a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - five weeks of lectures left
   - 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 27
Introduction to Supercomputing
Jan Verschelde, 24 October 2016

a Massively Parallel Processor: the GPU

1. Introduction to General Purpose GPUs
   - five weeks of lectures left
   - 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.

five weeks left in this course

Topics for the five weeks:

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; second edition, 2013.

The site `gpgpu.org` is a good start for many tutorials.

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
   - five weeks of lectures left
   - 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 `kepler`

- NVDIA Tesla K20c general purpose graphics processing unit
  1. number of CUDA cores: 2,496 (13×192)
  2. frequency of CUDA cores: 706MHz
  3. double precision floating point performance: 1.17 Tflops (peak)
  4. single precision floating point performance: 3.52 Tflops (peak)
  5. total global memory: 4800 MBytes
- CUDA programming model with `nvcc` compiler.

Two Intel E5-2670 (2.6GHz 8 cores) CPUs:

- 2.60 GHz $\times$ 8 flops/cycle = 20.8 GFlops/core;
- 16 core $\times$ 20.8 GFlops/core = 332.8 GFlops.

$\Rightarrow$ 1170/332.8 = 3.5. One K20c is as strong as 3.5 $\times$ 16 = 56.25 cores.

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

kepler versus pascal

NVIDIA Tesla K20 “Kepler” C-class Accelerator

- 2,496 CUDA cores, 2,496 = 13 SM $\times$ 192 cores/SM
- 5GB Memory at 208 GB/sec peak bandwidth
- peak performance: 1.17 TFLOPS double precision

NVIDIA Tesla P100 16GB “Pascal” Accelerator

- 3,586 CUDA cores, 3,586 = 56 SM $\times$ 64 cores/SM
- 16GB Memory at 720GB/sec peak bandwidth
- peak performance: 5.3 TFLOPS double precision

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.

a Massively Parallel Processor: the GPU

comparing flops on GPU and CPU

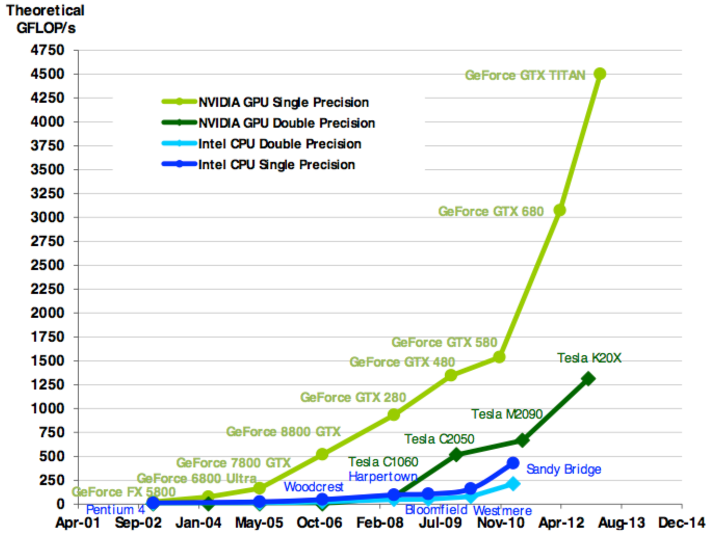
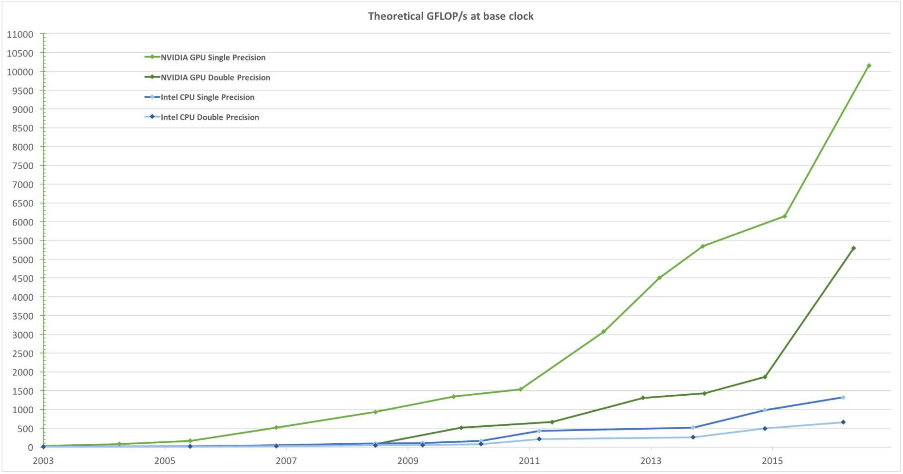


