Announcements:

- Grid size 20/30
- HW3 contest
- Grades
- HW2 answers
- Project!
GPU: Communication

What forms of communication exist at each scope?

<table>
<thead>
<tr>
<th>Simo</th>
<th>Shuffles / Sync + scatter</th>
</tr>
</thead>
<tbody>
<tr>
<td>WG</td>
<td>Atomic</td>
</tr>
<tr>
<td>Core</td>
<td>Atomic (but: no waiting)</td>
</tr>
</tbody>
</table>

Can we just do locking like we might do on a CPU?

| Partition 1 | Partition 2 |

- Independent, in progress
- Also good for NUMA
GPU Programming Model: Commentary

- “Vector” / “Warp” / “Wavefront”
  - Important hardware granularity
  - Poorly/very implicitly represented
- What is the impact of reconvergence?
What limits the amount of concurrency exposed to GPU hardware?

- num of scheduling slots
- size of the register file (variable size per tile)
- size of scratchpad

- L1P
- Group size

1000s
Memory Systems: Recap

Diagram showing the flow between the processor and memory, with signals such as D0..15, A0..15, R/W, and CLK, and annotations like "bunks".
Parallel Memories

Problem: Memory chips have only one data bus. So how can multiple threads read multiple data items from memory simultaneously?

- Split a wide bus (off-chip/global)
- Multiple multiplexed memory (hts, on-chip)

Where does banking show up?

- scratchpad
- register file
Memory Banking

Fill in the access pattern:

<table>
<thead>
<tr>
<th>Bank</th>
<th>2</th>
<th>6</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>22</th>
<th>⋯</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>⋯</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address
Memory Banking

Fill in the access pattern:

Bank

local_variable[lid(0)]
Memory Banking

Fill in the access pattern:

```
3  7 11 15 19 23 \cdots
```

Bank

```
2  6 10 14 18 22 \cdots
```

```
1  5  9 13 17 21 \cdots
```

```
0  4  8 12 16 20 \cdots
```

local_variable[BANK_COUNT*lid(0)]
Memory Banking

Fill in the access pattern:

Bank

| 2 | 6 | 10 | 14 | 18 | 22 | ... |

| 1 | 5 | 9 | 13 | 17 | 21 | ... |

| 0 | 4 | 8 | 12 | 16 | 20 | ... |

Thread

Address

local_variable[(BANK_COUNT+1)*lid(0)]
Memory Banking

Fill in the access pattern:

Bank

0 4 8 12 16 20 21 22 18 14 10 6 2 3

Address

local_variable[ODD_NUMBER*lid(0)]
Memory Banking

Fill in the access pattern:

Bank

Address

local_variable[2*lid(0)]
Memory Banking

Fill in the access pattern:

Bank:

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>21</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>22</td>
<td>...</td>
</tr>
</tbody>
</table>

Address

local_variable[f(gid(0))]

Looks bad
But: important special case with hw help
Memory Banking: Observations

- Factors of two in the stride: generally bad
- In a conflict-heavy access pattern, padding can help
  - Usually not a problem since scratchpad is transient by definition
- Word size (bank offset) may be adjustable (Nvidia)

Given that unit strides are beneficial on global memory access, how do you realize a transpose?

```
load from global w/ unit stride
large stride access w/ scratchy
shift to y w/ unit stride
```
GPU Global Memory System
GPU Global Memory Channel Map: Example

Byte address decomposition:

<table>
<thead>
<tr>
<th>Address</th>
<th>Bank</th>
<th>Chnl</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>?</td>
<td>11</td>
<td>108</td>
</tr>
</tbody>
</table>

Implications:

- Transfers between compute unit and channel have granularity
  - Reasonable guess: warp/wavefront size \( \times 32\text{bits} \)
  - Should strive for good utilization (‘Coalescing’)
- Channel count often not a power of two \( \rightarrow \) complex mapping
  - Channel conflicts possible
- Also banked
  - Bank conflicts also possible
GPU Global Memory: Performance Observations

Key quantities to observe for GPU global memory access:

- number of channels hit / lane stride
- group stride
- utilization

Are there any guaranteed-good memory access patterns?

- units strides

- Need to consider access pattern across entire device
- GPU caches: Use for spatial, not for temporal locality
- Switch available: L1/Scratchpad partitioning
  - Settable on a per-kernel basis
- Since GPUs have meaningful caches at this point:
  Be aware of cache annotations (see later)
Host-Device Concurrency

- Host and Device run asynchronously
- Host submits to queue:
  - Computations
  - Memory Transfers
  - Sync primitives
  - ...
- Host can wait for:
  - *drained* queue
  - Individual “events”
- Profiling
Timing GPU Work

How do you find the execution time of a GPU kernel?

submit work
drain queue
start timer
submit work
drain queue
stop timer

How do you do this asynchronously?

workers
Host-Device Data Exchange

- Sad fact: Must get data onto device to compute
  - Transfers can be a bottleneck
  - If possible, overlap with computation
  - Pageable memory incurs difficulty in GPU-host transfers, often entails (another!) CPU side copy
  - “Pinned memory”: un pageable, avoids copy
    - Various system-defined ways of allocating pinned memory
- “Unified memory”:
  - GPU directly accesses host memory
  - “Fine grain”: Byte-for-byte coherent
  - “Coarse grain”: Per-buffer fences
Performance: Ballard Numbers?

<table>
<thead>
<tr>
<th>Bandwidth host/device:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bandwidth on device:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flop throughput?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel launch overhead?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>