General Purpose Programming of Massively Parallel Graphics Processing Units (GPUs)

Reza Azimi

Department of Computer Science and Engineering, Shiraz University, Winter 88 – Spring 89

Single CPU Performance

- Past:
  - Doubled every 2 years for 30 years until 5 years ago.

- Current Situation:
  - Marginal improvement in the last 5 years.

- Main Reasons
  - Memory Wall
  - Power Wall
  - Processor Design Complexity
Memory Wall

Memory Bottleneck

<table>
<thead>
<tr>
<th>Relative Performance</th>
<th>10000</th>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU Frequency</td>
<td>DRAM Speeds</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CPU -- 2x Every 2 Years
- DRAM -- 2x Every 6 Years

Gap

source: www.opensparc.net

GPU Programming, Shiraz University, Winter 88/Spring 89, Reza Azimi

Memory Wall Implications

- Each memory access takes hundreds of CPU cycles.

- Increasing clock frequency doesn’t automatically improve performance anymore!
Stream Processing Characteristics

- Fairly simple computation on huge amount of data (streams)
  - Single Program Multiple Data (SPMD)

- Data Parallelism
  - e.g., Matrix Operations, Image Processing

- Limited Data Reuse
History

- Lots of research in computer architecture, but never got into the mainstream, until now…

- Huge amount of multi-media content generated on so many computers pushed stream processors into the mainstream.

- The most widely used stream processors:
  - **Programmable GPUs**

What is a GPU?

- Stands for “Graphics Processing Unit”
- Integration Scheme: a card on the motherboard
**GPUs: A Brief History**

- **Stage 1: Fixed Graphics Hardware**
  - Graphics-only platform
  - Very limited programmability

- **Stage 2: GPGPU**
  - Trick GPU to do general purpose computing
  - Programmable, but requires knowledge on computer graphics

- **Stream Processing Platforms**
  - High-level programming interface
  - No knowledge on Computer Graphics is required
  - Examples: NVIDIA’s CUDA, OpenCL

---

**How does it work?**

1. Copy Input Data
2. Call Computation
3. Return
4. Copy Result

---

*GPU Programming, Shinz University, Winter 88/Spring 88, Reza Azimi*
Motivations for Using GPUs

- Huge Computational Power
- Low Cost
- Availability

---

*GPU Programming, Shin* University, Winter 88/Spring 89 Reza Arjani

---

GPU’s Peak Computational Power

Source: NVIDIA

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>NV35</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>NV40</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>G70</td>
<td>3.0 GHz Core2 Duo</td>
</tr>
<tr>
<td>2006</td>
<td>G71</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>G80 Ultra</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>G80 Ultra</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>G80 Ultra</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>G92</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>GT200</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>GT200</td>
<td></td>
</tr>
</tbody>
</table>

*GT200 = GeForce GTX 280  G71 = GeForce 7900 GTX  NV35 = GeForce FX 5950 Ultra  
G92 = GeForce 9800 GTX  G70 = GeForce 7800 GTX  NV30 = GeForce FX 5800  
G80 = GeForce 8800 GTX  NV40 = GeForce 6800 Ultra*
Low Cost

- GPUs Main Use: Computer Games
  - Millions of GPUs sold every year.
  - Multi billion-dollar game industry helps pushing down the price.

- A GPU capable of 1 TeraFLOPs costs less than $500!
  - It would cost around $10000 to achieve this much computational power on a conventional platform.

Availability

- GPU’s Main Use: Computer Games
  - So, GPUs can be found everywhere (including here in Shiraz)

- Software Platform:
  - Free SDKs
  - Lots of documentation
  - Lots of code samples
  - So many people working on it => Industry Standards (e.g., OpenCL)
Desktop Supercomputers

Multiple GPUs can be installed on an inexpensive desktop motherboard, turning it to a multi-TeraFLOP supercomputer!

