Advanced Computer Architecture
ELEC3219 (2017/18)

Graphical Processing Unit (GPU)

Dr. Terrence Mak
tmak@ecs.soton.ac.uk
So as you can see, playing on an integrated GPU is very bad. Poor framerates, poor visuals, etc.

Now let's use the GeForce. (Note all stuttering in the GPU clips are all thanks to Fraps.)

(Seriously, Fraps KILLS your framerate when you record.)

In this video, I'll show you why playing games on a dedicated GPU is important.

The first 3 clips were recorded on an ATI Radeon X1200 integrated card.

The second 3 clips were recorded on an NVIDIA GeForce GT 440 dedicated card.
Overview

• What is GPU and why GPU
• Architecture
• CUDA
• The reality (my point of view)
The Evolution of GPU

• The Graphic Processing Unit (GPU) is a processor that was specialized for processing graphics
  – Only?
  – The GPU has recently evolved towards a more flexible architecture.

• Opportunity
  – We can implement *any algorithm*, not only graphics.

• Challenge
  – how to obtain efficiency and high performance.
Motivation

• GPU computing - key ideas
  – Massively parallel
  – Hundreds of cores
  – Thousands of threads.
  – Cheap
  – Highly available
  – Programmable: CUDA
Overview

- What is GPU and why GPU
- Architecture
- CUDA
- The reality (my point of view)
Motivation – NVidia’s POV

- NVidia model roughly 1999 (Classic pipeline):

  Processor per function

  Observation: All of these stages require the same basic functionality.

  Each stage needs different amount of time, depending on application. Therefore idle hardware.

nvidia.com
Unified processor

One processor core to do everything...
But lots of them.

NVidia 2002

nvidia.com
Streaming Processors, Texture Units, and On-chip Caches

- SP = Streaming Processors
- TF = Texture Filtering Unit
- TA = Texture Address Unit
- L1/L2 = Caches
GeForce 8800 (I)
G80 Architecture

• This is one "device"

• 128 Unified Streaming Processors (core)
  – Organised as 16 stream multiprocessors
  – (GTX280 has 24 stream multiprocessors)

• FB = Frame Buffer

• There is an L1 cache for each multiprocessor and an L2 cache shared by all multiprocessors
<table>
<thead>
<tr>
<th></th>
<th>GeForce 8800 Ultra</th>
<th>GeForce 8800 GTX</th>
<th>GeForce 8800 GTS</th>
<th>GeForce 8800 GT</th>
<th>GeForce 8800 GS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Processors</td>
<td>128</td>
<td>128</td>
<td>96</td>
<td>112</td>
<td>96</td>
</tr>
<tr>
<td>Core Clock (MHz)</td>
<td>612</td>
<td>575</td>
<td>500</td>
<td>600</td>
<td>550</td>
</tr>
<tr>
<td>Shader Clock (MHz)</td>
<td>1500</td>
<td>1350</td>
<td>1200</td>
<td>1500</td>
<td>1375</td>
</tr>
<tr>
<td>Memory Clock (MHz)</td>
<td>1080</td>
<td>900</td>
<td>800</td>
<td>900</td>
<td>800</td>
</tr>
<tr>
<td>Memory Amount</td>
<td>768MB</td>
<td>768MB</td>
<td>640MB or 320MB</td>
<td>512MB</td>
<td>384MB</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>384-bit</td>
<td>384-bit</td>
<td>320-bit</td>
<td>256-bit</td>
<td>192-bit</td>
</tr>
<tr>
<td>Memory Bandwidth (GB/sec)</td>
<td>103.7</td>
<td>86.4</td>
<td>64</td>
<td>57.6</td>
<td>38.4</td>
</tr>
<tr>
<td>Texture Fill Rate (billion/sec)</td>
<td>39.2</td>
<td>36.8</td>
<td>24</td>
<td>33.6</td>
<td>26.4</td>
</tr>
</tbody>
</table>
Threads

• Each core executes a thread, but
  – all cores in a stream multiprocessor execute the same instruction at the same time (**SIMD**)
  – NVidia call this **SIMT** – Single Instruction Multiple Thread)
  – Code is executed in a "warp" of 32 threads, i.e. time multiplexed over 4 clock cycles, on one stream multiprocessor
Memory model

- (This *is* confusing!)
- Device memory contains:
  - Global, constant, texture and *local* memory
- Each multiprocessor has *shared* memory
- Caches are not well explained
  - This fig. implies that L2 cache has constant and texture cache.
- And host memory on PC

Overview

• What is GPU and why GPU
• Architecture
• CUDA
• The reality (my point of view)
NVIDIA: GPU vendor

• GPU market: multi-billion dollars! (Nvidia, 30% market)
• Sold hundreds of millions of CUDA-capable GPUs
  – You will have one, if you are a game player
• New GPU generation every ~18 months
  – A bit like Moore’s law
• Strong support to GPU computing
  – Probably the best hardware for GPU
• Hardware side: developing flexible GPUs
• Software side: releasing and improving development tools
• Community side: support to academics.
  – Links: www.nvidia.com,
NVIDEA Chief Scientist
How a GPU looks like?

