Simultaneous Multikernel GPU
(extracted from paper presentation in HPCA-15)

Multitasking GPU: Hyper-Q

<table>
<thead>
<tr>
<th>Single HW Queue</th>
<th>Hyper-Q</th>
<th>Hyper-Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_0$ 128TBs</td>
<td>$K_0$ 128TBs</td>
<td>$K_0$ 256TBs</td>
</tr>
<tr>
<td>$K_1$ 64TBs</td>
<td>$K_1$ 64TBs</td>
<td>$K_1$ 64TBs</td>
</tr>
<tr>
<td>$K_2$ 128TBs</td>
<td>$K_2$ 128TBs</td>
<td>$K_2$ 256TBs</td>
</tr>
</tbody>
</table>

HW queue limits parallelism

No control over sharing
Multitasking GPU: Previous SW/HW Designs

- **No preemption**
  - Elastic Kernel
  - [ASPLOS ’13]

- **Sharing granularity is limited**
  - No Sharing within SMs
  - Spatial Partitioning
  - [ISCA ’14]

**Motivation**

- **Fine-grain sharing is good for utilization and performance**
  - Better resource utilization when sharing
  - Better performance when sharing
SMK: Partial Context Switch

**Full Context Switch**
- Running Kernels: \( K_0, K_1 \)
- Preemption blocks execution
- Swap out a SM

**Partial Context Switch**
- Running Kernels: \( K_0, K_1 \)
- Preemption overlaps with execution
- Enabling partial swapping

Preempted Queue

SMK: Resource Allocation

**On-Demand Allocation**
- Running Kernels: \( K_0, K_1 \)
- The kernel with least resources
- Allocate on-the-fly
- Fragmentation

**Resource Partitioning**
- Running Kernels: \( K_0, K_1 \)
- Allocate based on the partition

Preempted Queue
SMK: Fair Warp Scheduling

Baseline Scheduling

Fair Scheduling

Warp Scheduler

Warp Pool

Issue

Pick a warp

Not aware of multikernel

No performance fairness

Estimate isolated performance

Reset on every epoch

K₀

K₁

Quota

SM₀

SM₁

Similar slowdown for kernels

Prevent over-issuing

Barrier synchronization in GPU

• Barriers are only provided for threads within a TB.
Barrier synchronization in GPU

- Barriers are only provided for threads within a TB.

- No global barriers are provided in CUDA
- Need to use atomic global memory operations + busy-waiting to implement global barriers – but may deadlock.

Preemption allows barrier synchronization in GPU

- No global (inter TB) barriers – will deadlock in the absence of preemption
Preemption allows wait/signal between TBs

Pending queue

| TB0 | TB9 |
| TB4 | TB5 |
| TB6 | TB7 |

Wait list

Ready list

Preemption queue

Synchronizing the CPU with the GPU??

Nvidia GeForce card with multiple GPUs

Trinity’s built-in AMD Radeon HD 7660D GPU includes 384 graphics cores (which AMD calls stream processors) running at 800MHz. It’s worth noting that this number of stream cores gives the 7660D higher potential performance than AMD’s entry-level discrete GPUs, such as the Radeon HD 7450, which only have 160 graphics cores. Given the emphasis on graphics, it’s no surprise that the GPU takes up over half the Trinity die space.