Success Stories

<table>
<thead>
<tr>
<th>Project</th>
<th>Speedup</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation of a Lattice Boltzmann (LB) kernel based on a D3Q13 model</td>
<td>100x</td>
<td>paper: TeraFLOP computing on a desktop PC with GPUs for 3D CFD DOI: 10.1080/10618560802238276</td>
</tr>
<tr>
<td>Solving 2D heat conduction CFD problems using CUDA. Using Red-Black Gauss-Seidel with SOR</td>
<td>17x</td>
<td>presentation by: Aaron Shinn, Mechanical Eng. Dept., UIUC</td>
</tr>
</tbody>
</table>

For a detailed list of applications accelerated by GPUs look at: http://www.nvidia.com/object/cuda_home.html
Why GPUs are so Fast: GPUs vs. CPUs

CPU
- Control
- ALU
- ALU
- ALU
- Cache

DRAM
- Number of Cores: 8-16
- Cache Size: 4-16MB
- Complex Control Unit
- DRAM Size: Tens of GB

~20GB/s

GPU
- Control
- ALU
- ALU
- ALU
- Cache

DRAM
- Number of Cores: Hundreds
- Cache Size: < 1MB
- Simple Control Unit
- DRAM Size: < 4GB

~150GB/s

GPU Architecture (NVIDIA)

Host
- Input Assembler
- Thread Execution Manager

Parallel Data Cache
- Stream Processors: Up to 500

DRAM

Load/Store

Source: NVIDIA

GPU Programming, Shinaz University, Winter 88/Spring 89, Reza Azimi
GPU Architecture (AMD)

Source: AMD

CPU-GPU Interaction

- **GPU:**
  - Runs a number of compute-intensive *Kernels*

- **CPU:**
  - Provides data for GPU and manages it
  - Runs the rest of the computation

*GPU Programming, Shinz University, Winter 88/Spring 88, Reza Azimi*
Designing a Kernel: Basic Example

Sequential Code

/* N is large */
for (i = 0; i < N; i++) {
    a[i] = a[i] * c;
}

Parallel Code

/* assuming you have one core per array entry */
i = coreIndex;
a[i] = a[i] * c;

But we don’t have a million cores!

Partitioning Input Data

/* Code executed by each core */
partitionSize = N / numberOfCores;
start = myCoreIndex * partitionSize;
end = start + partitionSize - 1;

for (i = start; i < end; i++) {
    a[i] = a[i] * c;
}

GPU Programming, Shin University, Winter 88/Spring 89, Reza Azimi
Programming Environments

- Brook+
- RapidMind
- AMD’s Close-to-Metal (CTM)
- NVIDIA’s CUDA
- OpenCL
  - Industry Standard

This Course Goals

- Learn how to program GPUs
  - Using NVIDIA’s CUDA Technology

- Learn how to accelerate compute-intensive applications using GPUs

- Understand highly parallel architecture of GPUs, its potentials, its limitations
Prerequisites

- Basic knowledge on
  - Computer Architecture
  - Operating Systems
  - Programming in C
  - Algorithm Design

Outline of the Materials

- Refreshing our memories
  - Computer Architecture, Threads, C Programming and Debugging Environment
- Basic Programming in CUDA
- GPU Architecture
- CUDA Performance Optimization Techniques
- Classic Parallel Problems on GPU
- Case Study Applications
Grading Structure

- Six Programming Exercises: 30%
- Programming Project: 40%
  - Accelerate a real application
  - Any compute-intensive, parallelizable application can be a candidate
  - Applications can be picked from your domain of expertise, or from other disciplines (e.g., electrical engineering, mechanical engineering, physics, etc.)
- Final Exam: 30%

Programming Environment

- GPUs
  - NVIDIA’s GT280 (240 cores) and 8800GT (128 cores)
- CPUs
  - 2x4 core Xeon 5405
- Operating System
  - Linux (OpenSuse, both 64bit and 32bit)
- Software Development Kit
  - NVIDIA’s CUDA
Administrative

- Course Web Page
  - [http://www.cse.shirazu.ac.ir/~azimi/gpu88](http://www.cse.shirazu.ac.ir/~azimi/gpu88)
  - Will use moodle if it works.

- Student Hours
  - Monday 8-9am
  - Wednesday 11-12pm
  - Any other meeting with prior appointment

- Graders
  - Amin Abbasi
  - Reza Mokhtari