CUDA

Matthew Joyner, Jeremy Williams
Agenda

- What is CUDA?
- CUDA GPU Architecture
- CPU/GPU Communication
- Coding in CUDA
- Use cases of CUDA
- Comparison to OpenCL
What is CUDA?
CUDA is a parallel computing platform and programming model developed by NVIDIA.

- Simplifies the process of performing general purpose computing on the GPU.
- Capable of scheduling 30,720 threads per GPU.
CUDA GPU Architecture
Kepler GK110

- 15 SMX
- 6x 64-bit Memory Controllers
- 1.5 MB L2 Cache
Streaming Multiprocessor

- CUDA GPU is made from a scalable array of Streaming Multiprocessors
- Each multiprocessor is independent
- SMX is the multiprocessor used in the Kepler architecture
SMX

- 192 x Single-Precision CUDA cores
- 64 x Double-Precision CUDA cores
- 32 x Special Function Units
- 32 x Load/Store Units
- 16 x Texture Units
- 4 x Warp Schedulers
  - 2 x Dispatch Units
- 65,536 x 32-bit Registers
- 64KB Shared Memory/L1 Cache
- 48KB Read-Only Data Cache
- Max 2048 concurrent threads
  - With 15 SMX units this results in 30,720 concurrent threads per GK110
Memory

- **SMX Level**
  - 65,536 x 32-bit Registers
    - 255 Max Registers per thread
  - 64KB of Shared Memory/L1 Cache
    - Supports 16/48-32/32-48/16 split
    - Can be configured per application or per kernel
  - 48KB of Read-Only Data Cache
    - Used to be texture cache and had to go through texture units
    - Kepler allowed the processing cores access

- **GPU Level**
  - 1.5MB of L2 Cache
    - Primary point of data unification across SMX units
  - DRAM
    - Amount varies based on Graphics Card make/model

- **Uses Single-Error Correct Double-Error Detect (SECDED) Error Correction Code (ECC)**
  - Read-Only Data Cache also supports single-error correction using a parity check
CPU/GPU Communication
Communication

- Code sent from CPU to GPU is in Parallel Thread Execution (PTX)
- PTX is a pseudo-assembly language
- Graphics Drivers convert PTX into executable binary
Communication

- **Kernel** - Executed by CPU and sent to GPU
  - Assigned to GPU
  - A Kernel contains 1 Grid

- **Grid** - 2D array of Blocks
  - Assigned to GPU

- **Block** - Group of concurrent threads
  - Assigned to SMX
  - Broken down into Warps by SMX scheduler
  - Largest unit of synchronizable threads

- **Warp** - Group of 32 threads
  - Assigned to Warp Scheduler in SMX

- **Thread** - Sequential Instructions
  - Assigned to individual core
A GPU work queue can execute 1 Kernel at a time

Previous generations had a single work queue per GPU
- Resulted in poor GPU utilization due to inability to saturate the GPU
- Multiple Processes could submit to queue
  - Caused scheduler to flag artificial dependencies in Kernels from different processes

Kepler implemented Hyper-Q
- GPU now has 32 work queues
- Allows for better saturation of GPU
- Multiple Processes can submit to different queues
  - Removes artificial dependencies
Dynamic Parallelism

- Previous Generations
  - Kernels could only be launched by CPU
- Kepler
  - Kernels can be launched by both the CPU and the GPU
  - Allows for less communication between the CPU and GPU

*Makes GPU Computing Easier & Broadens Reach*
GPUDirect

- Allows the GPU to communicate with other devices directly
- Eliminates CPU bandwidth and latency bottlenecks
- Currently supports a variety of IB adapters, NICs, and SSDs
- Will complement NVIDIA’s planned Unified Virtual Address Space (UVA)
Programming for CUDA
Code modification

- Makes use of OpenACC standard compiler directives (similar to OpenMP)
- Program can be optimized for GPU acceleration without modifying the code itself, using only compiler directives
- Compiler directives can instruct the compiler to automatically parallelize a given code block, or can be more specific.
Example of Compiler Directives

```c
#include <stdio.h>
#define N 1000000

int main(void) {
    double pi = 0.0f; long i;
    #pragma acc parallel loop reduction(+:pi)
    for (i=0; i<N; i++) {
        double t = (double)((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%16.15f\n",pi/N);
    return 0;
}
```
CUDA-ready Libraries

- Many common math and science libraries have versions optimized for CUDA GPUs
- Developers can simply replace their traditional libraries with CUDA libraries to take advantage of speed increases up to 10x depending on operation
CUDA-ready Libraries

- Popular Math/Science operations with CUDA libraries:
  - BLAS
  - FFT
  - LAPACK
  - OpenCV
  - Sparse Matrix functions
  - Random Number Generation
  - Ocean Simulation
  - Many others
Use Cases of CUDA
Use Cases of CUDA - Titan Supercomputer

- Finished in October 2012 at the Oak Ridge National Laboratory
- #1 on the TOP500 list until June 2013
- Utilizes 18,688 compute nodes with both an AMD multi-core CPU and a NVIDIA K20x CUDA-based GPU
- First hybrid system (GPU and CPU) to perform at over 10 petaFLOPS (17.59 petaFLOPS LINPACK)
- Gemini Interconnect is capable of transmitting tens of millions of MPI messages per second at sub-microsecond latency, using a 3D torus network topology
Scientific Applications accelerated by Titan’s CUDA upgrade

- Enhanced modeling of:
  - Nuclear reactors
  - Supernovae
  - Molecular physics of combustion
  - Earth’s atmosphere

- Some of these applications saw improvements of over 2x once GPUs were utilized
Scientific Applications accelerated by CUDA

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seismic Database</td>
<td>66 to 100X</td>
</tr>
<tr>
<td>Mobile Phone Antenna Simulation</td>
<td>45X</td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td>240X</td>
</tr>
<tr>
<td>Neuron Simulation</td>
<td>100X</td>
</tr>
<tr>
<td>MRI Processing</td>
<td>245 – 415X</td>
</tr>
<tr>
<td>Atmospheric Cloud Simulation</td>
<td>50X</td>
</tr>
</tbody>
</table>

From http://www.nvidia.com/object/IO_43499.html
CUDA vs. OpenCL
OpenCL

- Not tied to a specific GPU architecture, built to enable acceleration by any co-processor (ASIC, DSP, FPGA, etc.)
- Implemented as a set of libraries and functions, places limitations on what programming constructs can be used
- Requires a major re-write to port existing code
Comparison

- CUDA is tied to a single hardware platform (NVIDIA’s GPUs) while OpenCL can be implemented in any kind of co-processor
- Since OpenCL requires re-write and CUDA can be used with only compiler directives, there are many more libraries already ported to CUDA
- Tradeoff between code complexity and portability
Questions?