Share-a-GPU
Providing Simple and Effective Time-Sharing on GPUs

Shaleen Garg, Kishore Kothapalli, Suresh Purini

Computer Systems Group
Needs?

Expensive Hardware
- Users Contend for Same Resource
- Exorbitant Wait times
- Native GPUs follow leftover policy

Cloud Service Providers
- Usual to Overbook their hardware
- Rely on user non-consumption
- Fails when faces high traffic

E-Waste
- Hardware getting old rapidly
- Simulates a bigger GPU using Software
- Not explored much

Energy Impact
- No need for more hardware
- Less energy consumption
- Lesser heat generation
Literature Survey

- **Wu et al. FLEP (ASPLOS ‘17) [Software]**
  - MicroKernels for weighted Round-Robin Scheduling
  - Assumes that GPU Memory accommodates all programs
  - Experiments show at most three loads running concurrently

- **Xu et al. Warped-Slicer (ISCA ‘16) [Simulator]**
  - Pairs programs such that they use all the resources on SMs
  - Needs prior profiling to find suitable coexisting Kernels

- **Anguilera et al. Fair Share (ICCD ‘14) [Simulator]**
  - Proposes allocation policy based on Fairness to all programs
  - Uses Spatial multitasking instead of cooperative multitasking
Overview

- Introduction
- Our Technique
- Experiments
- Conclusion
Overview

- Introduction
- Our Technique
- Experiments
- Conclusion
The best ideas are often also the simplest
Some Definitions

- **Compute Kernel** - Upto 3-D grid of GPU computations
- **Grid** - Consists of many Blocks (CTAs)
- **Compute Block** - Independent Unit of computations
- **Index** - Identity of each block (3 tuple of integers)
- **Timeslice** - Time of exclusive access to the GPU(s)
Stages of Preemption

Pause

Pause the Kernel after its time slice for context switching

Save State

Snapshot of the current program state

Resume

Start from where we stopped last
The Changeover

Native GPU (FIFO)

Scheduler (Round-Robin)
Memory Management

- Realistic Approach
- Double Buffering
- Memory transfers (PCI) concurrent to compute
- Guarantees $\frac{1}{2}$ GPU memory to each kernel
Program States

Wait
Prepare State

State Restored

Ready to Launch

State

Finish

More mKernels

No More mKernels

Save State

mKernel finished

Running

Launch mKernel
Scheduler - The Middleman

Scheduler

Kernel(s)

GPU(s)

Enqueue

When Next in Line

When GPU Free

GPU Free

Required Time Slice

Prepare State

Ready to Launch

Launch mKernel

Yielded GPU

Yielded GPU Mem

Wait

Memcpy

Done Memcpy

Ready To Launch

Running

Save State

Memcpy

Done Memcpy

Finish

GPU Mem loaded

GPU Busy

GPU Mem Free
Multi-GPU Support

- Multi-GPU installations common
- Simulates an aggregate GPU from many small GPUs
- Intra-Kernel Parallelism
- Workload divided automatically
The best systems do not bother their users
- Assisted Cooperation
- Modified API
- Usual Execution
- Transparent Working
Overview

- Introduction
- Our Technique
- Experiments
- Conclusion
Overview

- Introduction
- Our Technique
- Experiments
- Conclusion
## Experimental Programs

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Input</th>
<th>Memory footprint (MiB)</th>
<th>Native Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Memory</td>
<td>1321</td>
<td>0.4</td>
</tr>
<tr>
<td>Gaussian</td>
<td>Compute</td>
<td>883</td>
<td>327.45</td>
</tr>
<tr>
<td>Kmeans</td>
<td>Memory</td>
<td>1006</td>
<td>0.07</td>
</tr>
<tr>
<td>matMul</td>
<td>Compute</td>
<td>5424</td>
<td>54.19</td>
</tr>
<tr>
<td>matTrans</td>
<td>Memory</td>
<td>5433</td>
<td>0.17</td>
</tr>
<tr>
<td>dxtc</td>
<td>Compute</td>
<td>2280</td>
<td>20.78</td>
</tr>
<tr>
<td>StringSort</td>
<td>Memory</td>
<td>3072</td>
<td>0.203</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td><strong>19420</strong></td>
<td></td>
</tr>
</tbody>
</table>
Single Program on One GPU

- Average slowdown is 4 - 5 %
- Overheads decrease with increase in timeslice
- Overheads ~0.1 ms
Single Program on Two GPUs

- Average speedup $\sim 1.65 \times$
- No additional user effort
All Programs on One GPU

- The real test of Fire
- First to show this!
- Overheads as anticipated
All Programs on Two GPUs

- Another test of Fire
- First to show this!
- Automatic speedup of $\sim 1.5 \times$
Future Work

▷ Extend to a cluster setting
▷ Support heterogeneous multi-GPUs
▷ Extend it to commonly used libraries
Conclusion

✓ Capable of things never done before
✓ Provides a deployable solution
✓ Transparent & Easy to use
✓ Flexible scheduling algorithms
Thanks!

Any Questions?

You can contact us at:
shaleen.garg@research.iiit.ac.in
kkishore@iiit.ac.in
suresh.purini@iiit.ac.in