LoGA: Low-Overhead GPU Accounting Using Events

Jens Kehne, Stanislav Spassov, Marius Hillenbrand, Marc Rittinghaus, Frank Bellosa
10th ACM International Systems and Storage Conference
GPU Sharing

- GPUs increasingly popular in computing
- Not every application saturates a GPU

- Move GPUs to the cloud
  - Sharing increases cost-efficiency

- Problem: Need fairness
- Software scheduling is inefficient
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON (University of Rochester, ASPLOS '14)

- Applies fair queuing to GPUs
NEON: Accounting Problem

- NEON’s accounting disables GPU access

➔ High accounting overhead if application does not saturate the GPU
Idea

- GPUs have lots of status registers for the device driver
  - Leak information about GPU’s internal state
  - Reading has no effect on GPU driver or running application

- Idea:
  - Poll status registers
  - Infer GPU-internal context switches
Design

- App
- Scheduler
- Submit
- Query
- Account thread
- Poll
- STOP

Flow:
- App → Submit
- Submit → App
- App → STOP
- STOP → Scheduler
- Scheduler → Query
- Query → Accounting thread
- Accounting thread → Poll
- Poll → Accounting thread
- Accounting thread → STOP
Accounting Thread

Accounting thread

Poll

Status reg

App 1

App 2
Accounting Thread

Accounting thread

Poll

Status reg

App 1

App 2
Accounting Thread

- Poll frequency must be faster than kernel length
- High CPU load ➞ Poll periodically
Scheduling

- Scheduling thread queries accounting thread
- Updates fair queuing counters
- Stops applications if necessary

- Different metric than NEON
  - NEON: Average kernel length
  - LoGA: Total GPU time consumed
Benchmark Scenario

- Accounting overhead
- Accuracy of accounting (→ scheduling quality)

- Competing workload: Throttle
  - Creates well-defined GPU load
Accounting Overhead (10% load)

10 instances of throttle, 1% load each

Normalized against execution time w/o accounting

Scheduling disabled
Accounting Overhead (90% load)

- 10 instances of throttle, 9% load each
- Normalized against execution time w/o accounting
- Scheduling disabled

- Normalized runtime
- NEON vs LoGA

- Bar chart showing normalized runtime comparison between NEON and LoGA for various benchmarks.

- The chart includes different load configurations and demonstrates the overhead compared to execution time without accounting.

- The y-axis represents normalized runtime, ranging from 0.7 to 1.6, with specific tick marks at 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5, and 1.6.

- The x-axis lists various benchmarks, including 'backprop', 'bis', 'b-tree', 'cmd', 'dwt2d', 'gaussian', 'heartwall', 'hotspot', 'hotspot3d', 'huffman', 'hybridsort', 'kmeans', 'lavaMD', 'leukocyte', 'lud', 'mummergpu', 'myocyte', 'mn', 'nw', 'particlefilter', 'pathfinder', 'rdrad_v1', 'rdrad_v2', 'streamcluster', and 'Geomean'.
Fairness

- Goal: LoGA should not reduce fairness

- Problem: Which application runtime is fair?

- Measure total GPU time for each application

- Calculate optimal scheduling
  - Next slide: Speedup over fair schedule
Fairness: Results (4x throttle, 20% load each)
Conclusion

- Sharing beneficial if applications do not saturate the GPU
- Scheduler interference reduces sharing

- LoGA accounts GPU usage without overhead
  - Poll GPU status registers, detect context switches
  - Time between context switches as input for fair queuing
Finding Registers

- Envytools project has documented some registers
  - Unfortunately, far from complete

- Trial and error:
  - Run workload with known behavior
  - Dump register values
  - See which registers correlate with workload behavior

- Two registers identified:
  - ID of currently running GPU context
  - Activity status of entire GPU