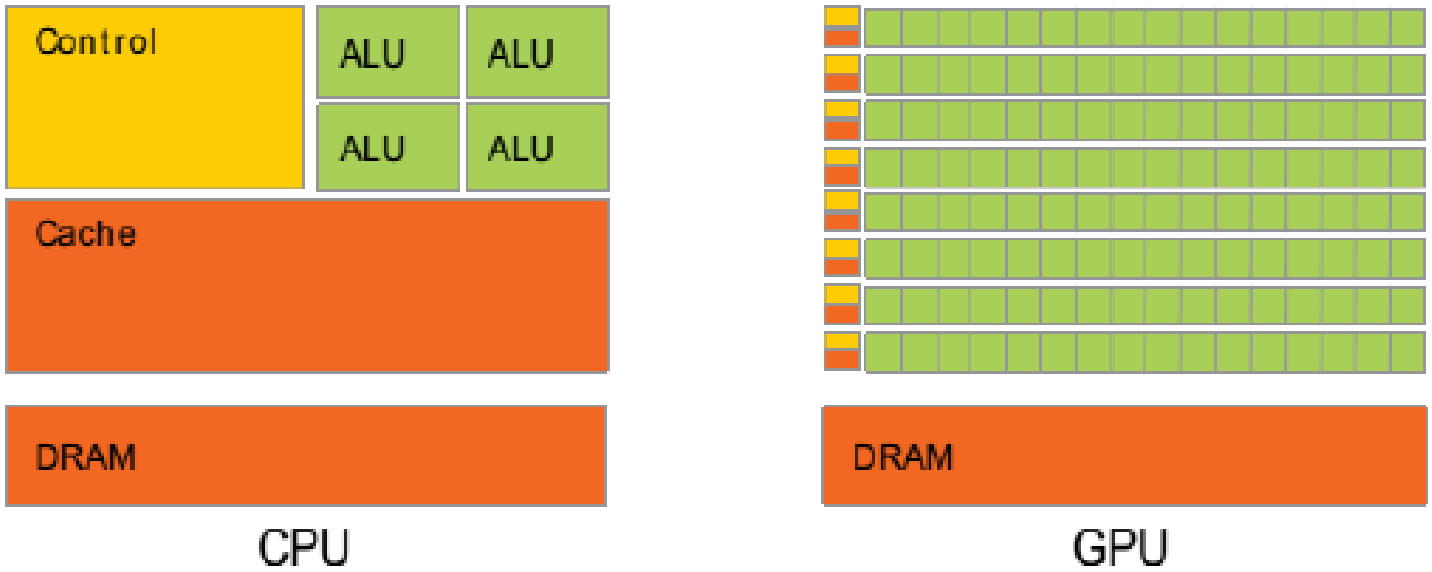
A bit about GPUs

Overview

- What does a GPU do?
- How does it do it?
- General-purpose GPU (GPGPU) programming, basic CUDA demo
- Why useful in deep learning?

What does a GPU do?

- Increased parallel computation
- CPU: latency optimized, GPU: throughput optimized
- Useful in graphics rendering




43
AMMO
100%
HEALTH
2 3 4
5 6 7
ARMS
100%
ARMOR
BULL 43 / 200
SHEL 0 / 50
ROKT 0 / 50
CELL 0 / 300

Doom (1993)


Ordinary VGA Quake
Resolution: 320x200
Colors: 256
Frame-rate: 30fps
OpenGL Quake on 3Dfx
Resolution: 640x480
Colors: 65,536
Frame-rate: 30fps

Quake (1996)

Quake 2 (1997)

Quake 3 (1999)

Doom 3 (2004)

Crysis 3 (2013)

Farcry 5 (2018)

