GPU Programming

Lecture 1: Introduction

Miaoqing Huang
University of Arkansas
Spring 2016
Outline

Course Introduction

GPUs as Parallel Computers
  Trend and Design Philosophies
  Programming and Execution Model

Programming GPUs using Nvidia CUDA
Outline

Course Introduction

GPUs as Parallel Computers
  Trend and Design Philosophies
  Programming and Execution Model

Programming GPUs using Nvidia CUDA
Course Objective

▶ GPU programming
  ▶ Learn how to program massively parallel processors and achieve
    ▶ High performance
    ▶ Scalability across future generations
  ▶ Acquire technical knowledge required to achieve the above goals
    ▶ Principles and patterns of parallel programming
    ▶ Processor architecture features and constraints
    ▶ Programming APIs, tools and techniques
Lab assignments, Projects, and Course grading

- Constituent components of course grading
  - ~10 lab assignments: 60%
  - Quizzes: 10%
    - Frequently given, approximately once a week
  - Final exam: 30%

- All lab assignments are supposed to be carried out individually
Lab Equipment

- GPU computing
  - Fermi GPU: GeForce GTX 480
    - 480 Fermi CUDA cores
    - All the workstations in JBHT 237 are equipped with one GTX 480 GPU
  - Fermi GPU: Tesla C2075
    - 448 Fermi CUDA cores
    - Low power consumption, high double-precision floating-point performance
  - Kepler GPU: Tesla K20
    - 2496 CUDA cores
    - High double-precision floating-point performance, advanced features
Outline

Course Introduction

**GPUs as Parallel Computers**
- Trend and Design Philosophies
- Programming and Execution Model

Programming GPUs using Nvidia CUDA
Why Massively Parallel Processor

- A quiet revolution and potential build-up
  - Performance advantages again multicore CPU
    - GFLOPS: 1,000 vs. 100 (in year 2009)
    - Memory bandwidth (GB/s): 200 vs. 20
  - GPU in every PC and workstation - massive volume and potential impact
Different Design Philosophies

- **CPU: sequential execution**
  - Multiple complicated ALU design
  - Complicated control logic, e.g., branch prediction
  - Big cache

- **GPU: parallel computing**
  - Many simple processing cores
  - Simple control and scheduling logic
  - No or small cache
512 Fermi streaming processors in 16 streaming multiprocessors
Architecture of Kepler GPU

- 2,880 streaming processors in 15 streaming multiprocessors
Each streaming multiprocessor contains 192 single-precision cores and 64 double-precision cores
Basic Programming Model on GPU

Figure 2-1. Grid of Thread Blocks

Grid

Block (0, 0) Block (1, 0) Block (2, 0)

Block (0, 1) Block (1, 1) Block (2, 1)

Block (2, 1) Block (1, 1) Block (0, 1)

Block (2, 0) Block (1, 0) Block (0, 0)

➡ Issue hundreds of thousands of threads targeting thousands of processors
Execution Model on GPU

Time (clocks)

Frag 1 … 8
Runnable

Frag 9… 16

Frag 17 … 24

Frag 25 … 32

Stall
Execution Model on GPU

Time (clocks)

Frag 1 … 8
Frag 9… 16
Frag 17 … 24
Frag 25 … 32

Runnable

Stall
Execution Model on GPU

Time (clocks)

Frag 1 … 8
Frag 9 … 16
Frag 17 … 24
Frag 25 … 32

Runnable
Runnable
Runnable
Runnable

Stall
Stall
Stall
Stall
Execution Model on GPU

Increase run time of one group
To maximum throughput of many groups
How to deal with branches?

```
if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}
```

<unconditional shader code>

<resume unconditional shader code>
How to deal with branches?

Beyond Programmable Shading: Fundamentals

But what about branches?

ALU 1  ALU 2  ...  ALU 8  ...  ALU 8

Time

T T F T F F F F F

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional shader code>
How to deal with branches?

Not all ALUs do useful work!
Worst case: 1/8 performance
How to deal with branches?

Beyond Programmable Shading: Fundamentals

But what about branches?

\[
\begin{array}{cccccccc}
\text{ALU 1} & \text{ALU 2} & \ldots & \ldots & \text{ALU 8} \\
T & T & F & T & F & F & F & F
\end{array}
\]

\[
\begin{align*}
\text{Time (clocks)} & \\
1 & 2 & \ldots & \ldots & 8
\end{align*}
\]

\[
\begin{align*}
\text{if} & \ (x > 0) \ {\{ & \\
\ & \ y = \text{pow}(x, \ exp); & \\
\ & \ y \ *= \ Ks; & \\
\ & \ \text{refl} = \ y + \ Ka; & \\
\} & \ \text{else} \ {\{ & \\
\ & \ \ x = 0; & \\
\ & \ \text{refl} = \ Ka; & \\
\} & \}
\end{align*}
\]

\[
\begin{align*}
\text{<unconditional shader code>} & \\
\text{<resume unconditional shader code>}
\end{align*}
\]
Partition of an application

- Increase the data parallel portion of an application
  - Analyze an existing application
  - Expand the data volume of the parallel part
Outline

Course Introduction

GPUs as Parallel Computers
  Trend and Design Philosophies
  Programming and Execution Model

Programming GPUs using Nvidia CUDA
Nvidia CUDA

- **CUDA driver**
  - Handle the communication with Nvidia GPUs

- **CUDA toolkit**
  - Contain the tools needed to compile and build a CUDA application

- **CUDA SDK**
  - Include sample projects that provide source code and other resources for constructing CUDA programs
Program GPUs in JBHT 237

1. CUDA 6.5, including drivers and toolkits, has been installed on all computers in JBHT 237

2. The environment variables have been properly set to compile your code

3. Log into the machine using your uark id: username

4. Remote log into the machine, say, hostname is csce-t7500-xx (e.g., 01-14): ssh username@hostname.ddns.uark.edu
   ▶ On Windows platform, SSH Secure Shell Client 3.2.9 is free to download and use