CIS 565: GPU Programming and Architecture

Original Slides by: Suresh Venkatasubramanian
Updates by Joseph Kider and Patrick Cozzi

Administrivia
Meeting
- Monday and Wednesday
- 6:00 – 7:30pm
- Moore 212

- Recorded lectures upon request
- Website: http://www.seas.upenn.edu/~cis565/

Administrivia
- Instructor: Patrick Cozzi '99-'03: Penn State
- 00: Intel
- 02: IBM
- 03: IBM
- 04-Present: AGI
- 06-'08: Penn
- 11: 3D Engine Design for Virtual Globes

- Email: pjcozzi_at_siggraph.org
  - Include [CIS565] in subject
- Office Hours
  - Monday and Wednesday, 7:30-9:00pm
  - Location: SIG Lab

Administrivia
Teaching Assistant
- Jonathan McCaffrey
  - Email: jmccaf_at_seas.upenn.edu
- Office Hours: Tuesday and Thursday 3-4:30pm
- Office Location: SIG Lab
What is GPU (Parallel) Computing

- Parallel computing: using multiple processors to...
  - More quickly perform a computation, or
  - Perform a larger computation in the same time
  - PROGRAMMER expresses parallelism

- Prerequisites
  - CIS 460: Introduction to Computer Graphics
  - CIS 501: Computer Architecture
  - Most important:
    - C/C++ and OpenGL.

CIS 534: Multicore Programming and Architecture

- Course Description
  - This course is a pragmatic examination of multicore programming and the hardware architecture of modern multicore processors. Unlike the sequential single-core processors of the past, utilizing a multicore processor requires programmers to identify parallelism and write explicitly parallel code. Topics covered include the relevant architectural trends and aspects of multicore, approaches for writing multicore software by extracting data parallelism (threads and SIMD), thread-level parallelism, and task-based parallelism, efficient synchronization, and program profiling and performance tuning. The course focuses primarily on mainstream shared-memory multicore with some coverage of graphics processing units (GPUs). Cluster-based supercomputing is not a focus of this course.
  - Several programming assignments and a course project will provide students hands-on experience with programming, experimentally analyzing, and tuning multicore software. Students are expected to have a solid understanding of computer architecture and strong programming skills (including experience with C/C++).
  - We will not overlap very much

Administrivia

- Course Overview
  - System and GPU architecture
  - Real-time graphics programming with
    - OpenGL and GLSL
  - General purpose programming with
    - CUDA and OpenCL
  - Problem domain: up to you
  - Hands-on
**Goals**
- Program massively parallel processors:
  - High performance
  - Functionality and maintainability
  - Scalability
- Gain Knowledge
  - Parallel programming principles and patterns
  - Processor architecture features and constraints
  - Programming API, tools, and techniques

**Grading**
- Homeworks (4-5) 40%
- Paper Presentation 10%
- Final Project 40% + 5%
- Final 10%

**Administrivia**
- Bonus days: five per person
  - No-questions-asked one-day extension
  - Multiple bonus days can be used on the same assignment
  - Can be used for most, but not all assignments
- Strict late policy: not turned by:
  - 11:59pm of due date: 25% deduction
  - 2 days late: 50%
  - 3 days late: 75%
  - 4 or more days: 100%
- Add a Readme when using bonus days

**Administrivia**
- Academic Honesty
  - Discussion with other students, past or present, is encouraged
  - Any reference to assignments from previous terms or web postings is unacceptable
  - Any copying of non-trivial code is unacceptable
    - Non-trivial = more than a line or so
    - Includes reading someone else’s code and then going off to write your own.
Administrivia

- **Academic Honesty**
  - Penalties for academic dishonesty:
    - Zero on the assignment for the first occasion
    - Automatic failure of the course for repeat offenses

- Textbook: None
- Related graphics books:
  - Graphics Shaders
  - OpenGL Shading Language
  - GPU Gems 1 - 3
- Related general GPU books:
  - Programming Massively Parallel Processors
  - Patterns for Parallel Programming

Do I need a GPU?

- Yes: NVIDIA GeForce 8 series or higher
- No
  - Moore 100b - NVIDIA GeForce 9800s
  - SIG Lab - NVIDIA GeForce 8800s, two GeForce 480s, and one Fermi Tesla

Demo: What GPU do I have?

Demo: What version of OpenGL/CUDA/OpenCL does it support?
Aside: This class is about 3 things

- PERFORMANCE
- PERFORMANCE
- PERFORMANCE

- Ok, not really
- Also about correctness, "abilities", etc.
- Nitty Gritty real world wall-clock performance
- No Proofs!

