Reading Assignment #5 (until Mar. 10)

Read (required):

- Programming Massively Parallel Processors book, Chapter 3 (*Introduction to CUDA*)
- Programming Massively Parallel Processors book, Chapter 4 (*CUDA Threads*) until (including) 4.3

Read (optional):

- NVIDIA Fermi graphics (GF100) white paper:
- NVIDIA Fermi compute white paper:
Quiz #2: Mar. 6

Organization

• First 30 min of lecture
• No material (book, notes, ...) allowed

Content of questions

• Lectures (both actual lectures and slides)
• Reading assignments
• Programming assignments (algorithms, methods)
• Solve short practical examples
“Compute Unified Device Architecture”

Extensions to C(++) programming language

- `__host__`, `__global__`, and `__device__` functions
- Heavily multi-threaded
- Synchronize threads with `__syncthreads()`, ...
- Atomic functions
  (before compute capability 2.0 only integer, now also float)

- Compile `.cu` files with NVCC
- Uses general C compiler (Visual C, gcc, ...)
- Link with CUDA run-time (`cudart.lib`) and cuda core (`cuda.lib`)
CUDA Multi-Threading

- CUDA model groups threads into blocks; blocks into grid

- Execution on actual hardware:
  - Block assigned to SM (up to 8 blocks per SM)
  - 32 threads grouped into warp
Threads in Block, Blocks in Grid

- Identify work of thread via
  - threadIdx
  - blockIdx

```
float x = input[threadIdx];
float y = func(x);
output[threadIdx] = y;
```

Thread Block 0
```
float x = input[threadIdx];
float y = func(x);
output[threadIdx] = y;
```

Thread Block 1
```
float x = input[threadIdx];
float y = func(x);
output[threadIdx] = y;
```

Thread Block N - 1
```
float x = input[threadIdx];
float y = func(x);
output[threadIdx] = y;
```

blockIdx == 0

blockIdx == 1

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2009
ECE 498AL, University of Illinois, Urbana-Champaign
CUDA Memory Model and Usage

• `cudaMalloc()`, `cudaFree()`

• `cudaMallocArray()`, `cudaMalloc2DArray()`, `cudaMalloc3DArray()`

• `cudaMemcpy()`

• `cudaMemcpyArray()`

• Host ↔ host
  Host ↔ device
  Device ↔ device

• Asynchronous transfers

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2009
ECE 498AL, University of Illinois, Urbana-Champaign
CUDA Software Development

CUDA Optimized Libraries: math.h, FFT, BLAS, ...

Integrated CPU + GPU C Source Code

NVIDIA C Compiler

NVIDIA Assembly for Computing (PTX)

CPU Host Code

CUDA Driver

Standard C Compiler

Profiler

GPU

CPU

© 2008 NVIDIA Corporation.
Compiling CUDA Code

C/C++ CUDA Application

NVCC

CPU Code

PTX Code

Virtual

PTX to Target Compiler

Physical

G80

Target code

...
CUDA Kernels and Threads

- Parallel portions of an application are executed on the device as **kernels**
  - One **kernel** is executed at a time
  - Many threads execute each **kernel**

- Differences between CUDA and CPU threads
  - CUDA threads are extremely lightweight
    - Very little creation overhead
    - Instant switching
  - CUDA uses 1000s of threads to achieve efficiency
    - Multi-core CPUs can use only a few

**Definitions**

- **Device** = GPU
- **Host** = CPU
- **Kernel** = function that runs on the device
Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code
  - Each thread has an ID that it uses to compute memory addresses and make control decisions

```plaintext
... float x = input[threadID];
float y = func(x);
output[threadID] = y;
...
```
Thread Batching

- Kernel launches a **grid of thread blocks**
  - Threads within a block cooperate via shared memory
  - Threads within a block can synchronize
  - Threads in different blocks cannot cooperate
- Allows programs to *transparencyly scale* to different GPUs
Transparent Scalability

- Hardware is free to schedule thread blocks on any processor
- A kernel scales across parallel multiprocessors
Execution Model

**Software**
- Thread
- Thread Block
- Grid

**Hardware**
- Thread Processor
- Multiprocessor
- Device

**Threads are executed by thread processors**
- Thread blocks are executed on multiprocessors
- Thread blocks do not migrate
- Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)

