Lecture 1
Basic concept

Outline

- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

Outline

- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

What is GPU?

- **A special card in the computer that processes the graphical tasks**
- **Example of GPU tasks**
  - Video editing, gaming, animation work
- **Your computer can have both an integrated graphics and a GPU**

What is a Graphics Card?

- GPU – Graphics Processing Unit
- A circuit board that draws pretty pictures
- A specialized piece of hardware, originally designed as a graphics processor
- Designed for maximum performance in image drawing

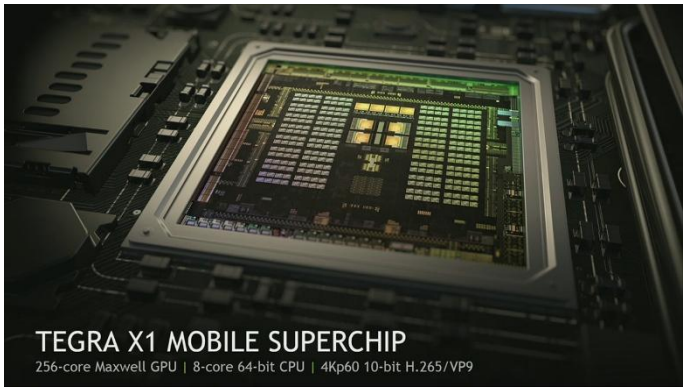
NVIDIA Geforce-9800-GT

What is a Graphics Card?

- **Modern games require enormous performance**
- **GPU computing is becoming more versatile**
- **Nvidia's GeForce 256 (1999) – first GPU**
- **Improves on its predecessor (RIVA TNT2)**
  - Increasing the number of fixed pixel pipelines

Modern GPUs Are Present

- **Embedded systems**
  - NVIDIA Tegra X1
- **Personal computers**
- **Game consoles**
- **Mobile phones**
  - Apple A8x, w/ PowerVR GPU
- **Workstations**
- **Supercomputers**
  - Titan



The first video game console

Main Manufacturers

- **GPU's for high performance computing**
  - NVIDIA (Kepler, Maxwell, Pascal)
  - Intel (HD Graphics)
  - AMD (Radeon)

- **GPU's for embedded/mobile computing**
  - ARM (Mali)
  - NVIDIA (Tegra K1, X1)
  - QUALCOMM (Adreno)
  - Imagination (PowerVR)
  - Intel (HD Graphics, Iris Graphics)
  - Other in house designs

Outline

- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

Why GPU Computing?

- **Technology evolution**
  - –Memory wall
  - –Power wall
  - –ILP wall
- **Usage changes**
  - –New applications and constraints
- **Latency vs. throughput**

Technology Evolution

- **Technology trend has shifted computing paradigms**
  - Memory wall
  - Memory speed does not increase as fast as computing speed
  - More difficult to hide memory latency
- **Power wall**
  - Power consumption of transistors does not decrease as fast as density increases
  - Performance is now limited by power consumption
- **ILP wall**
  - Diminishing returns on Instruction-Level Parallelism