Figure 1 Floating-Point Operations per Second for the CPU and GPU

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

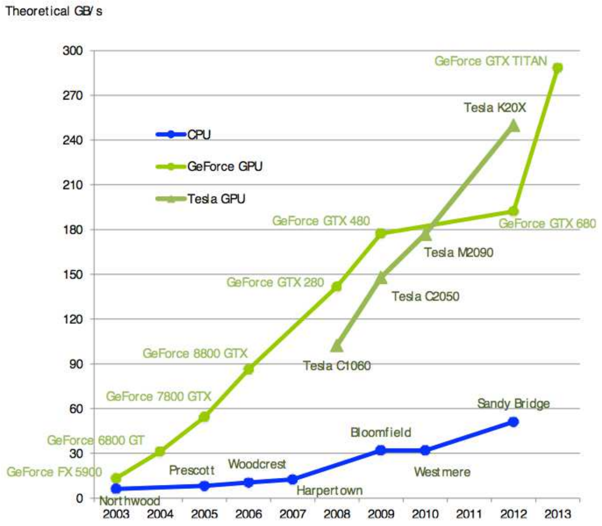
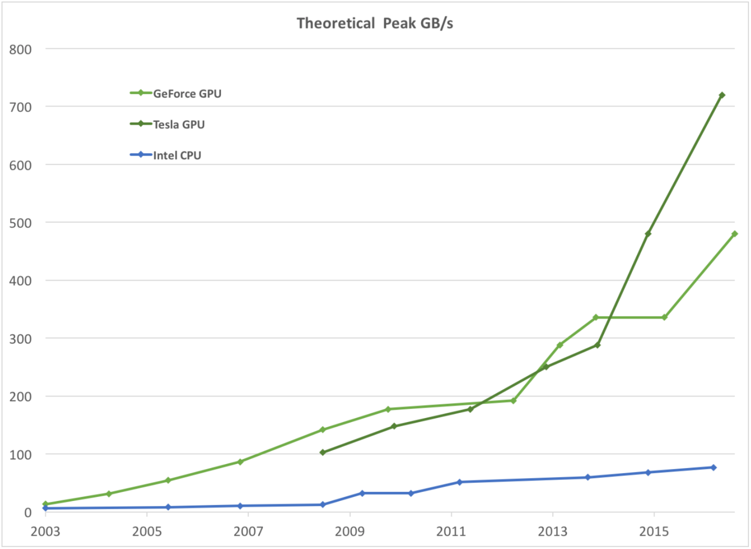


Figure 2 Memory Bandwidth for the CPU and GPU

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 kepler:

- Intel Xeon E5-2600: the memory bandwidth is 10.66GB/s.
- NVDIA Tesla K20c: the memory bandwidth is 143GB/s.

For pascal:

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

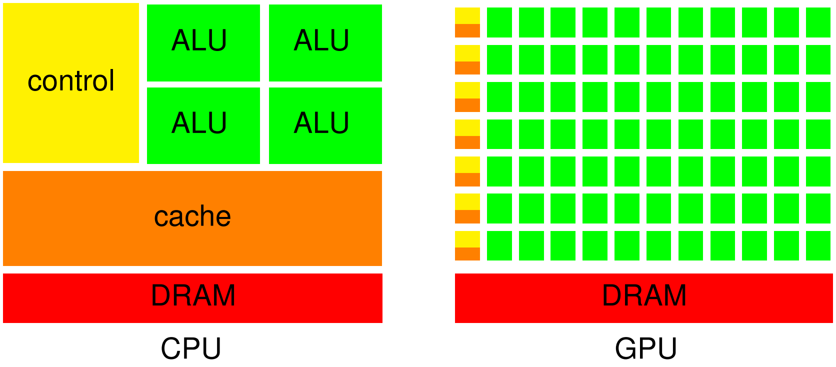
Straightforward parallel implementations on GPGPUs often achieve directly a speedup of 10, saturating the memory bandwidth.

a Massively Parallel Processor: the GPU

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.



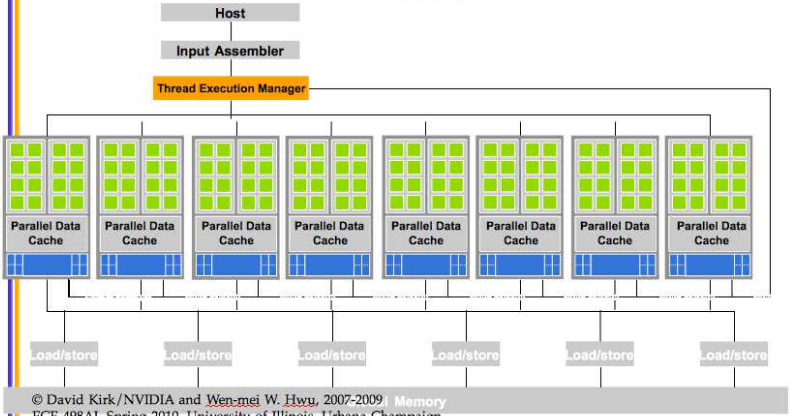
architecture of a modern GPU

- 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

GeForce 8800 (2007)

16 highly threaded SM's, >128 FPU's, 367 GFLOPS, 768 MB DRAM, 86.4 GB/S Mem BW, 4GB/S BW to CPU



© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2009
ECE 498AL Spring 2010, University of Illinois, Urbana-Champaign

our NVIDIA Tesla K20C and P100 GPUs

The K20C GPU has

- 13 streaming multiprocessors (SM),
- each SM has 192 streaming processors (SP),
- $13 \times 192 = 2496$ cores.

The P100 GPU has

- 56 streaming multiprocessors (SM),
- each SM has 64 streaming processors (SP),
- $56 \times 64 = 3,586$ 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
   - five weeks of lectures left
   - 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):

"*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.

On `kepler`, adjust your `.bashrc` (the \ means *ignore newline*):

```
export LD_LIBRARY_PATH\
  =$LD_LIBRARY_PATH:/usr/local/cuda-6.5/lib64

PATH=/usr/local/cuda-6.5/bin:$PATH; export PATH
```

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 `http://www.nvidia.com/object/tesla_software.html`

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).

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.

Exercise: visit `gpgpu.org`.

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