- Most computers have one
- Billions of transistors
- Computing
  - 1 Teraflop (Single precision)
  - 100 Gflops (Double precision)
- A heater for winter time!!!
- Used as Supercomputer
Applications

- Many can be found at the NVIDIA site!
Questions

• Question 1: Why accelerator technology today? If it has been around since the 70’s!
• Question 2: Can I really get 100x in my application?
• Question 3: CUDA? vendor dependent?
• Question 4: GPU computing = General-purpose on GPU?
Why accelerator technology today?

- Investment on GPU technology makes more sense today
- CPU uni-processor speedup is saturated, but GPU does not!
- Investing in an accelerator that gives a ~10x speedup:
  - 2004 speedup 1.52x per year
    - 10x today would be 1.3x acceleration in 5 years
- Consider the point that GPU parallel performance is doubling every 18 months!
Can I get 100x speedups?

- You can get hundred-fold speedup for some applications (algorithms)
- Depends on the non-parallel part: Amdahl’s law.
- Complex application normally make use of many algorithms
- Look for alternative ways to perform the computations that are more parallel
- Significance: An accelerated program is going to be as fast as its serial part!
GPU computing = General-purpose GPU?

• With CUDA you can program in C but with some restrictions.
• However, GPU are still highly specialized hardware.
  – SIMD not others
• Performance in the GPU does not come from the flexibility
  – But highly dedicated SIMD processing
Which one is better?

- Scooter
  - Well, depends
    - Romantic dinner or
    - Pizza transfer

- Sport car
Which one is better?

A team of Scooter

Deliver many packages within a reasonable timescale.

Sport car

Deliver a package as soon as possible
Now, comparing these two

High throughput and reasonable latency

Compute many jobs within a reasonable timeframe.

Low latency and reasonable throughput

Compute a job as fast as possible.
Comparison of NVIDIA GPU generations. Current generation: GT200. Table from NVIDIA Fermi whitepaper.

<table>
<thead>
<tr>
<th>GPU</th>
<th>G80</th>
<th>GT200</th>
<th>Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>681 million</td>
<td>1.4 billion</td>
<td>3.0 billion</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>128</td>
<td>240</td>
<td>512</td>
</tr>
<tr>
<td>Double Precision Floating Point Capability</td>
<td>None</td>
<td>30 FMA ops / clock</td>
<td>256 FMA ops /clock</td>
</tr>
<tr>
<td>Single Precision Floating Point Capability</td>
<td>128 MAD ops/clock</td>
<td>240 MAD ops / clock</td>
<td>512 FMA ops /clock</td>
</tr>
<tr>
<td>Special Function Units (SFUs) / SM</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Warp schedulers (per SM)</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Shared Memory (per SM)</td>
<td>16 KB</td>
<td>16 KB</td>
<td>Configurable 48 KB or 16 KB</td>
</tr>
<tr>
<td>L1 Cache (per SM)</td>
<td>None</td>
<td>None</td>
<td>Configurable 16 KB or 48 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>None</td>
<td>None</td>
<td>768 KB</td>
</tr>
<tr>
<td>ECC Memory Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concurrent Kernels</td>
<td>No</td>
<td>No</td>
<td>Up to 16</td>
</tr>
<tr>
<td>Load/Store Address Width</td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
</tbody>
</table>
CUDA architecture

- Support of languages: C, C++, OpenCL.
- Windows, Linux, OS X compatible.
**Compute Unified Device Architecture (CUDA)**

- Introduced by NVIDIA in late 2006
- CUDA is a compiler and toolkit for programming NVIDIA GPUs
- CUDA API extends the C programming language
- Runs on thousands of threads.
- Scalable model
- Objectives:
  - Express parallelism.
  - Give a high level abstraction from hardware.
Further points of CUDA

- Abstracting from the hardware
- Abstraction by the CUDA API
  - You don’t see every little aspect of the machine.
- Gives flexibility to the vendor
  - Change hardware but keep legacy code.
- Forward compatible.
- Automatic Thread management (can handle +100k threads)
- Multithreading
  - Hides latency and helps maximize the GPU utilization.
- Transparent for the programmer
- Limited synchronization between threads is provided
- No deadlock
  - SIMD, again
• Question?
CUDA language is vendor dependent?

- Yes, and nobody wants to locked to a single vendor.
- OpenCL is going to become an industry standard. (Some time in the future.)
- OpenCL is a low level specification, more complex to program with than CUDA C.
- CUDA C is more mature and currently makes more sense (to me).
- However, OpenCL is not “that” different from CUDA. Porting CUDA to OpenCL should be easy in the future.
- Personally, I’ll wait until OpenCL standard & tools are more mature.
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 (£200 to £1000).
• Number crunching: 1 card ~ 1 teraflop ~ small cluster.
• Small factor of the GPU.
• Important factors to consider: power and cooling!