An Introduction to CUDA

James Gain
jgain@cs.uct.ac.za
29 April – 3 May 2013
Motivation: Why GPU?

Kepler Series GPUs vs. Quad-core Sandy Bridge CPUs

- Kepler delivers equivalent performance at:
  - 1/18th the power consumption
  - 1/9th the cost

So

- Awesome performance per Watt
- Awesome performance per $

Price/Performance/Power:

- NVIDIA GeForce GTX 680 3,090 GFLOPS at 195 W for $460
- 3,090 GFLOPS / 195 W ≈ 15.8 GFLOPS/W
- 3,090 GFLOPS / $460 ≈ 6.7 GFLOPS/$

“The Soul of a Supercomputer in the Body of a GPU”

Which costs more: buying a Playstation or running it continuously for a year?
Is a speedup of 1400x for a GPU implementation plausible?
The Effect of Memory Bandwidth

Theoretical Peak FLOPS
- An unrealistic measure obtained by multiplying the ALU throughput by number of cores
- A good measure would also account for I/O performance, cache coherence, memory hierarchy, integer ops

GPUs win again on memory transfer
- On average 7X higher internal memory bandwidth
- 177.4 GB/s (GTX4xx,5xx) vs 25.6 GB/s (Intel Core i7)
- However CPU - GPU transfer much slower (~8 GB/s)
Case Study: Molecular Docking

- 1400-fold speed-ups are possible for the right problem and with sufficient development effort

- Coarse-grained replica exchange Monte Carlo protein docking
  - A statistical sampling approach to aligning molecules

- Viral capsid construction:
  - 680,000 residues, 100 million iterations
  - 3000 years on a single CPU
  - < 1 year on a cluster of GPUs
A Difference in Design Philosophies

CPU

Control

ALU

ALU

ALU

Cache

DRAM

GPU

DRAM
Design Implications

CPU:
- Optimized for sequential code performance
- Lower memory bandwidths (< 50 GB/s)
- Large cache and control

GPU:
- Optimized for parallel numeric computing
- Higher memory bandwidths (> 150 GB/s)
- Small cache and control

Ideal is a combination of CPU and GPU, as provided by CUDA
Motivation: Why CUDA?

What is it?
- Compute Unified Data Architecture (CUDA)
- Offers control over both CPU and GPU from within a single program
- Written in C with a small set of NVIDIA extensions

Better than the GLSL/HLSL/Cg alternative:
- Forcing a square peg into a round hole (forcing a Computer Graphics program to be general purpose)

More features:
- Shared memory, scattered reads, fully supported integer and bitwise ops, double precision if needed
Motivation: Why not GPU?

- GPU’s are not a cure-all
- Not suited to all algorithms
  - Work needs to be divisible into small largely-independent fragments
  - Does not cope well with recursive highly-branching tightly-dependent algorithms
- Difficult to program
  - Relatively easy to get moderate speedups (2-5X)
  - Better performance requires understanding of the architecture and careful tuning
Feeding the Beast

- Need thousands of threads to:
  - Saturate processors
  - Hide data transfer latency
  - Handle other forms of synchronisation

- Supported by low thread scheduling overhead

- But not all problems are amenable to such a decomposition
Effective memory use is absolutely crucial to GPU acceleration.

- Computation per SM/SMX: ~24,000 GB/s
- Register Memory: ~8,000 GB/s
- Shared Memory: ~1,600 GB/s
- Global Memory: 177 GB/s
- CPU to GPU: ~6 GB/s
Motivation: Why *not* CUDA?

- Proprietary product
  - Only supported on NVIDIA GPUs
- Stripped down version of C:
  - No recursion (< cc2.0), no function pointers
- Branching may damage performance
- Double precision deviates in small ways from IEEE 754 standard
## CUDA Compared

<table>
<thead>
<tr>
<th>Platform</th>
<th>✗</th>
<th>✔</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shader Languages (GLSL, Compute)</td>
<td>• Contorted code (for a non-graphics fit)</td>
<td>• Supported on more GPUs</td>
</tr>
<tr>
<td></td>
<td>• More passes required</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Restricted access to features</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Harder to learn</td>
<td></td>
</tr>
<tr>
<td>OpenCL</td>
<td>• Still underdeveloped</td>
<td>• Cross-platform standard</td>
</tr>
<tr>
<td></td>
<td>• Somewhat verbose</td>
<td>• Similar in design to CUDA</td>
</tr>
<tr>
<td>ATI Stream</td>
<td>• Late to the party</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Also proprietary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DEAD?</td>
<td></td>
</tr>
</tbody>
</table>
Implications of Computer Graphics

Legacy

Games Industry:
- Constant drive for performance improvement
- Commoditisation – high demand leads to high volumes, lower prices

Massively multi-threaded:
- Millions of incoming polygons and outgoing pixels, each largely independent
- Best supported by millions of lightweight threads
Computation Implications

Coherence:
- Nearby pixels / vertices have similar access patterns and computation
- Consequently, GPU’s expect memory access and branch coherence

Single-precision floating point:
- Geometric operations in CG require floating point but don’t need the accuracy of double precision
- Consequently, integers and doubles weren’t well supported until recently
Memory Implications

**Memory Bandwidth:**
- Must transfer millions of elements from vertex buffers and to the framebuffer or the frame rate stalls
- Consequently, memory transfers have high bandwidth

**Textures:**
- Images that are wrapped onto geometry to cheaply provide additional realism
- Consequently, GPU’s support large on-chip memories with high bandwidth coherent access
CUDA Programming Model

- Data parallel, compute intensive functions should be off-loaded to the device
- Functions that are executed many times, but independently on different data, are prime candidates
  - i.e. body of for-loops
- CUDA API:
  - Minimal C extensions
  - A host (CPU) component to control and access GPU(s)
  - A device component
  - CUDA source files must be compiled with the nvcc compiler
With current barriers to higher clock speeds, Parallel Computing is recognised as the only viable way to significantly accelerate applications.

Many-core GPU architectures are a strong alternative to multi-core (dual-core, quad-core, etc) CPU architectures.

Programming in CUDA can provide considerable speedup for numerically intensive applications.

- But more significant speedups often require extensive tuning and algorithm restructuring.

**Take-home Messages**

1. Not all problems are suited to a GPU solution
2. Refactoring and careful tuning required for best performance
Slide References