**A kernel is launched as a grid of thread blocks**
- Only one kernel can execute on a device at one time
CUDA Programming Model

- **Kernel**
  - GPU program that runs on a thread grid

- **Thread hierarchy**
  - Grid: a set of blocks
  - Block: a set of warps
  - Warp: a SIMD group of 32 threads
  - Grid size * block size = total # of threads
CUDA Memory Structure

- Memory hierarchy
  - PC memory: off-card
  - GPU global: off-chip / on-card
  - GPU shared/register/cache: on-chip
- The host can read/write global memory
- Each thread communicates using shared memory
Kernel Memory Access

- **Per-thread**
  - Registers
  - On-chip
  - Local Memory
  - Off-chip, uncached
  - cached on Fermi/Kepler

- **Per-block**
  - Blocks
  - On-chip, small
  - Fast

- **Per-device**
  - Global Memory
    - Off-chip, large
    - Uncached
    - Persistent across kernel launches
    - Kernel I/O
    - cached on Fermi/Kepler

© 2008 NVIDIA Corporation.
## Memory Architecture

<table>
<thead>
<tr>
<th>Memory</th>
<th>Location</th>
<th>Cached</th>
<th>Access</th>
<th>Scope</th>
<th>Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>On-chip</td>
<td>N/A</td>
<td>R/W</td>
<td>One thread</td>
<td>Thread</td>
</tr>
<tr>
<td>Local</td>
<td>Off-chip</td>
<td>No*</td>
<td>R/W</td>
<td>One thread</td>
<td>Thread</td>
</tr>
<tr>
<td>Shared</td>
<td>On-chip</td>
<td>N/A</td>
<td>R/W</td>
<td>All threads in a block</td>
<td>Block</td>
</tr>
<tr>
<td>Global</td>
<td>Off-chip</td>
<td>No*</td>
<td>R/W</td>
<td>All threads + host</td>
<td>Application</td>
</tr>
<tr>
<td>Constant</td>
<td>Off-chip</td>
<td>Yes</td>
<td>R</td>
<td>All threads + host</td>
<td>Application</td>
</tr>
<tr>
<td>Texture</td>
<td>Off-chip</td>
<td>Yes</td>
<td>R</td>
<td>All threads + host</td>
<td>Application</td>
</tr>
</tbody>
</table>

* cached on Fermi/Kepler
## Basic Limits (CUDA C Programming Guide, Appendix F.1)

<table>
<thead>
<tr>
<th>Technical Specifications</th>
<th>1.0</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.x</th>
<th>3.0</th>
<th>3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum dimensionality of grid of thread blocks</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum x-dimension of a grid of thread blocks</td>
<td>65535</td>
<td></td>
<td></td>
<td></td>
<td>2^{31}-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum y- or z-dimension of a grid of thread blocks</td>
<td></td>
<td>65535</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum dimensionality of thread block</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum x- or y-dimension of a block</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum z-dimension of a block</td>
<td></td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of threads per block</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warp size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of resident blocks per multiprocessor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Maximum number of resident warps per multiprocessor</td>
<td>24</td>
<td>32</td>
<td>48</td>
<td></td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of resident threads per multiprocessor</td>
<td>768</td>
<td>1024</td>
<td>1536</td>
<td></td>
<td>2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of 32-bit registers per multiprocessor</td>
<td>8 K</td>
<td>16 K</td>
<td>32 K</td>
<td></td>
<td>64 K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Basic Limits (CUDA C Programming Guide, Appendix F.1)

<table>
<thead>
<tr>
<th>Technical Specifications</th>
<th>1.0</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.x</th>
<th>3.0</th>
<th>3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of 32-bit registers per thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63</td>
<td>255</td>
</tr>
<tr>
<td>Maximum amount of shared memory per multiprocessor</td>
<td>16 KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>48 KB</td>
<td></td>
</tr>
<tr>
<td>Number of shared memory banks</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Amount of local memory per thread</td>
<td>16 KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>512 KB</td>
<td></td>
</tr>
<tr>
<td>Constant memory size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64 KB</td>
</tr>
<tr>
<td>Cache working set per multiprocessor for constant memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 KB</td>
</tr>
<tr>
<td>Cache working set per multiprocessor for texture memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Device dependent, between 6 KB and 8 KB</td>
</tr>
</tbody>
</table>
Thank you.