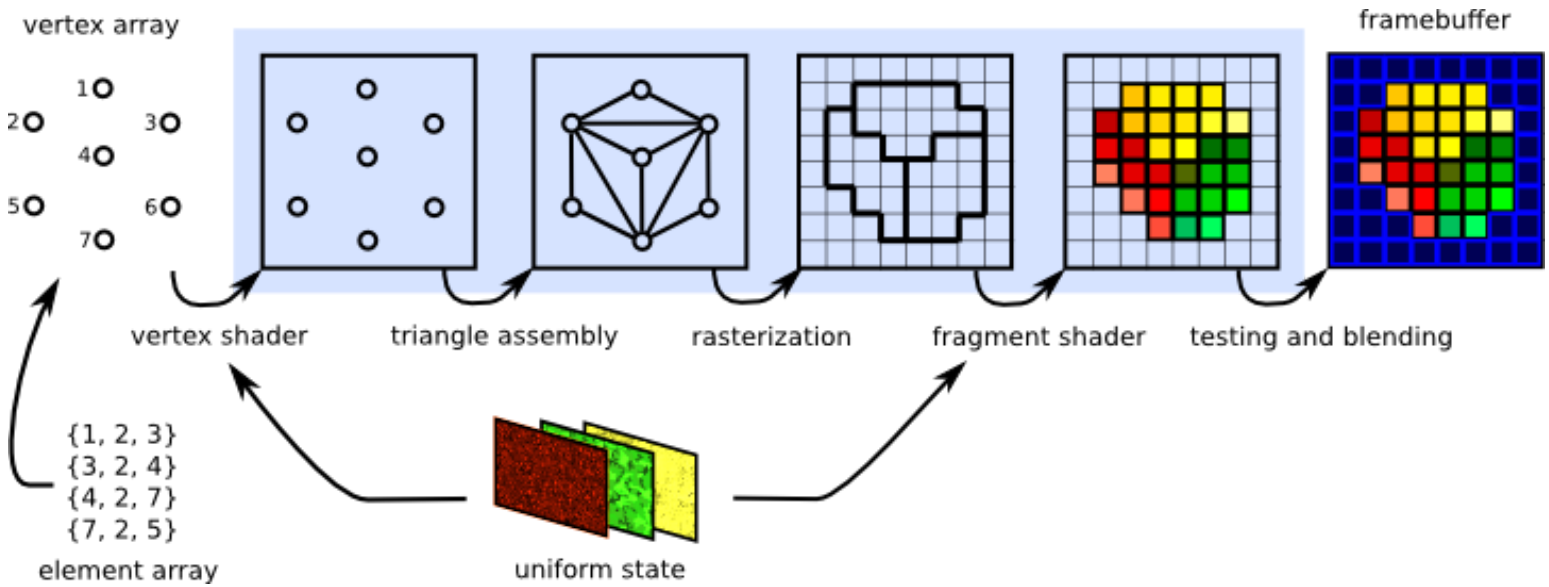
1990s

- 1992: Silicon Graphics releases OpenGL 1992
  Fixed rendering pipeline
- 1996: Microsoft release Direct3D
- 1999: Nvidia releases GeForce 256 ("first GPU")

2000s

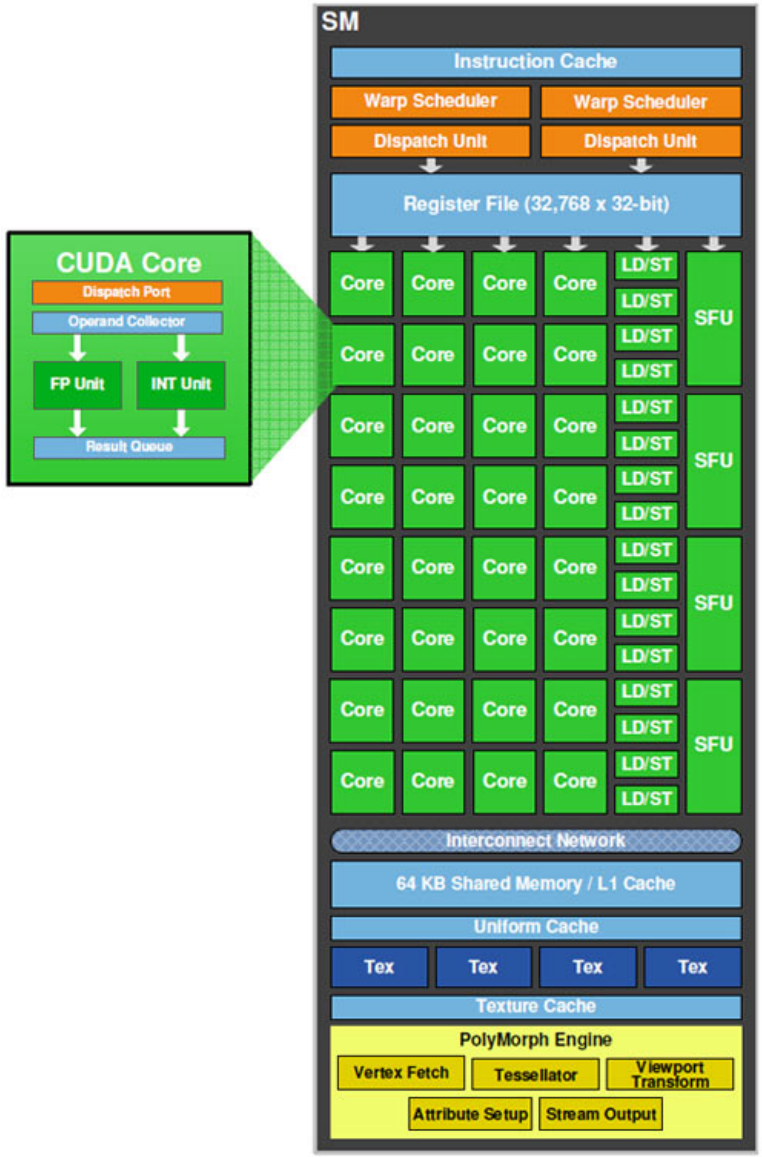
- 2003: OpenGL ES -- graphics for phones/tablets/etc
- 2004: OpenGL 2.0, fully programmable pipelines
- 2007: Nvidia releases CUDA -- GPGPUs
- 2009: Khronos group releases OpenCL
- 2009: OpenGL 3.2: geometry shaders -- increased flexibility
- 2011: WebGL. OpenGL ES for the browser

