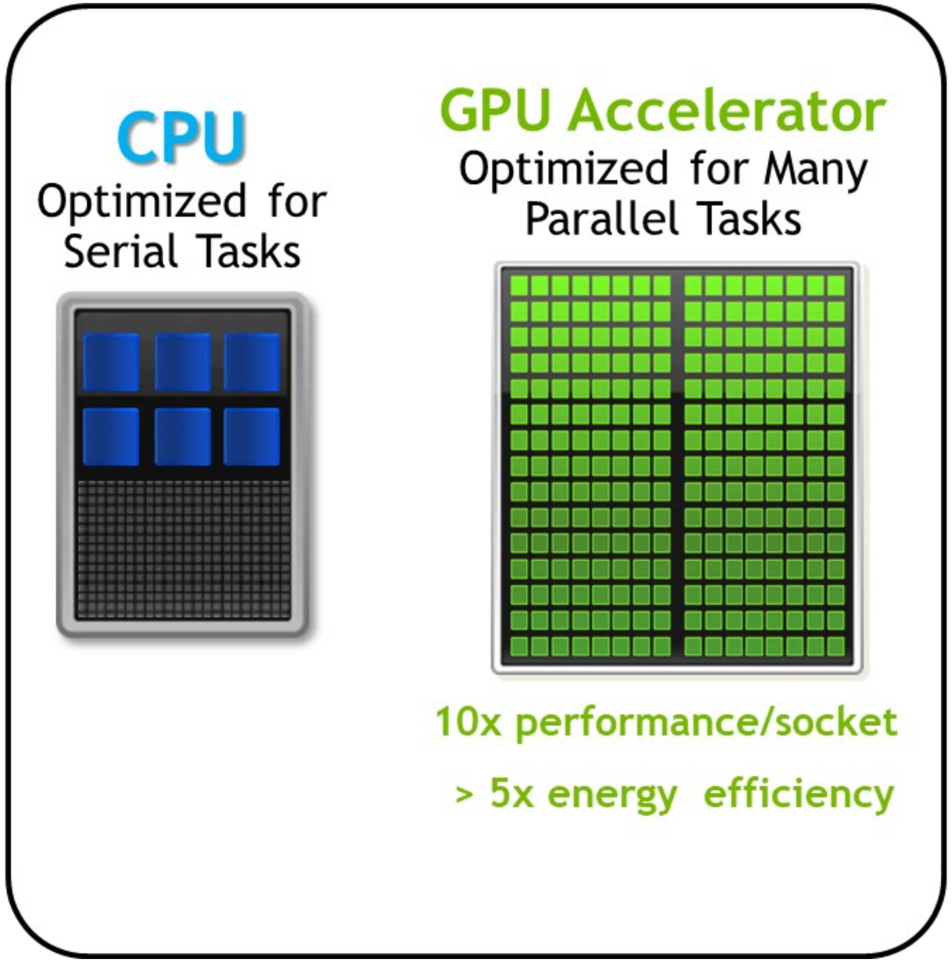
CPU+GPU Acceleration



Big Speed-up


Application Code
Compute-Intensive Functions
Use GPU to Parallelize
Rest of Sequential CPU Code
CPU
+

