CUDA introduction

Felipe A. Cruz

Nagasaki Advanced Computing Center
Nagasaki University, Japan
## NVIDIA GPU Architecture

<table>
<thead>
<tr>
<th>GPU</th>
<th>G80</th>
<th>GT200</th>
<th>Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>681 million</td>
<td>1.4 billion</td>
<td>3.0 billion</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>128</td>
<td>240</td>
<td>512</td>
</tr>
<tr>
<td>Double Precision Floating Point Capability</td>
<td>None</td>
<td>30 FMA ops / clock</td>
<td>256 FMA ops / clock</td>
</tr>
<tr>
<td>Single Precision Floating Point Capability</td>
<td>128 MAD ops / clock</td>
<td>240 MAD ops / clock</td>
<td>512 FMA ops / clock</td>
</tr>
<tr>
<td>Special Function Units (SFUs) / SM</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Warp schedulers (per SM)</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Shared Memory (per SM)</td>
<td>16 KB</td>
<td>16 KB</td>
<td>Configurable 48 KB or 16 KB</td>
</tr>
<tr>
<td>L1 Cache (per SM)</td>
<td>None</td>
<td>None</td>
<td>Configurable 16 KB or 48 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>None</td>
<td>None</td>
<td>768 KB</td>
</tr>
<tr>
<td>ECC Memory Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concurrent Kernels</td>
<td>No</td>
<td>No</td>
<td>Up to 16</td>
</tr>
<tr>
<td>Load/Store Address Width</td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
</tbody>
</table>

Comparison of NVIDIA GPU generations. Current generation: GT200. Table from NVIDIA Fermi white-paper.
Strong points of CUDA

• Abstracting from the hardware

  Abstraction by the **CUDA API**. You don’t see every little aspect of the machine.

  Gives **flexibility to the vendor**. Change hardware but keep legacy code.

• **Forward compatible.**

• **Automatic Thread management** (can handle +100k threads)

  **Multithreading**: hides latency and helps maximize the GPU utilization.

  Transparent for the programmer (you don’t worry about this.)

  Limited **synchronization between threads** is provided.

  **Difficult to dead-lock.** (No message passing!)
Programmer effort

• Analyze algorithm for **exposing parallelism:**
  
  Block size
  
  Number of threads
  
  • Tool: pen and paper

Challenge: **Keep machine busy** (with limited resources)

  Global data set (Have efficient data transfers)

  Local data set (Limited on-chip memory)

  Register space (Limited on-chip memory)

  • Tool: Occupancy calculator
Outline

• Thread hierarchy.
• Memory hierarchy.
• Basic C extensions.
• GPU example.
Thread hierarchy

- Kernels are executed by thread.

  A kernel is a simple C program.

  Each thread has its own ID.

  Thousands of threads execute the same kernel.

- Threads are grouped into blocks.

  Threads in a block can synchronize execution.

- Blocks are grouped in a grid.

  Blocks are independent (Must be able to be executed in any order.)
Memory hierarchy

- Three **types** of memory in the graphic card:
  - Global memory: 4GB
  - Shared memory: 16 KB
  - Registers: 16 KB

**Latency:**
- Global memory: 400-600 cycles
- Shared memory: Fast
- Register: Fast

**Purpose:**
- Global memory: IO for grid
- Shared memory: thread collaboration
- Registers: thread space
Basic C extensions

Function modifiers

- __global__ : to be called by the host but executed by the GPU.
- __host__ : to be called and executed by the host.

Kernel launch parameters

- Block size: (x, y, z). \( x \times y \times z = \) Maximum of 768 threads total. (Hw dependent)
- Grid size: (x, y). Maximum of thousands of threads. (Hw dependent)

Variable modifiers

- __shared__ : variable in shared memory.
- __syncthreads() : sync of threads within a block.

Check CUDA programming guide for all the features!