CAWS: Criticality-Aware Warp Scheduler for GPGPU Workloads

Shin-Ying Lee and Carole-Jean Wu
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Graphics Processing Units (GPUs)

- In addition to render video frames, modern GPUs are also deployed to process parallel workloads.
- The benefit of using GPUs to perform general computation is to hide operation latency
  - *Massive multi-threading*
  - *Fast context-switching*
Unified GPU Architecture (Computation Path)

- The computation and graphics unified GPU architecture to support general computation
Research Questions

- Is the current GPU design good enough to hide execution latency?
- What kind of execution latency is hidden under the GPU scheduler?
- How do we propose a better scheduling policy to improve GPU’s latency hiding ability?
Outline

- GPU Latency Characterization
- Warp/Wavefront Criticality
- Criticality-Aware Warp Scheduling (CAWS)
- Discussion and Conclusion
Root-cause of Warp Stall

- Pipeline hazards
  - Data hazard
  - Structural hazard
  - Control hazard
- Memory access latency
- Instruction fetch policy
- Synchronization barrier
- Warp scheduling policy
Latency Characterization Algorithm

\[ \text{probe}(w) \]

\[ w.\text{Scheduling} + = \text{CurTime} - w.\text{PrevTime} \]

\[ w.\text{PrevTime} = \text{CurTime} \]

\[ \text{if}(\text{Sync}) \quad w.\text{Sync}++ \]

\[ \text{else if}(\text{EmptyInstBuffer}) \quad w.\text{Fetch}++ \]

\[ \text{else if}(\text{Branch Taken}) \quad w.\text{CtrlHazard}++ \]

\[ \text{else if}(\text{DataDependency}) \]

\[ \quad \text{if}(\text{OldDataCacheMiss}) \quad w.\text{DataCacheMiss}++ \]

\[ \quad \text{else} \quad w.\text{DataHazard}++ \]

\[ \text{else if}(\text{FU unavailable}) \quad w.\text{StructuralHazard} \]

\[ \text{else} \quad w.\text{Exec}++ \]
Warp/Wavefront Scheduler

1. Select a warp according to the scheduling policy at every cycle
2. Probe and record the selected warp’s current status
3. Iteratively till find a ready warp

```plaintext
while(NotVisitedAllWarps)
    w = FindNextWarp()
    probe(w)
    if(w is a ready warp)
        issue(w)
        break
```
Latency Characterization Results
Outline

- GPU Latency Characterization
- Warp/Wavefront Criticality
- Criticality-Aware Warp Scheduling (CAWS)
- Discussion and Conclusion
Warp/Wavefront Criticality

- Slow warps/wavefronts take much longer time to finish their designated jobs.
- Fast warps are blocked at an implicit/explicit synchronization barrier to wait for the slow warps/wavefronts.
Factor Causing Warp Criticality for *bfs*

workload imbalance caused by branches

![Graph showing factors causing warp criticality for bfs](image)
Outline

- GPU Latency Characterization
- Warp/Wavefront Criticality
- Criticality-Aware Warp Scheduling (CAWS)
- Discussion and Conclusion
Criticality-Aware Warp Scheduling

• To equalize the execution time disparity, the Criticality-Aware Warp Scheduling (CAWS) algorithm prioritizes and schedule warps by warps’ weight.

• Assign warps different weight based on their criticality.

• Slower warps receive more time slots to run advance.
Types of CAWS Policies

- **CAWS-Blk**
  - Prioritizing warps within a thread block
  - To equalize execution time disparity within a thread block

- **CAWS-SM**
  - Prioritizing warps within as SM
  - To equalize execution time disparity within an SM

- **CAWS-Avg**
  - Identifying the critical warps based on the average execution time across an SM
Experimental Environment

- GPU simulation infrastructure
  - GPGPU-sim v3.2.0*
  - nVIDIA nvcc toolchain v3.2
  - nVIDIA GTX480 architecture

- Benchmarks
  - Imbalance workloads: bfs
  - Small parallel regions: b+tree
  - I-cache intensive: tpacf
  - D-cache intensive: kmeans

CAWS Speedup with Oracle Knowledge


Latency Characterization of \textit{bfs}

Exe Time Normalized to the Critical

War of Basline RR Policy


MSHR  Others  Scheduling  Execution

RR  CAWS
CAWS with Criticality Prediction on \textit{bfs}
Outline

• GPU Latency Characterization
• Warp/Wavefront Criticality
• Criticality-Aware Warp Scheduling (CAWS)
• Discussion and Conclusion
Research Questions

- Is the current GPU design good enough to hide execution latency?
- What kind of execution latency is hidden under the GPU scheduler?
- How do we propose a better scheduling policy to improve GPU’s latency hiding ability?
Conclusion

- This is the first latency characterization algorithm for GPUs that enables a deep understanding of how well the latency hiding ability is for modern GPU schedulers.
- We present the GPGPU workloads’ results to indicate places for performance improvement.
- We design a family of CAWS policies to improve GPGPU workload performance by equalizing warp execution time.
- The CAWS policies can potentially achieve 17% of performance improvement on average.
Thank You!

CAWS: Criticality-Aware Warp Scheduler for GPGPU Workloads
Shin-Ying Lee and Carole-Jean Wu
Arizona State University
BACKUP
bfs Algorithm

while(not visited all node)
    kernel_1
        travers all nodes’ children in current depth
        /* an implicit barrier here */

    kernel_2
        go to next depth
        /* an implicit barrier here */
CAWS Implementation

- Implemented by count-down counters
  - Every warp has its own priority counter.
  - A priority counter’s initial value is corresponding the warp’s priority/criticality
  - The counter value decrements cycle by cycle.
  - Scheduler picks up the warp having the lowest counter value to be issued.
  - Once a warp is issued, its counter value is reset to the initial value.
Criticality Inversion

- A fast warp becomes a new critical warp due to starving.
- It may limit the overall speedup or make performance even worse.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Speedup</th>
<th>Criticality Inversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAWS-blk</td>
<td>1.16</td>
<td>8.89%</td>
</tr>
<tr>
<td>CAWS-SM</td>
<td>1.18</td>
<td>2.22%</td>
</tr>
<tr>
<td>CAWS-Avg</td>
<td>1.21</td>
<td>0%</td>
</tr>
</tbody>
</table>
## Comparison of CAWS Policies

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CAWS-Blk</strong></td>
<td>To quickly finish a thread block</td>
<td>Useless for thread blocks having no critical warp</td>
</tr>
<tr>
<td><strong>CAWS-SM</strong></td>
<td>Useful for cases such that critical warps mapped to the same thread block</td>
<td>Some thread blocks might get starvation</td>
</tr>
<tr>
<td><strong>CAWS-Avg</strong></td>
<td>Criticality inversion avoidance</td>
<td>More complicated to implement</td>
</tr>
</tbody>
</table>
Latency Characterization Comparison

Latency Breakdown Normalized to Baseline RR

- Scheduling
- Execution
- Sync.
- Inst. Fetch
- MSHR
- Data$
- Ctrl. Hzd.
- Strcl. Hzd.
- Data Hzrd.