Rendering pipeline



- The same instruction on multiple data (SIMD)
- Not much branching (if statements)
- [vispy example]

How do (Nvidia) GPUs work?



Streaming Multiprocessor (SM)

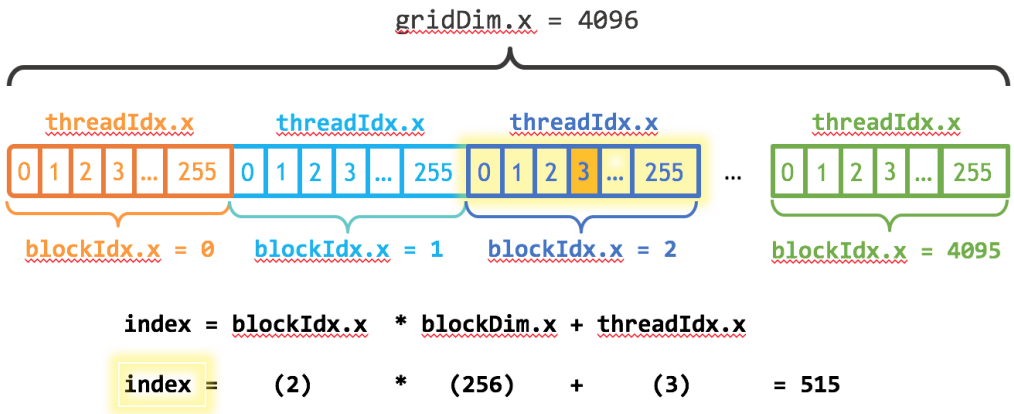
How do (Nvidia) GPUs work?

GM204 archiecture: 2048 cores

Comparisons with CPU

- GPU memory bandwidth: ~750GB/s, vs CPU: ~50GB/s
- GPU register memory: x30 larger than CPU (store convolution filters, network weights, etc)

A CUDA program



GPU:

- Split task into blocks, threads (and warps)
- grid_size = n_blocks * block_size
- Write kernel to perform task on each thread