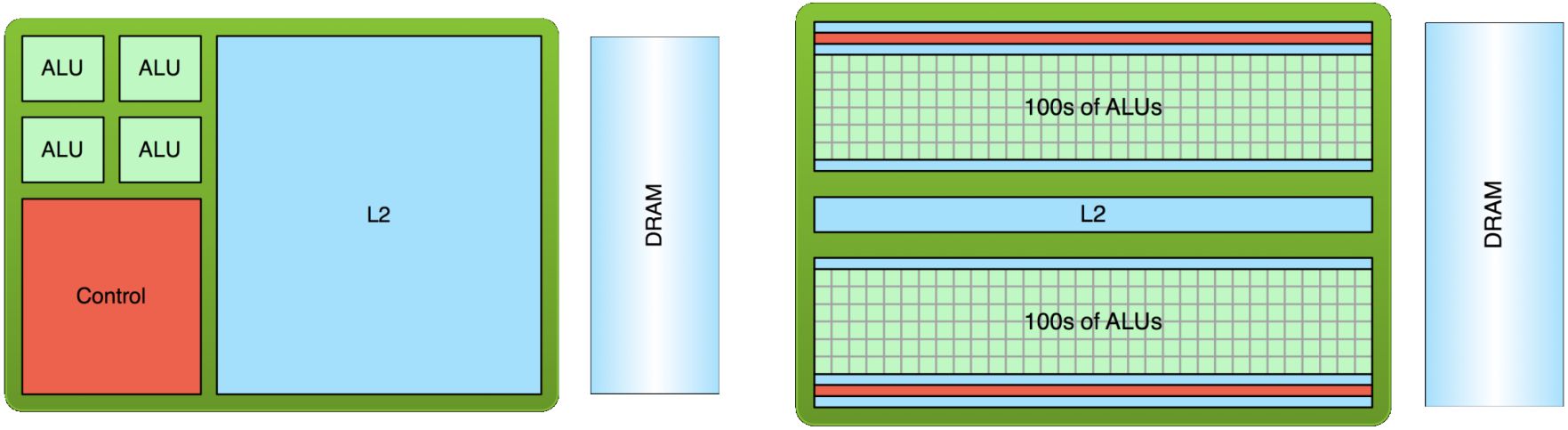
Usage Changes

- **New applications demand parallel processing**
  - Computer games: 3D graphics
  - Search engines, social networks… "big data" processing
- **New computing devices are power-constrained**
  - Laptops, cell phones, tablets…
    - Small, light, battery-powered
  - Datacenters
    - High power supply and cooling costs

Latency and Throughput

- **Latency is a *time delay* between the moment something is initiated, and the moment one of its effects begins or becomes detectable**
  - For example, the time delay between a request for texture reading and texture data returns
- **Throughput is the amount of work done in a given amount of time**
  - For example, how many triangles processed per second

Low Latency or High Throughput?



- **CPU**
  - Optimized for low-latency access to cached data sets
  - Control logic for out-of-order and speculative execution
- **GPU**
  - Optimized for data-parallel, throughput computation
  - Architecture tolerant of memory latency
  - More transistors dedicated to computation

Low Latency or High Throughput?

- **CPU**
  - Latency:
    time to solution

- **GPU**
  - Throughput:
    quantity of tasks processed
    per unit of time

Outline

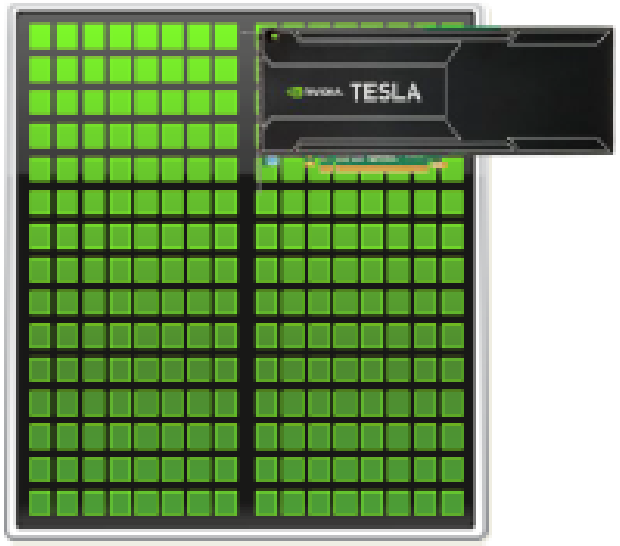
- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

CPU vs. GPU

CPU

CPUs consist of a few cores optimized for serial processing

GPU



GPUs consist of hundreds or thousands of smaller, efficient cores designed for parallel performance

