Evaluating Warp Schedulers for Graphics Workloads

Aurora Tomás[1], Juan L. Aragón[2] and Antonio González[1]

[1]Universitat Politècnica de Catalunya
[2]Universidad de Murcia

Abstract

Domain-specific computing systems are increasingly important for meeting performance and energy-efficiency demands, with GPUs being a notable example. GPUs rely on massive multithreading to tolerate long-latency operations, yet limited on-chip resources, inefficient scheduling, and communication bottlenecks can still lead to hardware underutilization. Prior work has shown that warp scheduling can significantly affect GPU performance, motivating considerable research effort toward exploring novel warp scheduling mechanisms, particularly for GPGPU workloads, where intra-warp locality is important due to their program characteristics. In this work, we evaluate eight state-of-the-art warp schedulers for graphics workloads using scenes from 32 commonly used graphics applications on a mobile GPU, an evaluation that, to the best of our knowledge, has not been previously conducted. Our results show that none of these schedulers outperforms the baseline Loose Round-Robin (LRR). We further characterize these workloads and show that graphics applications exhibit execution and locality patterns that differ substantially from those assumed by GPGPU-oriented schedulers. These findings highlight the need for warp scheduling mechanisms specifically designed for the distinct execution characteristics of graphics workloads.

1 Introduction

Contemporary GPUs comprise many shader cores that exploit the inherent parallelism of graphics workloads, where computations for different pixels can often proceed independently. To maximize throughput and hide memory latency, shader cores support a large number of concurrent threads and follow the SIMT execution model, in which threads are grouped into *warps* and execute in lock-step on SIMD hardware.

Each shader core can issue a limited number of instructions per cycle. The warp scheduler selects which ready warp issues an instruction each cycle, making the issue stage a key determinant of how effectively the core utilizes its execution resources. After issue, the instruction reads its source operands from the Register File and is then dispatched to the appropriate execution pipeline. Arithmetic instructions are executed by the ALUs, whereas memory instructions access the memory hierarchy through the memory pipeline. Consequently, scheduling decisions can strongly affect operand readiness, cache locality, memory pressure, latency hiding, and overall execution time.

Architects have devoted significant effort to developing warp schedulers that improve GPU efficiency. Loose Round Robin (LRR) is commonly used as a baseline because of its simplicity and fairness, selecting ready warps in round-robin order while skipping stalled or ineligible ones. Greedy-Then-Oldest (GTO) continues issuing from the current warp while it can make progress and otherwise selects the oldest ready warp, favoring intra-warp locality. Oldest-first (OLD) always prioritizes the oldest ready warp, reducing starvation, whereas Greedy-Then-Youngest (GTY) [1] continues with the current warp until it stalls and then selects the youngest ready warp. Two-Level Active Scheduler (TLAS) [2] introduces hierarchical scheduling by first selecting a subset of active warps and then choosing the next warp from this reduced candidate set, reducing scheduling complexity while retaining latency-hiding capability.

These schedulers follow fixed policies and do not adapt to workload behavior at runtime. More sophisticated proposals therefore use dynamic information to tailor scheduling decisions to program characteristics. In our evaluation, we consider three widely studied adaptive schedulers: CCWS, CAWS, and MASCAR.

CCWS [3] targets cache-sensitive workloads by exploiting intra-warp locality (i.e., reuse among memory accesses issued by the same warp). CAWS [4] addresses large differences in warp completion times within a thread block by prioritizing critical warps, aiming to reduce the time until the block releases its resources. MASCAR [5] targets memory-intensive phases by reducing memory-system contention, preventing an excessive number of outstanding misses from saturating structures such as MSHRs and delaying otherwise ready memory accesses.

However, these scheduling mechanisms have been primarily explored for general-purpose GPU workloads. In this study, we evaluate these schedulers on graphics applications on a mobile GPU, which, to the best of our knowledge, has not been previously explored. Our results show that none of the evaluated schedulers provides a performance advantage over the baseline LRR policy in graphics applications. This is mainly because their execution characteristics differ substantially from those of GPGPU workloads, and the assumptions underlying these schedulers are not tailored to them. These findings suggest that warp scheduling for graphics workloads remains an important design space that deserves further exploration.