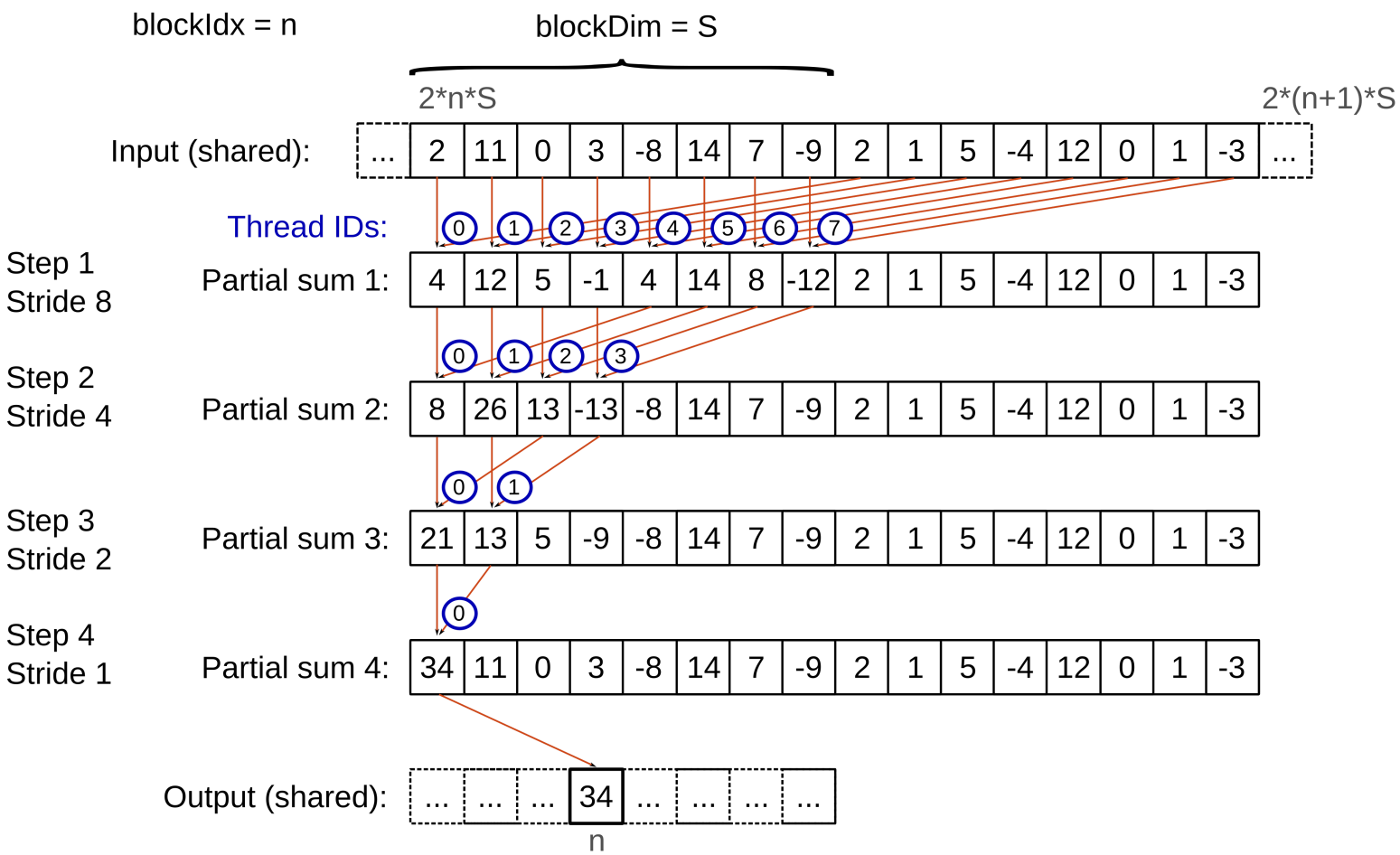
CPU:

- Initialize memory accessible by CPU and GPU
- Execute kernel
- Synchronize/Wait for all threads to finish
- Collect output from GPU, continue program