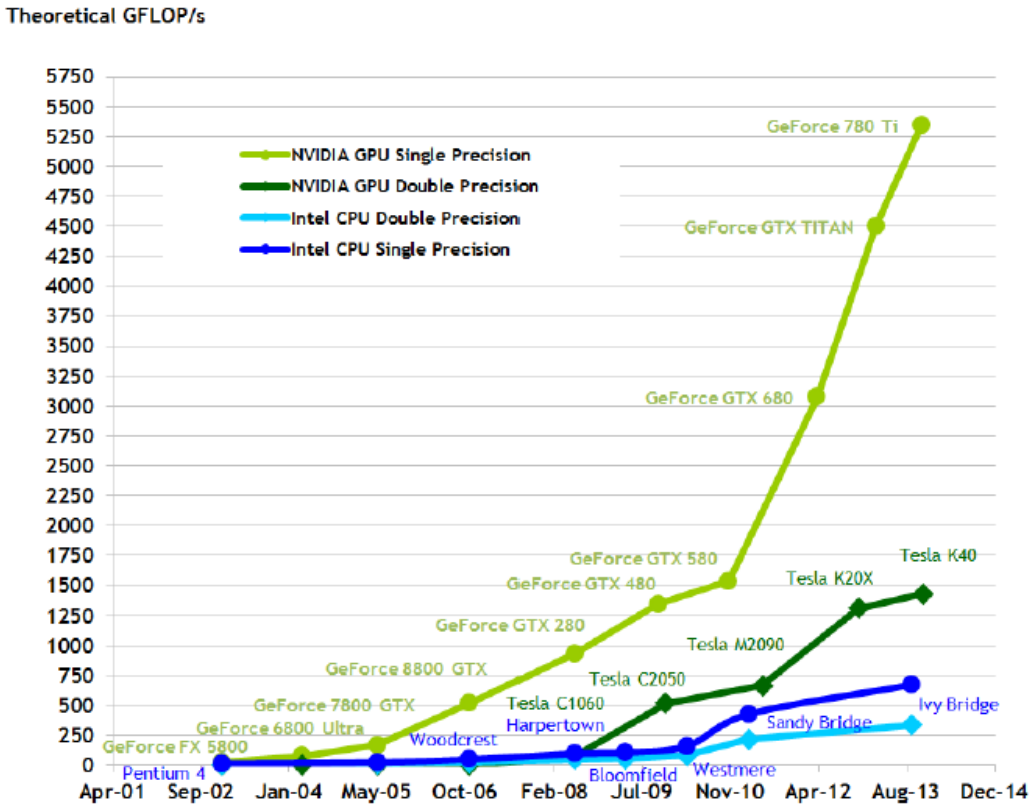
CPU vs. GPU

Comparison With

- Performance in global floating-point operations per second (GFLOP/s)
- Performance in memory bandwidth
- Transistor level hardware