Exercise

- Parallel Sorting

Credits

- David Kirk (NVIDIA)
- Wen-mei Hwu (UIUC)
- David Lubke
- Wolfgang Engel
- Etc. etc.

What is a GPU?

GPU: Graphics Processing Unit
Processor that resides on your graphics card.

GPUs allow us to achieve the unprecedented graphics capabilities now available in games.
What is a GPU?

- Demo: **NVIDIA GTX 400**
- Demo: Triangle throughput

Why Program the GPU?

- Compute
  - Intel Core i7 – 4 cores – 100 GFLOP
  - NVIDIA GTX280 – 240 cores – 1 TFLOP

- Memory Bandwidth
  - System Memory – 60 GB/s
  - NVIDIA GT200 – 150 GB/s

- Install Base
  - Over 200 million NVIDIA G80s shipped

How did this happen?

- Games demand advanced shading
- Fast GPUs = better shading
- Need for speed = continued innovation
- The gaming industry has overtaken the defense, finance, oil and healthcare industries as the main driving factor for high performance processors.

GPU = Fast co-processor?

- GPU speed increasing at cubed-Moore’s Law.
- This is a consequence of the data-parallel streaming aspects of the GPU.
- GPUs are cheap! Put a couple together, and you can get a super-computer.

So can we use the GPU for general-purpose computing?

Yes! Wealth of applications

- Data Analysis
- Motion Planning
- Voronoi Diagrams
- Geometric Optimization
- Physical Simulation
- Matrix Multiplication
- Conjugate Gradient
- Particle Systems
- Force-field simulation
- Molecular Dynamics
- Graph Drawing
- Database queries
- Sorting and Searching
- Range queries
- Image Processing
- Signal Processing
- Radar, Sonar, Oil Exploration
- … and graphics too!!

When does “GPU=fast co-processor” work?

Real-time visualization of complex phenomena

The GPU (like a fast parallel processor) can simulate physical processes like fluid flow, n-body systems, molecular dynamics

In general: Massively Parallel Tasks

When does “GPU=fast co-processor” work?

Interactive data analysis

For effective visualization of data, interactivity is key
When does “GPU=fast co-processor” work?

Rendering complex scenes

Procedural shaders can offload much of the expensive rendering work to the GPU. Still not the Holy Grail of “80 million triangles at 30 frames/sec”, but it helps.

* Alvy Ray Smith, Pixar.

Note: NVIDIA Quadro 5000 is calculated to push 950 million triangles per second.


Stream Programming

- A stream is a sequence of data (could be numbers, colors, RGBA vectors,...)
- A kernel is a (fragment) program that runs on each element of a stream, generating an output stream (pixel buffer).

Kernel = vertex/fragment shader
Input stream = stream of vertices, primitives, or fragments
Output stream = frame buffer or other buffer (transform feedback)
Multiple kernels = multi-pass rendering sequence on the GPU.

To program the GPU, one must think of it as a (parallel) stream processor.
What is the cost of a stream program?

- **Number of kernels**
  - Readbacks from the GPU to main memory are expensive, and so is transferring data to the GPU.
- **Complexity of kernel**
  - More complexity takes longer to move data through a rendering pipeline
- **Number of memory accesses**
  - Non-local memory access is expensive
- **Number of branches**
  - Divergent branches are expensive

What will this course cover?

1. **Stream Programming Principles**
   - OpenGL programmable pipeline
   - The principles of stream hardware
   - How do we program with streams?

2. **Shaders and Effects**
   - How do we compute complex effects found in today's games? Examples:
     - Parallax Mapping
     - Reflections
     - Skin and Hair
     - Particle Systems
     - Deformable Mesh
     - Morphing
     - Animation
3. GPGPU / GPU Computing

- How do we use the GPU as a fast co-processor?
  - GPGPU Languages: CUDA and OpenCL
  - High Performance Computing
  - Numerical methods and linear algebra:
    - Inner products
    - Matrix vector operations
    - Matrix Matrix operations
    - Sorting
    - Fluid Simulations
    - Fast Fourier Transforms
    - Graph Algorithms
    - And More...
  - At what point does the GPU become faster than the CPU for matrix operations? For other operations?

4. Optimizations

- How do we use the full potential of the GPU?
- What tools are there to analyze the performance of our algorithms?

What we want you to get out of this course!

1. Understanding of the GPU as a graphics pipeline
2. Understanding of the GPU as a high performance compute device
3. Understanding of GPU architectures
4. Programming in GLSL, CUDA, and OpenCL
5. Exposure to many core graphics effects performed on GPUs
6. Exposure to many core parallel algorithms performed on GPUs