Table 1: *GPU simulation parameters.*

	Global Parameters
Tech Specs	800 MHz, 1V, 22nm
Screen Resolution	1920x1080 (Full HD)
Tile Size	32x32 pixels
Number of cores	8
Main Memory	LPDDR4, 8GB
L2 Cache (shared)	64-bytes/line, 2MB, 8-way
	Core Configuration
# Warps	64
Issue width	4
L1 Texture Cache	64-bytes/line, 32KB, 4-way
# MSHRs (per cache)	128
Warp Schedulers	LRR, GTO, GTY, OLD, TLAS, CAWS, CCWS, MASCAR

Table 2: *Evaluated benchmarks.*

Benchmark	**Footprint (MB)**	**Benchmark**	**Footprint (MB)**
Air Attack	2.1	Gravity Tetris	0.8
Among Us	3.4	Gravity: Don't Let Go	5.2
Angry Birds	1.1	Hill Climb Racing	2.8
Archery Master 3D	2.3	Hot Wheels: Race Off	8.3
Beach Buggy Racing	7.5	Jetpack Joyride	0.7
Block Blast!	3.6	3D Maze / Labyrinth	3.6
Candy Crush Saga	2.6	Plants vs. Zombies	2.7
Captain America	1.5	World Robot Boxing	4.9
City Racing 3D	3.5	Rise of Kingdoms	7.1
Clash of Clans	2.4	Royal Match	27.5
Counter Strike	0.7	Sniper 3D	6.0
Crazy Snowboard	0.9	Sonic Dash	4.8
Derby Destruction	3.2	Subway Surfers	2.9
Forest 2	4.7	Tetris	9.4
Geometry Dash Lite	1.2	Tigerball	6.5
Golf Battle	4.9	Vegas Crime Simulator	4.3

2 Evaluation Methodology

We use the TEAPOT [6] cycle-accurate simulator, which runs unmodified Android applications and models an ARM Valhall-like mobile GPU. Table 1 summarizes the GPU configuration adopted in our experiments. LRR serves as the baseline, and we extend TEAPOT to evaluate GTO, GTY, OLD, TLAS, CAWS, CCWS, and MASCAR.

Our evaluation uses a broad set of commercial Android graphics applications spanning different genres and graphics complexities. Table 2 summarizes the selected benchmarks, whose average memory footprint per frame is nearly 5 MB, approximately twice the L2 cache capacity, placing significant pressure on the memory hierarchy.

3 Results & Discussion

Figure 1 reports the per-benchmark IPC variation of the evaluated warp schedulers compared to the baseline LRR policy. Overall, none of the evaluated schedulers provides a significant performance improvement over LRR.

We characterize several current commercial graphics applications on a mobile GPU and observe substantially different warp-level behavior from that reported for GPGPU workloads in prior works. The evaluated graphics applications execute approximately 800K warp instances per frame on average, with each warp executing only about 13 instructions and 1–2 texture instructions on average. Prior GPGPU characterization reports substantially longer warp instruction streams and, thus, more memory instructions per warp [7]. These short graphics shaders are also consistent with previous PC-game characterizations, which reported approximately 14.4 fragment-shader instructions and 2.9 texture instructions on average [8]. Although these workloads correspond to an older generation of desktop games, their shader lengths closely match those observed in our current mobile graphics applications. This similarity is consistent with the evolution of mobile GPUs, whose increased computational capabilities have brought current mobile graphics workloads closer to those of earlier desktop systems.

Among the evaluated warp schedulers, TLAS exhibits the largest degradation, reducing IPC by 20.1% on average. By restricting scheduling decisions to only one quarter of the active warps, TLAS relies on this reduced subset to provide sufficient instruction-level and thread-level parallelism to keep the execution pipelines busy. This assumption does not hold well for our graphics applications, which exhibit frequent data dependencies and memory operations that limit the number of warps able to make progress without stalling. Compared with typical GPGPU workloads, graphics shaders are substantially shorter and provide less opportunity to expose sufficient independent work within a restricted set of warps. Consequently, when several warps in the selected subset are stalled, fewer alternative warps are available to hide their latency. As a result, limiting the scheduler to a small subset of warps significantly reduces its latency-hiding capability in graphics workloads.

GTO, CCWS, and GTY provide the same performance as LRR on average, which is consistent with our expectations. These schedulers are designed to exploit intra-warp locality, which is particularly important in general-purpose workloads. However, the small number of memory accesses issued by each graphics warp provides limited opportunity for reuse within the same warp. Instead, graphics locality is dominated by inter-warp reuse, as neighboring fragments often access the same texture data. For example, a cache line fetched by one warp can subsequently be reused by another. This difference explains why schedulers designed to promote intra-warp locality provide little benefit for graphics workloads.