CPU vs. GPU

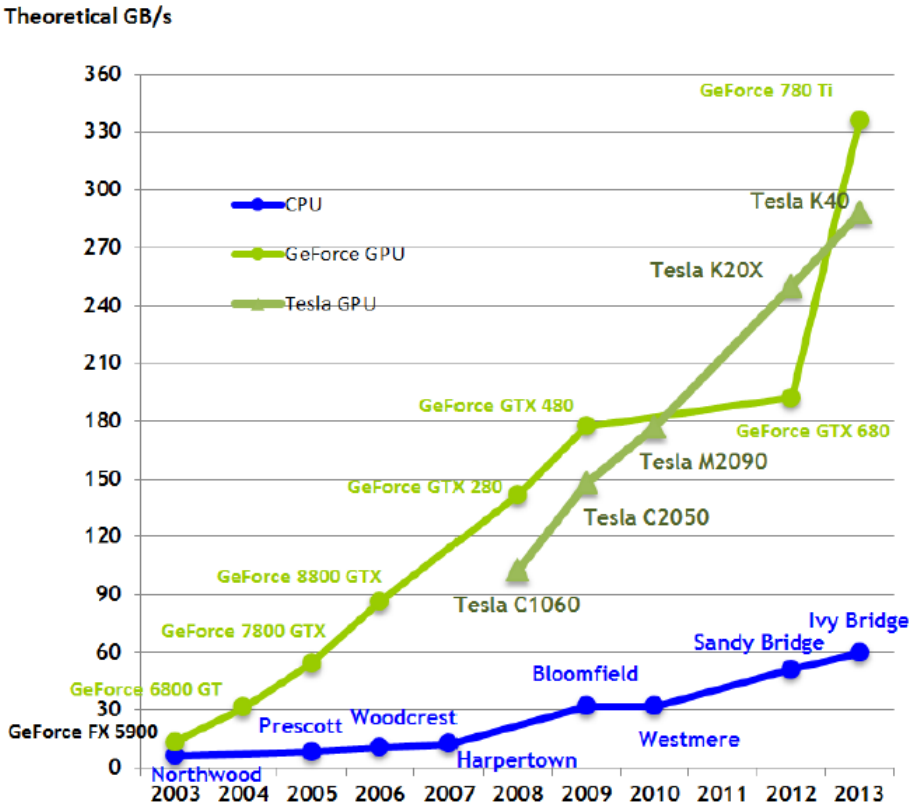
Floating-Point Operations per Second



http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3g8riFfwE

CPU vs. GPU

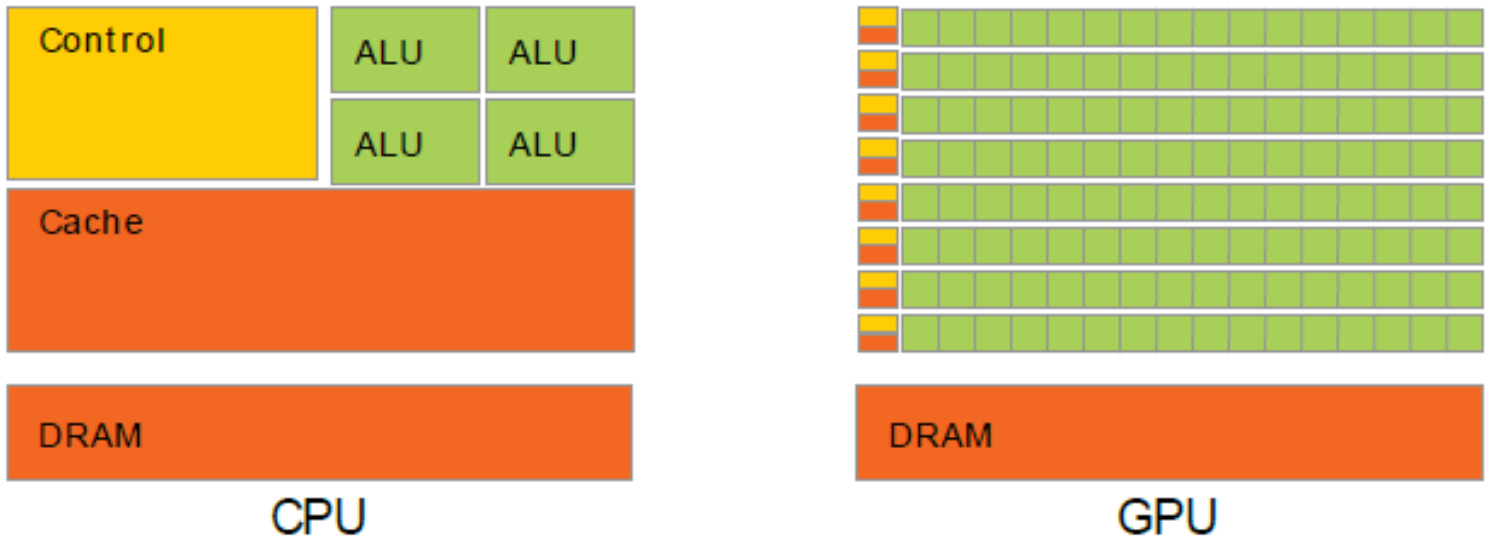
Memory Bandwidth