High Energy Efficiency

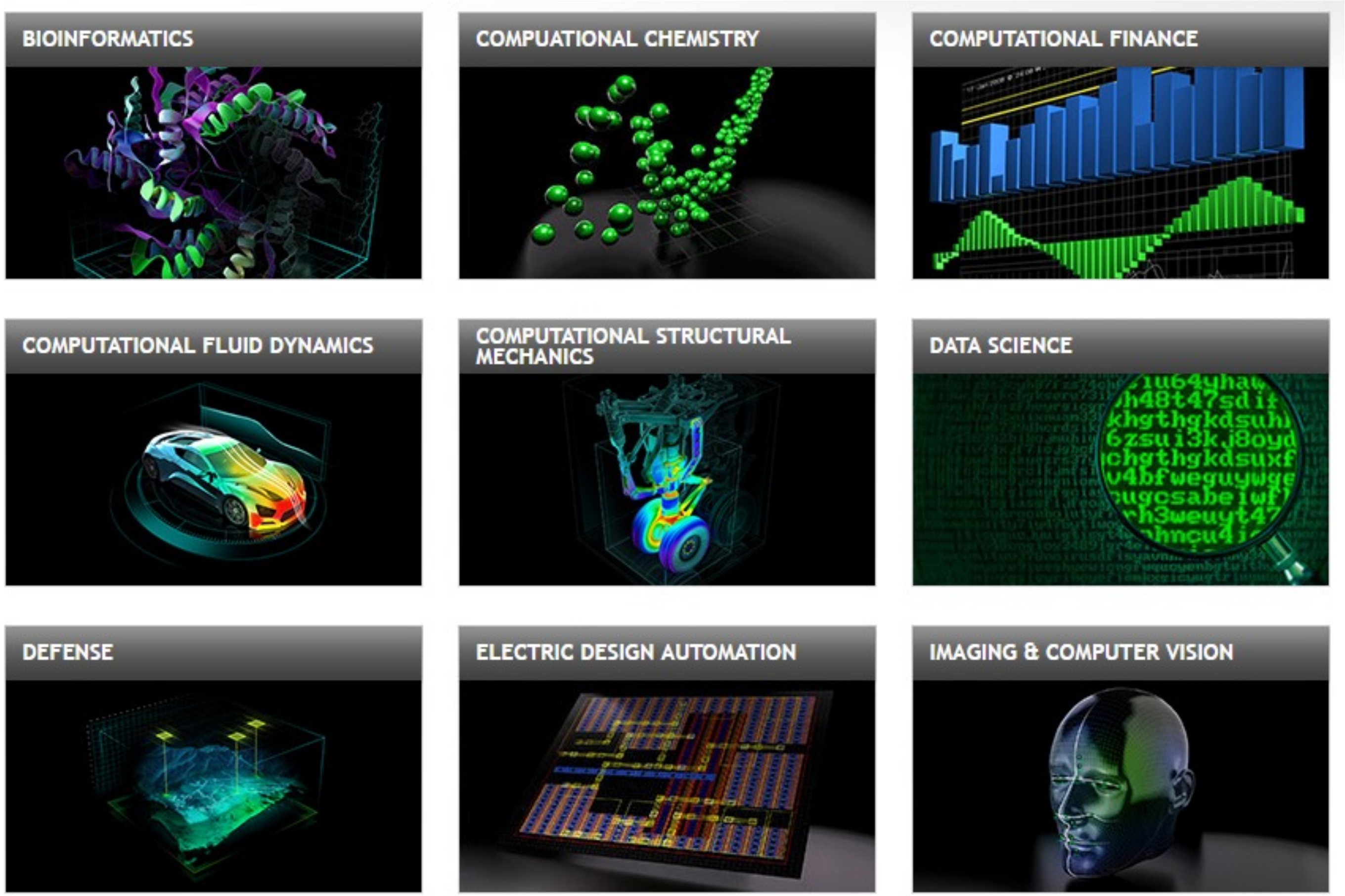


Era of GPU accelerated computing is here

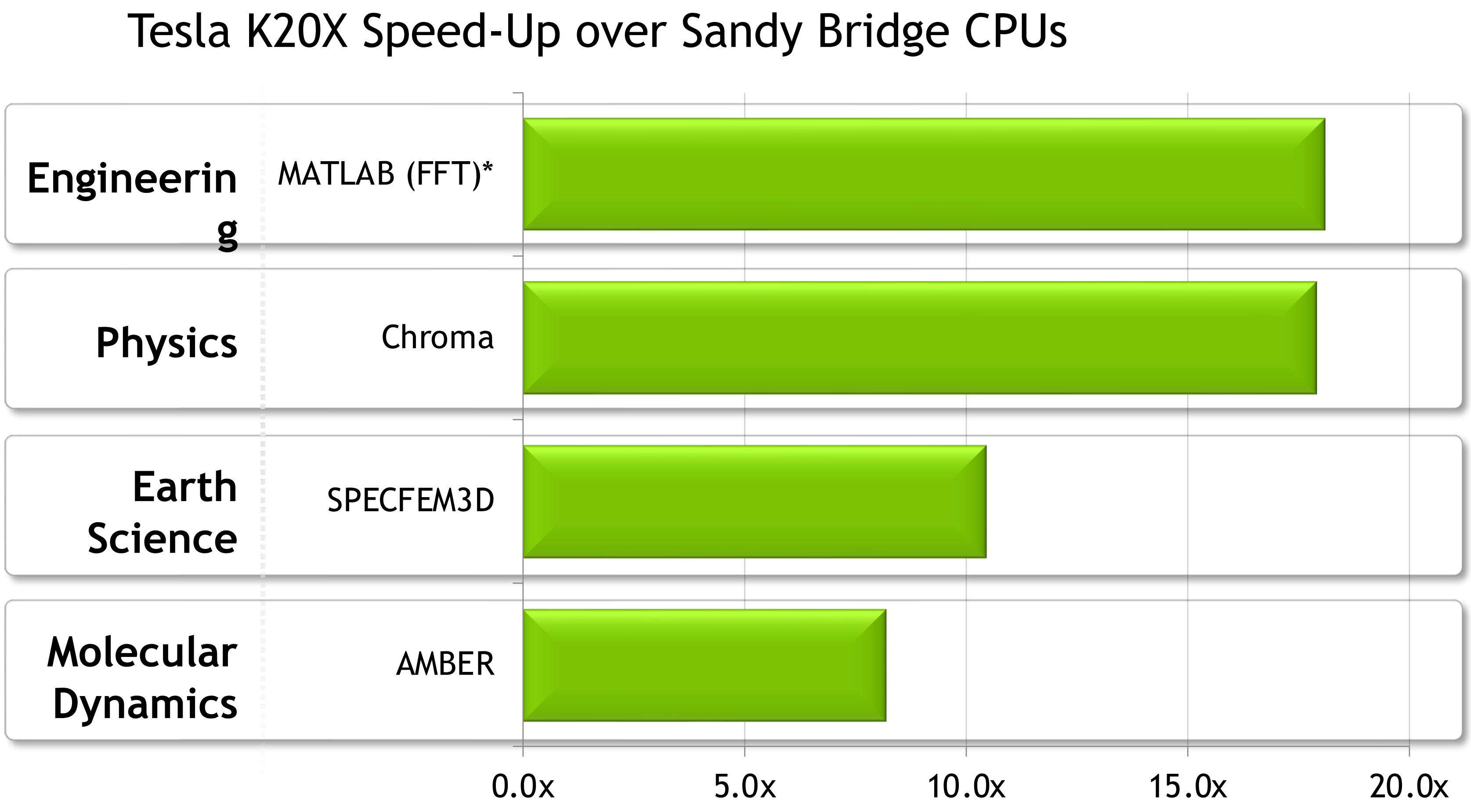
GPGPU

- General Purpose Computation on Graphics Processing Units
  - Perform demanding calculations on the GPU instead of the CPU
  - High processing power in parallel
- Key concepts for fast implementation:
  - Maximizing parallel execution
  - Minimizing data transfers between host and device
  - Minimizing memory latency on the device

GPGPU CUDA Applications



Fastest Performance on Scientific Applications



CPU results: Dual socket E5-2687w, 3.10 GHz, GPU results: Dual socket E5-2687w + 2 Tesla K20X GPUs
*MATLAB results comparing one i7-2600K CPU vs with Tesla K20 GPU
Disclaimer: Non-NVIDIA implementations may not have been fully optimized

GPU Programing

- GPU programming is typically SIMD
- Programming for GPUs requires rethinking algorithms
- Best algorithm for CPU not necessarily best for GPU
- Knowledge of GPU hardware required for best performance
- GPU programming works best if:
  - Perform same operation simultaneously on multiple pieces of data
  - Organize operations to be as independent as possible
  - Arrange data in GPU memory to maximize rate of data access