S7105 - ADAS/AD CHALLENGES: GPU SCHEDULING & SYNCHRONIZATION

Venugopala Madumbu, NVIDIA
GTC 2017 - 210D
ADVANCED DRIVING ASSIST SYSTEMS (ADAS) & AUTONOMOUS DRIVING (AD)
High Compute Workloads Mapped to GPU
ADAS/AD
Requirements & Challenges

Real-Time Behavior
• Determinism
• Freedom from Interference
• Priority of Functionalities

Performance
• Maximum Throughput
• Minimal Latency

Multi-Core CPU
GPU/DSP/HWA
ADAS/AD WORKLOADS

Challenges Illustrated

Scenario#1 - Standalone Exec

GL Workload

X msec

Scenario#2 - Standalone Exec

CUDA Workload

Y msec

Scenario#3 - Concurrent Exec

GL Workload

CUDA Workload

> (X+Y) msec

Time Shared GPU Execution

If so, How to

- Achieve determinism
- Achieve Freedom from interference
- Prioritize one Workload over other

While also having

- maximum throughput
- minimum latency
GPU IN TEGRA
High Level Tegra SoC Block Diagram

- CPU submits job/work to GPU
- GPU runs asynchronously to CPU
- GPU has its own hardware scheduler (Host)
  - It switches between workloads without CPU involvement
GPU SCHEDULING

Concepts

Channel - independent stream of work on the GPU

Command Push Buffer - Command buffer written by Software and read by Hardware

Channel Switching - Save/restore GPU state on a channel switch

Semaphores/SyncPoints - Synchronization mechanism for events within the GPU

Time Slice - How long a GPU executes commands of a channel before a channel switch

Run-list - An ordered list of channels that SW wants the GPU to execute
Channel switching occurs when any ONE of the following happens:

- Time slice expires
- Engine runs out of work (no more commands)
- Blocked on a semaphore

Channel Switch time = Drain Time + Save/Restore time

Preemption can reduce Channel Switch times drastically
GPU SCHEDULING

Preemption
GPU SCHEDULING

Channel Switching with Time Slice Scenarios

1. **Channel finishes before time slice expires**
   - Context switch to next channel

2. **Channel preemption**
   - Stop all commands in pipeline
   - Wait for engines to idle
   - Higher Context Switch time

3. **Channel Reset**
   - Engine could not idle and context could not save before channel switch timeout
   - Callback to notify kernel of channel reset event
CHALLENGE REVISTED

How can we achieve both?

Real-Time behavior:
• Determinism
• Freedom from Interference
• Priority of Functionalities

Performance:
• Maximum Throughput
• Minimal Latency
1. User Driver Level (GPU Synchronization Approach)
   - Syncpoints/Semaphores for Synchronization
     - Through EglStreams, EGLSync etc

2. Kernel Driver Level (GPU Priority Scheduling Approach)
   - Run-List Engineering
     - How long channel runs
     - Order of Channel execution
GPU SYNCHRONIZATION APPROACH

No Synchronization Case
GPU SYNCHRONIZATION APPROACH

Synchronization on CPU: Not good for GPU

- **Kernel launch**
- **GPU Semaphore**

Diagram shows:
- Priority GPU Task
- GPU Task
- CPU Task

Timeline from 0 to 35 msec.
GPU SYNCHRONIZATION APPROACH

Synchronization on GPU: No Context Switches

- Determinism
- Freedom from Interference
- Priority of Functionalities
# GPU PRIORITY SCHEDULING APPROACH

Hypothetical Example

<table>
<thead>
<tr>
<th>TASK</th>
<th>PRIORITY</th>
<th>FPS</th>
<th>WORST CASE EXECUTION TIME (WCET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>High</td>
<td>60</td>
<td>9ms</td>
</tr>
<tr>
<td>M1</td>
<td>Medium</td>
<td>30</td>
<td>4ms</td>
</tr>
<tr>
<td>M2</td>
<td>Medium</td>
<td>30</td>
<td>4ms</td>
</tr>
<tr>
<td>L1</td>
<td>Low/Best Effort</td>
<td>30</td>
<td>10ms</td>
</tr>
</tbody>
</table>
GPU PRIORITY SCHEDULING APPROACH
Engineered Run-list and Time Slice Ensuring FPS and Latency

- H1 (Max Exec Time = 9 ms)
  Time slice = 9 ms
- M1 (Max Exec Time = 4 ms)
  Time slice = 3 ms
- M2 (Max Exec Time = 4 ms)
  Time slice = 3 ms
- L1 (Max Exec Time = 10 ms)
  Time slice = 1 ms

Ensured not >16 ms for 60fps operation
GPU PRIORITY SCHEDULING APPROACH
Reduce Latency for GPU Work Completion

- Ensure timeslice is long enough to complete work
- Ensure work is continually submitted and also well ahead in time
  - To Avoid
    - GPU idle time
    - Unnecessary context switches
GPU SCHEDULING
Best Practices to Keep GPU Busy

- Submit work in advance
  - So the GPU has some work to execute at any point of time
- Try to reduce/eliminate work dependencies
- Have contingency plan for work overload
  - If feedback shows over budget, submit work few frames ahead and spread
- Plan for worst case scenario
  - Deal with GPU reset case esp for the Low priority cases
    - GL Robustness Extensions
CONCLUSION

GPU Synchronization & Scheduling Approaches

Real-Time behavior:
• Determinism
• Freedom from Interference
• Priority of Functionalities

Performance:
• Maximum Throughput
• Minimal Latency
ACKNOWLEDGEMENTS

• Scott Whitman, NVIDIA
• Vladislav Buzov, NVIDIA
• Amit Rao, NVIDIA
• Yogesh Kini, NVIDIA
Q&A
THANK YOU