http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3g8riFfwE

CPU vs. GPU

Transistors Level

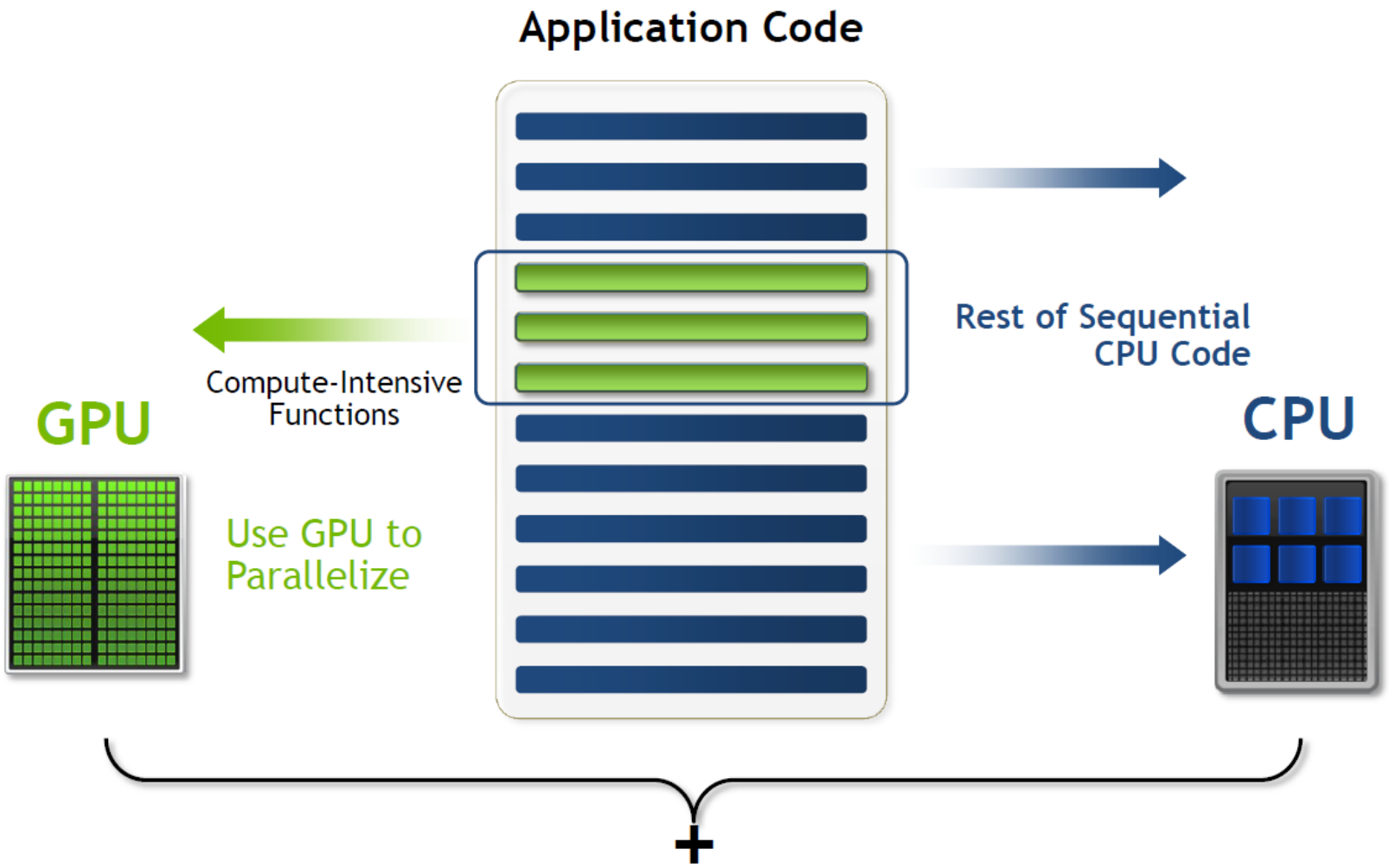


- CPUs are great for *task* parallelism
- GPUs are great for *data* parallelism

What Accounts For This Difference?

