CSE 599 I
Accelerated Computing - Programming GPUs

Course Introduction
Administrivia

Office hours:

Tentatively Thursday 1-3, or by appointment

Grading:

50% programming assignments, 50% final project

Textbook (very optional):


David B. Kirk and Wen-mei W. Hwu.

I can provide students with a code for a 30% discount on the textbook from Elsevier.

Computing resources:

Students will need access to a CUDA-capable device (I can help with this)
Lecture 1.1 – Course Introduction
Course Introduction and Overview
Course Goals

- Learn how to program heterogeneous parallel computing systems and achieve
  - High performance and energy-efficiency
  - Functionality and maintainability
  - Scalability across future generations
  - Portability across vendor devices

- Technical subjects
  - Parallel programming API, tools and techniques
  - Principles and patterns of parallel algorithms
  - Processor architecture features and constraints
People

- Wen-mei Hwu (University of Illinois)
- David Kirk (NVIDIA)
- Joe Bungo (NVIDIA)
- Mark Ebersole (NVIDIA)
- Abdul Dakkak (University of Illinois)
- Izzat El Hajj (University of Illinois)
- Andy Schuh (University of Illinois)
- John Stratton (Colgate College)
- Isaac Gelado (NVIDIA)
- John Stone (University of Illinois)
- Javier Cabezas (NVIDIA)
- Michael Garland (NVIDIA)
Outline

- Course Introduction
- Intro to CUDA C
- CUDA parallelism model
- Memory and data locality
- Thread execution / computational efficiency
- Memory performance
- Parallel patterns:
  - Stencil (convolution)
  - Prefix sum (aka scan)
  - Histogram
  - Sparse matrices
  - Graph search
- Floating point considerations
- Dynamic parallelism / recursion
- GPU as part of a heterogeneous system
- Case studies
- ???
Lecture 1.2 – Course Introduction
Introduction to Heterogeneous Parallel Computing
Objectives

– To learn the major differences between latency devices (CPU cores) and throughput devices (GPU cores)
– To understand why winning applications increasingly use both types of devices
Heterogeneous Parallel Computing

- Use the best match for the job (heterogeneity in mobile SOC)
CPU and GPU are designed very differently

**CPU**
- Latency Oriented Cores
- Chip
- Core
- Local Cache
- Registers
- SIMD Unit
- Control

**GPU**
- Throughput Oriented Cores
- Chip
- Compute Unit
- Cache/Local Mem
- Registers
- SIMD Unit
- Threading
CPUs: Latency Oriented Design

- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
GPUs: Throughput Oriented Design

- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
  - Require massive number of threads to tolerate latencies
    - Threading logic
    - Thread state
Winning Applications Use Both CPU and GPU

- CPUs for sequential parts where latency matters
  - CPUs can be 10X+ faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  - GPUs can be 10X+ faster than CPUs for parallel code

Core i7-6950X:
~300 GFLOPS

nVidia Titan X (Pascal):
~11,000 GFLOPS
# GeForce GTX 1050

## GPU Engine Specs:

<table>
<thead>
<tr>
<th></th>
<th>1050 Ti</th>
<th>1050</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA CUDA® Cores</td>
<td>768</td>
<td>640</td>
</tr>
<tr>
<td>Base Clock (MHz)</td>
<td>1290</td>
<td>1354</td>
</tr>
<tr>
<td>Boost Clock (MHz)</td>
<td>1392</td>
<td>1455</td>
</tr>
</tbody>
</table>

## Memory Specs:

<table>
<thead>
<tr>
<th></th>
<th>1050 Ti</th>
<th>1050</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Speed</td>
<td>7 Gbps</td>
<td>7 Gbps</td>
</tr>
<tr>
<td>Standard Memory Config</td>
<td>4 GB GDDR5</td>
<td>2 GB GDDR5</td>
</tr>
<tr>
<td>Memory Interface Width</td>
<td>128-bit</td>
<td>128-bit</td>
</tr>
<tr>
<td>Memory Bandwidth (GB/sec)</td>
<td>112</td>
<td>112</td>
</tr>
</tbody>
</table>
# GeForce GTX 1080

