SM – Streaming multi-processors with multiple processing cores
- Each SM contains 32 processing cores
- Execute in a Single Instruction Multiple Thread (SIMT) fashion
- Up to 16 SMs on a card for a maximum of 512 compute cores
CUDA Programming Model Review

- **Grid** is composed of blocks which are completely independent.
- **Block** is composed of threads which can communicate within their own block.
- 32 threads form a **warp**.
- Instructions are issued per warp.
- If an operand is not ready the warp will stall.
  - Context switch between warps when stalled.
  - Context switch must be very fast.

<table>
<thead>
<tr>
<th>Grid 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Block</td>
<td>Block</td>
<td>Block</td>
</tr>
<tr>
<td></td>
<td>(0, 0)</td>
<td>(1, 0)</td>
<td>(2, 0)</td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Block</td>
<td>Block</td>
</tr>
<tr>
<td></td>
<td>(0, 1)</td>
<td>(1, 1)</td>
<td>(2, 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block (1, 1)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, 0)</td>
<td>(1, 0)</td>
<td>(2, 0)</td>
<td>(3, 0)</td>
<td>(4, 0)</td>
</tr>
<tr>
<td>Thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, 1)</td>
<td>(1, 1)</td>
<td>(2, 1)</td>
<td>(3, 1)</td>
<td>(4, 1)</td>
</tr>
<tr>
<td>Thread</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, 2)</td>
<td>(1, 2)</td>
<td>(2, 2)</td>
<td>(3, 2)</td>
<td>(4, 2)</td>
</tr>
</tbody>
</table>
Fast Context Switching

- Registers and shared memory are allocated for a block as long as that block is active
  - Once a block is active it will stay active until all threads in that block have completed
  - Context switching is very fast because registers and shared memory do not need to be saved and restored
- **Goal: Have enough transactions in flight to saturate the memory bus**
  - Latency can be hidden by having more transactions in flight
  - Increase active threads or Instruction Level Parallelism (ILP)
  - Fermi can have up to 48 active warps per SM (1536 threads)
Maximizing Memory Throughput

- Increment of an array of 64M elements
  - Two accesses per thread (load then store)
  - The two accesses are dependent, so really 1 access per thread at a time
- Tesla C2050, ECC on, theoretical bandwidth: ~120 GB/s

Several independent smaller accesses have the same effect as one larger one.

For example:
Four 32-bit $\sim$ one 128-bit
Occupancy

- Occupancy = Active Warps / Maximum Active Warps
- Remember: resources are allocated for the entire block
  - Resources are finite
  - Utilizing too many resources per thread may limit the occupancy
- Potential occupancy limiters:
  - Register usage
  - Shared memory usage
  - Block size
Occupancy Limiters: Registers

- Register usage: compile with `--ptxas-options=-v`
- Fermi has 32K registers per SM

**Example 1**
- Kernel uses 20 registers per thread (+1 implicit)
- Active threads = 32K/21 = 1560 threads
  - > 1536 thus an occupancy of 1

**Example 2**
- Kernel uses 63 registers per thread (+1 implicit)
- Active threads = 32K/64 = 512 threads
- 512/1536 = .3333 occupancy

- Can control register usage with the nvcc flag: `--maxrregcount`
Occupancy Limiters: Shared Memory

- Shared memory usage: compile with --ptxas-options=-v
  - Reports shared memory per block
- Fermi has either 16K or 48K shared memory
- Example 1, 48K shared memory
  - Kernel uses 32 bytes of shared memory per thread
  - $48K/32 = 1536$ threads
  - occupancy=1
- Example 2, 16K shared memory
  - Kernel uses 32 bytes of shared memory per thread
  - $16K/32 = 512$ threads
  - occupancy=.3333
- Don’t use too much shared memory
- Choose L1/Shared config appropriately.
Occupancy Limiter: Block Size

- Each SM can have up to 8 active blocks
- A small block size will limit the total number of threads

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Active Threads</th>
<th>Occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>256</td>
<td>0.1666</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.3333</td>
</tr>
<tr>
<td>128</td>
<td>1024</td>
<td>0.6666</td>
</tr>
<tr>
<td>192</td>
<td>1536</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>2048 (1536)</td>
<td>1</td>
</tr>
</tbody>
</table>

- Avoid small block sizes, generally 128-256 threads is sufficient
What Occupancy Do I Need?

- Depends on your problem…
  - Many find 66% is enough to saturate the bandwidth

- Look at increasing occupancy only if the following are true!
  - The kernel is bandwidth bound
  - The achieved bandwidth is significantly less than peak

- Instruction Level Parallelism (ILP) can have a greater effect than increasing occupancy
  - Vasily Volkov’s GTC2010 talk “Better Performance at Lower Occupancy”
Cuda Occupancy Calculator

- A tool to help you investigate occupancy
- Demo: CUDA_Occupancy_calculator.xls
Compute Profiler 4.0

- A useful profiling tool which can help you investigate occupancy, throughput, and bandwidth.
- Measures actual occupancy and thus may detect problems that shouldn’t appear in theory
- Demo: `computeprof`
In order to achieve peak global memory bandwidth we need to have enough transactions in flight to hide latency.

We can increase the number of transactions by:
- Increasing occupancy
- Increasing instruction level parallelism

Occupancy can be limited by:
- Register usage
- Shared memory usage
- Block size

Use the cuda occupancy calculator and the visual profiler to investigate memory bandwidth/occupancy.
Questions?