- **Need to understand how CPUs and GPUs differ**
  - Latency Intolerance vs. Latency Tolerance
  - Task Parallelism vs. Data Parallelism
  - Multi-threaded Cores vs. SIMT (Single Instruction Multiple Thread) Cores
  - 10s of Threads vs. 10,000s of Threads

Small Changes, Big Speed-up



Ideal Apps to Target GPGPU

- Large data sets
- High parallelism
- Minimal dependencies between data elements
- High arithmetic intensity
- Lots of work to do without CPU intervention

Outline

- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

GPGPU Programming Framework

- Brook
- CUDA
- OpenCL
- C++ AMP
- OpenACC
- ArrayFire

Brook

- **One of the earlies GPU frameworks by Standford University**
- **The Brook programming language and its implementation BrookGPU were an early and influential attempt to enable general-purpose computing on graphics processing units**
- **Brook has been in beta for a long time**
  - Renewed development stopped again in November 2007

CUDA

- A parallel computing platform and application programming interface (API) model created by NVIDIA
- It allows software developers to use a CUDA-enabled graphics processing unit (GPU)
- The CUDA platform is designed to work with programming languages such as C, C++ and Fortran
- Also, CUDA supports programming frameworks such as OpenACC and OpenCL

OpenCL

- Open source general framework by Khronos Group
- A framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, DSPs, FPGAs and other processors
- OpenCL specifies a language (based on C99) for programming these devices
- OpenCL provides parallel computing using task-based and data-based parallelism

C++ AMP

- Open C++ extension by Microsoft
- Native programming model that contains elements that span the C++ programming language and its runtime library
- Provides an easy way to write programs that compile and execute on data-parallel hardware, such as graphics cards (GPUs)

OpenACC

- C, C++ and Fortran extension
- A programming standard for parallel computing
- The standard is designed to simplify parallel programming of heterogeneous CPU/GPU systems
- New suggested capabilities include new controls over data movement, and support for explicit function calls and separate compilation

ArrayFire

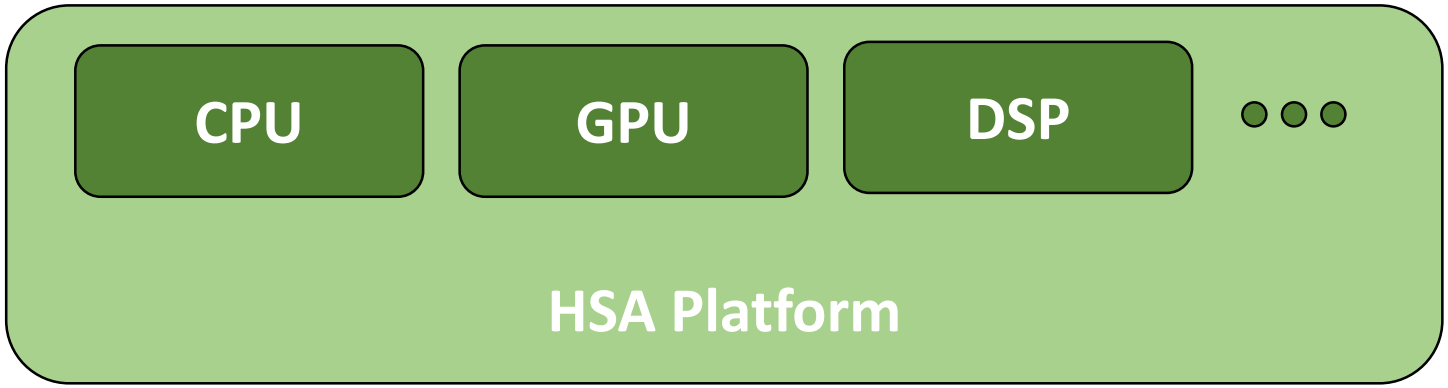
- **ArrayFire is a GPU matrix library for the rapid development of general purpose GPU (GPGPU) computing applications within C, C++, Fortran, and Python**
  - ArrayFire contains a simple API and provides full GPU compute capability on CUDA and OpenCL capable devices