## GPU Engine Specs:
- **NVIDIA CUDA Cores**: 2560
- **Base Clock (MHz)**: 1607
- **Boost Clock (MHz)**: 1733

## Memory Specs:
- **Memory Speed**: 10 Gbps
- **Standard Memory Config**: 8 GB GDDR5X
- **Memory Interface Width**: 256-bit
- **Memory Bandwidth (GB/sec)**: 320
## NVIDIA TITAN X

### GPU Engine Specs:

<table>
<thead>
<tr>
<th>Spec'article'Class'</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA CUDA® Cores</td>
<td>3584</td>
</tr>
<tr>
<td>Base Clock (MHz)</td>
<td>1417</td>
</tr>
<tr>
<td>Boost Clock (MHz)</td>
<td>1531</td>
</tr>
</tbody>
</table>

### Memory Specs:

<table>
<thead>
<tr>
<th>Spec'article'Class'</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Speed</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>Standard Memory Config</td>
<td>12 GB GDDR5X</td>
</tr>
<tr>
<td>Memory Interface Width</td>
<td>384-bit</td>
</tr>
<tr>
<td>Memory Bandwidth (GB/sec)</td>
<td>480</td>
</tr>
</tbody>
</table>
Heterogeneous Parallel Computing in Many Disciplines

- Financial Analysis
- Scientific Simulation
- Engineering Simulation
- Data Intensive Analytics
- Medical Imaging
- Digital Audio Processing
- Digital Video Processing
- Computer Vision
- Biomedical Informatics
- Electronic Design Automation
- Statistical Modeling
- Numerical Methods
- Ray Tracing Rendering
- Interactive Physics
Lecture 1.3 – Course Introduction

Scalability in Heterogeneous Parallel Computing
Objectives

– To understand the importance and nature of scalability in parallel programming
Keys to Software Cost Control

- Scalability
Keys to Software Cost Control

- Scalability
  - The same application runs efficiently on new generations of cores
Keys to Software Cost Control

- Scalability
  - The same application runs efficiently on new generations of cores
  - The same application runs efficiently on more of the same cores
More on Scalability

- Performance growth with HW generations
  - Increasing number of compute units (cores)
  - Increasing number of threads
  - Increasing vector length
  - Increasing pipeline depth
  - Increasing DRAM burst size
  - Increasing number of DRAM channels
  - Increasing data movement latency
The GPU Teaching Kit is licensed by NVIDIA and the University of Illinois under the Creative Commons Attribution-NonCommercial 4.0 International License.
What is CUDA?

- A set of C language extensions
  - Requires a separate compiler (nvcc)
- A runtime API
- Development tools
When to use CUDA?

- When the program contains portions that are parallelizable
  - Task parallel or data parallel
- When parallelizable portions make up a significant amount of runtime:
  - If CUDA is used to achieve a 100X speedup of a portion of an application that accounted for 30% of runtime, the total applications speedup will be ~1.4X.
- When there is enough work to justify the overhead
  - Typically you want 5000+ active threads
Why is GPU programming hard?

- One must take care not to increase the computational complexity
- Algorithms are very often memory-bound rather than compute-bound
- Running times can be much more sensitive to dynamic input values
- Familiar sequential patterns such as recursion can map to very non-intuitive parallel patterns
Why learn CUDA?

Can’t I just use somebody else’s GPU-accelerated library?

- You may need features that the library doesn’t support
- It is still helpful to know what is happening at lower levels

Can’t I just wait until parallelization is handled automatically by the compiler?

- By some estimates, this could be ten years or more in the future
Why learn CUDA (specifically)?

nVidia has an installation base of about 1 billion CUDA-capable devices

Most concepts in this course generalize beyond CUDA (e.g. to OpenCL)