GPU Warp Scheduling Implementation Report

Task 1: GTO Implementation

Algorithm Description

The GTO scheduler implements a greedy approach that prioritizes keeping the same warp scheduled until it encounters a long-latency event (such as a memory access). When switching warps, it selects the oldest warp based on dispatch timestamps.

Key Implementation Details:

- Added dispatch_time field to warp_s struct to track when each warp was dispatched.
- Maintained c_last_scheduled_warp pointer in core to track the previously scheduled warp.
- Greedy behavior: If the last scheduled warp is still in the dispatch queue, schedule it again.
- Oldest selection: When switching, find the warp with the minimum dispatch_time.

Performance Analysis

Benchmark	RR Stall Cycles	GTO Stall Cycles	Improvement
lavaMD_5	28,827	71,109	-147%
nn_256k	663,693	576,007	+13%
backprop_8192	534,051	492,417	+8%
crystal_q12	3,157,674	2,855,964	+10%
hotspot_r512h2i2	1,341,698	1,459,497	-9%

Analysis: GTO shows mixed results. While it improves performance for some benchmarks (nn_256k, backprop_8192, crystal_q12) by reducing stall cycles through better cache locality, it performs worse on lavaMD_5 and hotspot_r512h2i2. This suggests that the greedy approach benefits workloads with good temporal locality but can hurt performance for workloads with poor locality or high memory contention.

Task 2: CCWS Implementation

Algorithm Description

CCWS dynamically reduces the number of scheduled warps to enhance cache locality through a sophisticated feedback mechanism:

VTA (Victim Tag Array) Management:

- Each warp maintains a VTA to track evicted cache lines.
- On L1 eviction (L2 hit or memory response), evicted tags are inserted into the warp's VTA.
- VTA uses LRU replacement with 8-way associativity.

LLS (Lost Locality Score) Calculation:

- On L1 miss, check if the tag exists in the warp's VTA.
- If VTA hit: LLS = (VTA_Hits × K_Throttle × Cum_LLS_Cutoff) / Num_Instr
- LLS scores decay by 1 each cycle (minimum = Base_Score = 100).

Dynamic Throttling:

- Calculate cumulative cutoff: Num_Active_Warps × Base_Locality_Score.
- Sort warps by LLS score (descending).
- Select warps whose cumulative scores reach the cutoff.
- Use Round Robin scheduling within the selected set.

Benchmark	RR Misses/1K	GTO Misses/1K	CCWS Misses/1K	CCWS vs RR
lavaMD_5	9.99	3.01	9.92	-0.7%
nn_256k	93.71	91.13	93.71	0%
backprop_8192	36.09	37.27	36.09	0%
crystal_q12	75.47	75.47	75.47	0%
hotspot_r512h2i2	180.92	182.05	180.92	0%

Performance Analysis: CCWS shows minimal impact on miss rates compared to Round Robin, with most benchmarks showing identical performance. This suggests that the dynamic throttling mechanism is not significantly reducing cache conflicts for these workloads. The algorithm may be more effective for workloads with higher cache pressure or different access patterns.

Observations

Workload Sensitivity

- Memory-Bounded Workloads: Benchmarks like nn_256k and hotspot_r512h2i2 show high miss rates, indicating memory-bounded behavior. GTO provides modest improvements for nn_256k but hurts hotspot_r512h2i2.
- Compute-Bounded Workloads: lavaMD_5 shows low miss rates (9.99/1K), suggesting compute-bounded behavior. GTO's greedy approach hurts performance here, likely due to reduced parallelism.
- Cache-Friendly Workloads: backprop_8192 and crystal_q12 show moderate miss rates and benefit from GTO's locality improvements.

Algorithm Trade-offs

GTO Benefits:

- Improved cache locality through warp persistence
- Reduced context switching overhead
- Better performance for workloads with good temporal locality

GTO Drawbacks:

- Reduced parallelism when warps have poor locality
- Potential for increased stall cycles in memory-intensive workloads
- May not adapt well to changing access patterns

CCWS Characteristics:

- Sophisticated feedback mechanism for adaptive throttling
- Minimal performance impact in current benchmarks
- May be more effective with higher cache pressure or different workload characteristics