Outline

- What is GPU
- Why GPU Computing
- CPU vs. GPU
- GPGPU Framework
- Why OpenCL

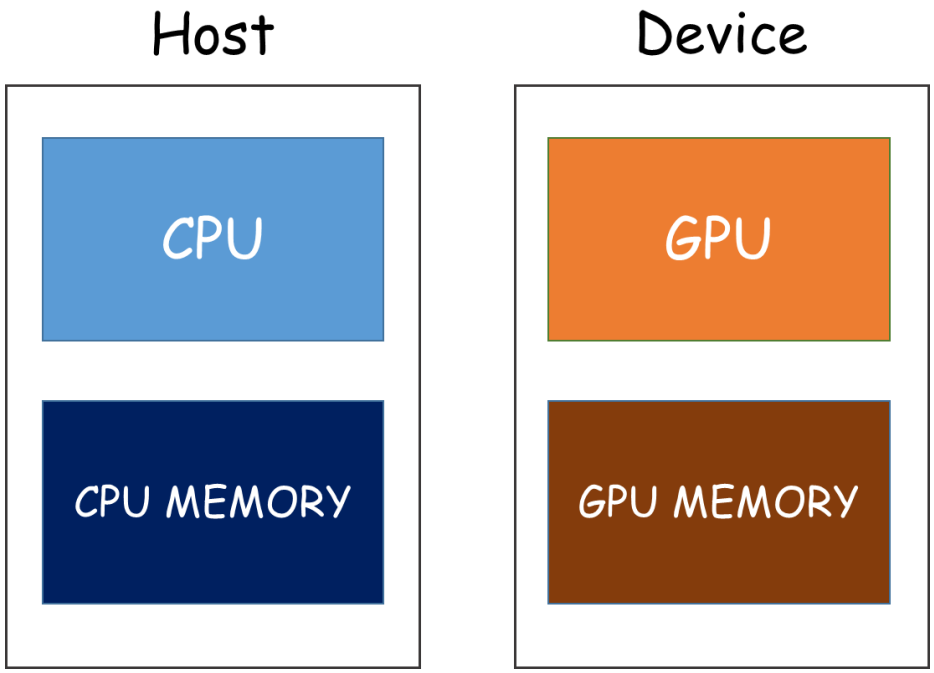
It's a Heterogeneous world

- **Heterogeneous computing is an integrated system that consists of different types of processor**
  - GPGPU (general purpose graphics processing unit)
  - DSP (digital signal processor)
  - FPGA (field-programmable gate array)
  - ASIC (application-specific integrated circuit)