OLD provides only a negligible average IPC improvement of 0.1%. OLD prioritizes the oldest eligible warp and is effective when there is a large disparity in waiting time among eligible warps. However, such disparities are not prevalent in our graphics workloads, limiting the benefit of this policy and resulting in a minor performance advantage over LRR.

Similarly, CAWS achieves essentially the same performance as LRR. CAWS targets disparities in completion time among warps within the same CTA by prioritizing critical warps to accelerate CTA completion and resource release. However, such execution organization is less repre-

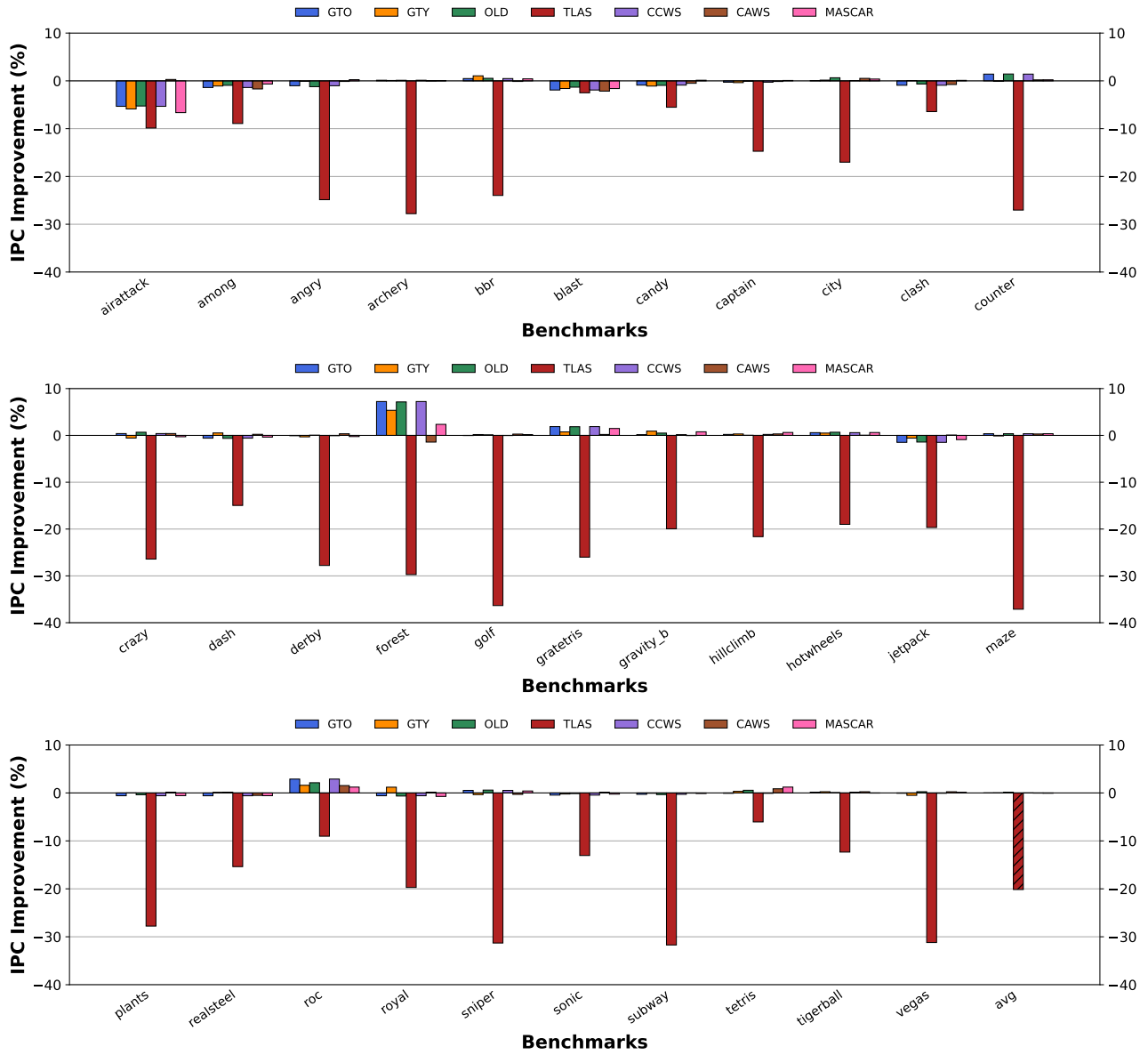


Figure 1: *Per-benchmark IPC improvement of the evaluated warp schedulers compared to LRR.*

sentative of graphics workloads and provides no additional performance benefit over LRR on average.

