Announcements:
- Project submission logistics
- CPU-focused HW3: tonight/tomorrow
“CPU-style” Cores

- Fetch/Decode
- ALU (Execute)
- Execution Context
- Out-of-order control logic
- Fancy branch predictor
- Memory pre-fetcher
- Data cache (A big one)

[Fatahalian ‘08]
Slimming down

Idea #1:
Remove components that help a single instruction stream run fast

[Fatahalian ‘08]
More Space: Double the Number of Cores

[Fatahalian ‘08]
Even more

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Latency Hiding

- Latency (mem, pipe) hurts non-OOO cores
- Do something while waiting

What is the unit in which work gets scheduled on a GPU?

Before:
- Fetch/Decode
- Register File
- Scratchpad/L1

After:

more state space

How can we keep busy?

- SW
- LP

Change in architectural picture?
GPUs: Core Architecture Ideas

Three core ideas:

- Simpler / many of cores
- SIMD
- latency hiding through concurrency
\( \text{cor index} \)

\( \text{vector index} \)

\( (\text{cor index}, \text{vector line index}) \)

\( x = \text{vec} \% 17 \)

\( y = \text{vec} // 17 \)
Wrangling the Grid

- `grid_id`: 1
- `grid_dim0`: 1
- `thread_id`: `axورzoomxyz`

- `get_local_id(axis)`? `/size(axis)`?
- `get_group_id(axis)`? `/num_groups(axis)`?
- `get_global_id(axis)`? `/size(axis)`?

axis=0,1,2,...
Demo CL code

Demo: machabstr/Hello GPU
'SIMT' and Branches

ALU 1  ALU 2  ...  ALU 8  ...

Time (clocks)

if \( x > 0 \) {
  y = \text{pow}(x, \text{exp});
  y *= Ks;
  \text{refl} = y + Ka;
}
else {
  x = 0;
  \text{refl} = Ka;
}

<resume unconditional shader code>

[Fatahalian '08]
GPU Abstraction: Core Model Ideas

How do these aspects show up in the model?

- View concrete counts as an implementation detail
  - SIMD lane
  - Core
  - Scheduling slot
- Program as if there are infinitely many of them
- Hardware division is expensive
  Make nD grids part of the model to avoid it
- Design the model to expose extremely fine-grain concurrency (e.g. between loop iterations!)
- Draw from the same pool of concurrency to hide latency
## GPU Program 'Scopes'

<table>
<thead>
<tr>
<th>Hardware</th>
<th>CL adjective</th>
<th>OpenCL</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD lane</td>
<td>private</td>
<td>Work Item</td>
<td>Thread</td>
</tr>
<tr>
<td>SIMD Vector</td>
<td>—</td>
<td>Subgroup</td>
<td>Warp</td>
</tr>
<tr>
<td>Core</td>
<td>local</td>
<td>Workgroup</td>
<td>Thread Block</td>
</tr>
<tr>
<td>Processor</td>
<td>global</td>
<td>NDRRange</td>
<td>Grid</td>
</tr>
</tbody>
</table>
GPU: Communication

What forms of communication exist at each scope?

- 3D games, Vector shuffles
  - Scratchpad + barrier, atomic + mem fences
  - across 3D + GPUs; atomics + mem fences

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

no; indep. fw progress required
GPU Programming Model: Commentary

- “Vector” / “Warp” / “Wavefront”
  - Important hardware granularity
  - Poorly/very implicitly represented
- What is the impact of reconvergence?