What is Heterogeneous Computing

- Consist of host side and device side
- Simplified architecture

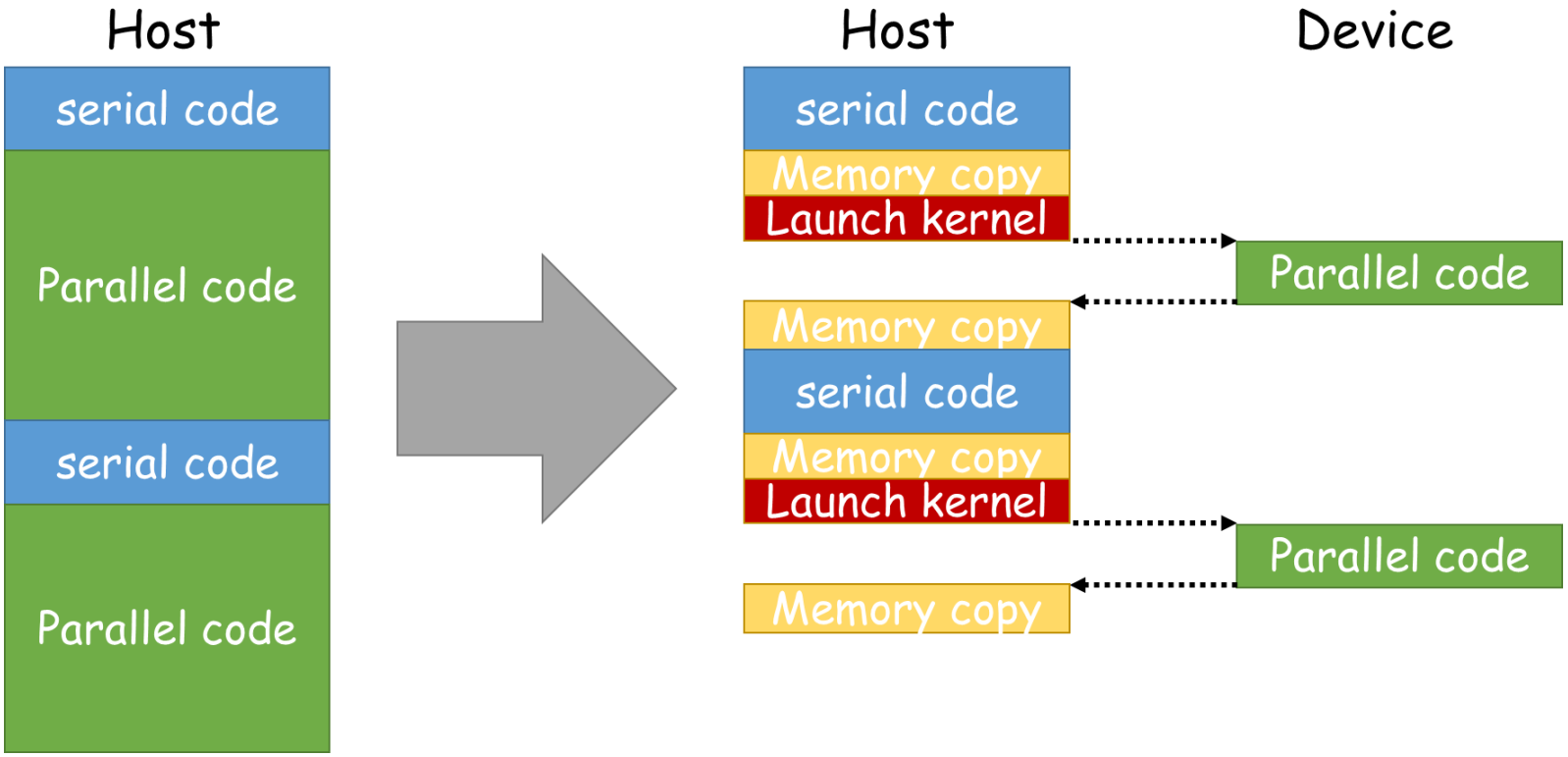


Why OpenCL

1. **Portable**
   - OpenCL routines can be executed on GPUs and CPUs from major manufacturers like AMD, NVIDIA, Intel, and so on.
2. **Nonproprietary**
   - it's based on a public standard, and you can freely download all the development tools you need.
3. **Parallel programming**
   - OpenCL provides full task-parallelism and data-parallelism
4. **Standardized Parallelization API**
5. **Extended version of C to allow parallel programming**

Why OpenCL

- Serial code executes in host (CPU)
- Parallel kernel executes in device (GPGPU)



Next Lecture Will Discuss

- Introduction to OpenCL
- OpenCL Framework
- The flow of OpenCL program

References

- Introduction to OpenCL.
  Cliff Woolley, NVIDIA
  http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf
- CUDA C Programming Guide
  http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3g8riFfwE
- CUDA Overview.
  Cliff Woolley, NVIDIA
  http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/02-cuda-overview.pdf
- Introduction to CUDA
  James Balfour, NVIDIA Research
  http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf
- The OpenCL Programming Book - Free HTML version
  Publisher: Fixstars
  Author: Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Jeongdo Son, Satoshi Miki
- OpenCL in action
  Author: Matthew Scarpino