Finally, MASCAR provides a minor performance loss, with an average IPC change of -0.1%. MASCAR is designed for memory-intensive execution phases in which a warp can issue several independent memory requests before reaching an instruction that depends on one of the outstanding loads. This behavior differs substantially between graphics and GPGPU workloads. Graphics shaders typically exhibit short sequences of ALU instructions followed by a texture instruction that is immediately followed by a dependent instruction. In our workloads, the first consumer of a texture instruction appears after only 1.3 instructions on average. Consequently, graphics warps expose a short instruction window between a texture access and its dependent use, leaving little opportunity to issue additional independent memory instructions before stalling. This makes MASCAR a poor match for graphics workloads and explains its negligible impact on performance.

In summary, graphics workloads expose abundant thread-level parallelism through a large number of short-lived warps. This behavior provides fewer opportunities for schedulers designed around fewer, longer-running warps, while favoring the simplicity and fairness of LRR.

4 Potential Directions

Potential future work should investigate warp scheduling mechanisms tailored to the specific characteristics of graphics workloads. As shown by our analysis, graphics applications exhibit shorter, more dependency-heavy shaders with fewer memory instructions per warp than typical GPGPU workloads, while their locality is largely driven by inter-

warp reuse. Future schedulers could therefore incorporate information about recently accessed memory regions when making scheduling decisions, with the goal of identifying short-term reuse opportunities across warps.

5 Conclusion

Our analysis shows that existing warp schedulers provide no benefit for graphics workloads because they are tailored to GPGPU applications. LRR remains the most suitable choice among the evaluated schedulers due to its simplicity and fairness, while its round-robin execution can help preserve inter-warp spatial locality in graphics workloads. However, dynamically exploiting inter-warp spatial locality remains largely unexplored for warp scheduling in graphics workloads.

Acknowledgment

This work has been supported by the Spanish State Research Agency (MCIU/AEI/10.13039/501100011033), FEDER/UE and the European Social Fund Plus (ESF+) under grants PID2020-113172RB-I00 and PID2024-155476OB-I00, the ICREA Academia program and the FPI research grant PRE2021-100336.

References

[1] Rodrigo Huerta, Mojtaba Abaie Shoushtary, José-Lorenzo Cruz, and Antonio Gonzalez. "Dissecting and Modeling the Architecture of Modern GPU Cores". In: *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 2025, pp. 369–384.

[2] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, Rustam Miftakhutdinov, Onur Mutlu, and Yale N. Patt. "Improving GPU Performance via Large Warps and Two-Level Warp Scheduling". In: *MICRO*. 2011, pp. 308–317.

[3] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. "Cache-Conscious Wavefront Scheduling". In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 2012, pp. 72–83.

[4] Shin-Ying Lee and Carole-Jean Wu. "CAWS: Criticality-Aware Warp Scheduling for GPGPU Workloads". In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. 2014, pp. 175–186.

[5] Ankit Sethia, D. Anoushe Jamshidi, and Scott Mahlke. "Mascar: Speeding up GPU Warps by Reducing Memory Pitstops". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015, pp. 174–185.

[6] Jose-Maria Arnau, Joan-Manuel Parcerisa, and Polychronis Xekalakis. "TEAPOT: A Toolset for Evaluating Performance, Power and Image Quality on Mobile Graphics Systems". In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. 2013, pp. 37–46.

[7] Nagesh B Lakshminarayana and Hyesoon Kim. "Effect of Instruction Fetch and Memory Scheduling on GPU Performance". In: *Workshop on Language, Compiler, and Architecture Support for GPGPU*. Vol. 88. 2010.

[8] Jordi Roca, Victor Moya, Carlos González, Chema Solís, Agustín Fernández, and Roger Espasa. "Workload Characterization of 3D Games". In: *2006 IEEE International Symposium on Workload Characterization*. IEEE. 2006, pp. 17–26.