Simple example: hello world, vector addition

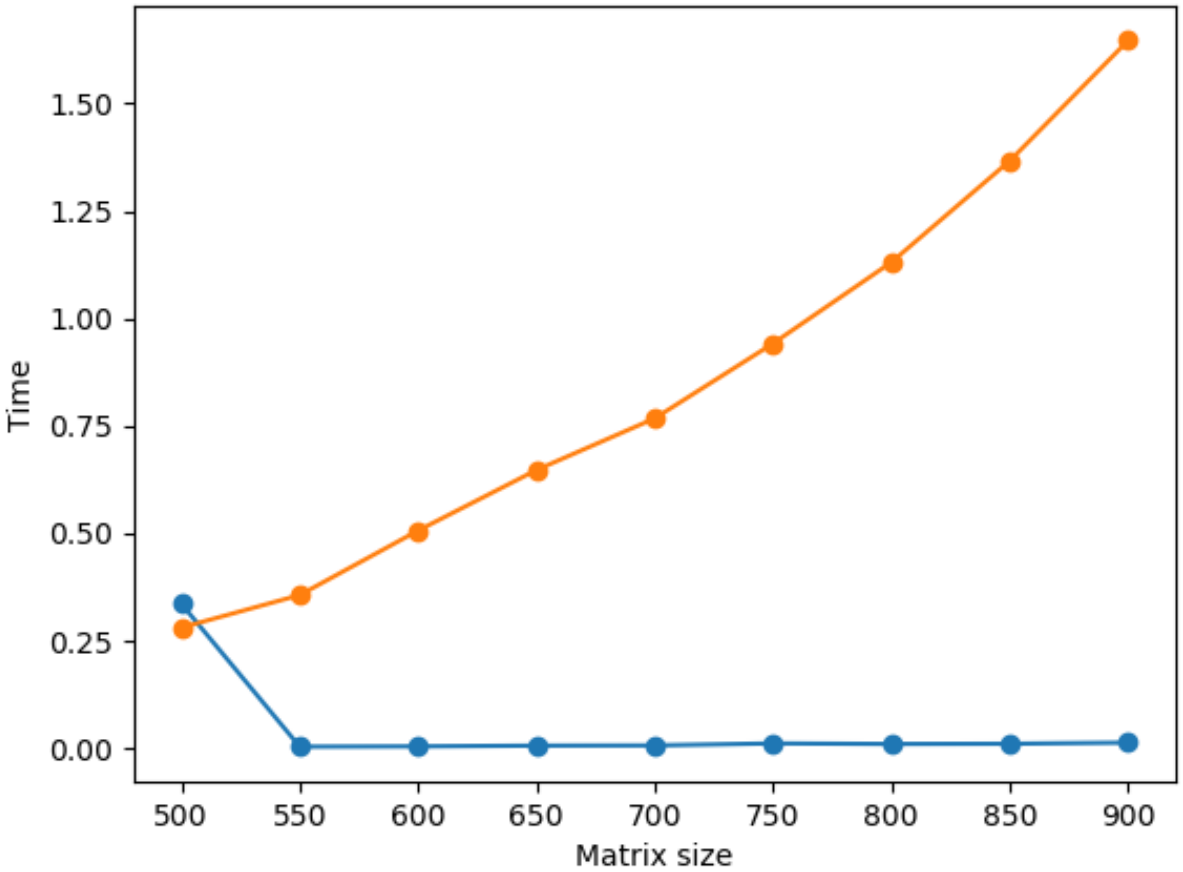
[Code]

More complicated example: dot product



$O(\log(N))$ steps

Matrix multiplication



$N \times N$ dense matrix multiply in TensorFlow.
Orange = CPU, Blue = GPU

Where are GPUs useful?

- Linear algebra
- PDEs
- Physics simulation
- Computer vision applications: segmentation, optic flow
- Neural networks: backprop is a long chain of matrix and element-wise multiplications

GPUs and deep learning

$$\mathbf{h}^n = f \circ (W^n \mathbf{h}^{n-1})$$

- A neural network: same instruction multiple data (SIMD) paralleism
- Minimal branching in training algorithms
- Steinkrau et al 2005: two-layer fully connected network. 3x speed up over CPU
- Chellapilla et al 2009: convolutional networks
- Raina et al 2009: 10-100x speedup over CPU

Performance tips

- Do preprocessing on CPU
- Optimize batch size -- overhead in copying data to GPU
- Use fused operations: multiple operations in same kernel. e.g. fused batch norm
- What else?

The future

- Convergence of CPU and GPU design:
  - increased autonomy for GPU cores --> faster sequential computation,
  - increased parallelism for CPUs --> faster parallel computation
- TPUs? Only for Google...
- Nvidia Turing architecture: real-time ray tracing

A couple resources to learn more

- These demos/slides:
  github.com/benlansdell/gpu_samples
- Easy OpenGL in python:
  conda install vispy
- Play with CUDA samples:
  cp -r /usr/local/cuda/samples ~/cuda_samples
- Use CUDA in python:
  https://mathema.tician.de/software/pycuda/

63 FPS (27-63)