Macsim Mini: A Lightweight Cycle-Level GPU Simulator for Architecture Education

Euijun Chung*, Saurabh Singh*, Huanzhi Pu, Yuxiao Jia, Anurag Kar, Sam Jijina, Scott Madeira, Hyesoon Kim
{*echung67, saurabh.s, hpu8, yjia305, akar34, sam.jijina, scottmadeira*}*@gatech.edu, hyesoon@cc.gatech.edu*
Georgia Institute of Technology

***Abstract*—Cycle-level GPU simulators are valuable educational tools, but existing frameworks are either too complex for students to navigate or too abstract to convey microarchitectural details. We present Macsim Mini, a lightweight cycle-level GPU simulator designed for computer architecture education. By concentrating on the memory hierarchy and thread scheduling rather than detailed compute pipelines, Macsim Mini captures the architectural trade-offs most central to GPU performance in a codebase small enough for students to read and modify within course assignments. Macsim Mini has been deployed in a graduate-level GPU architecture and programming course for seven semesters, serving ∼1,000 students with high completion rates and average scores above 90%.**

***Index Terms*—GPU simulation, computer architecture education, cycle-level modeling**

I. Introduction

Microarchitecture simulators are widely used in computer architecture research and are equally valuable as educational tools. CPU simulators such as ChampSim [4] and gem5 [16] allow students to implement and evaluate components such as cache replacement policies and prefetchers, deepening their understanding beyond lectures alone [5], [20]. However, no comparable educational tool exists for GPU architectures.

GPUs pose a greater educational challenge than CPUs. Their massively parallel execution model exposes low-level hardware details and requires programmers to explicitly manage thread scheduling, memory coalescing, and specialized units such as tensor cores [12], [14], [19]. The tight coupling between software and hardware requires GPU programmers to understand both the software and the underlying architecture, making simulation-based learning particularly valuable.

Despite this need, existing GPU simulators pose significant barriers to educational use. Research-grade cycle-level simulators such as GPGPU-Sim [1], Macsim [11], MGPUSim [27], and Accel-Sim [10] prioritize accuracy over accessibility, resulting in large, detailed codebases that are difficult for students to navigate within a semester. At the other extreme, functional models [6], [25] are too abstract to convey microarchitectural details. A gap remains for a simulator that is detailed enough to be instructive yet simple enough for students to learn, modify, and conduct performance analysis.

We present *Macsim Mini*, a lightweight, cycle-level GPU simulator designed for computer architecture education. Following ZSim's insight [26] that memory is the primary throughput bottleneck in massively parallel processors, corroborated by empirical evidence that memory-system modeling errors dominate GPU simulation inaccuracy [9], Macsim Mini uses fixed-latency compute but models the cache hierarchy and warp/block scheduling in detail. Table I summarizes how Macsim Mini compares to Accel-Sim [10]. As Table II shows, by focusing on modeling the memory system while keeping the compute model simple, Macsim Mini achieves fast simulation speeds while allowing students to observe meaningful architectural trade-offs in GPUs.

Table I. Comparison of Accel-Sim and Macsim Mini.

	Accel-Sim [10]	Macsim Mini (ours)
Granularity	Cycle-level	Cycle-level
Simulation speed (MIPS)*	1.46	14.1
Lines of code	64.2 K	3.8 K
Accuracy vs. real HW	High	Trend-accurate

*Arithmetic mean of MIPS (Million Instructions Per Second), measured on Intel Xeon E5-2696 v4 CPU with the Rodinia benchmark suite [2].

Table II. Macsim Mini's modeled components, configurable parameters, and enabled analysis and experiments.

Sim. Component	Configurable Params	Analysis & Experiments
Warp/block scheduling	# SMs, # warps per SM, scheduling policy	RR/GTO/CCWS comparison, occupancy
L1/L2 cache	Cache size, latency, associativity, repl. policy	Cache capacity trade-offs, replacement policy
Tensor core	Execution width, latency	FP16 vs. 32 throughput
Execution units	# units, width, latency per type	Compute throughput scaling
DRAM backend	Latency, backend selection (Ramulator)	Memory latency sensitivity
RAW dependency	–	Pipeline stall analysis
Memory coalescing	–	Access pattern impact
Trace-level configuration		
Kernel selection	Kernel IDs	Kernel sampling for large workloads [3]
Not modeled in detail (fixed-latency or omitted)		
Compute pipeline	Fixed latency	–
Sub-core, NoC, MSHR, Addr. translation, Operand collector	Omitted	–

II. Macsim Mini

A. Design and Simulation Flow

Macsim Mini focuses on the two dimensions that most directly determine GPU performance: compute throughput and memory latency hiding [7], [8]. By configuring parameters shown in Table II, such as warp/block scheduling, execution units, and L1/L2 cache, Macsim Mini captures the correct performance trends, which are sufficient for students to understand why one design choice outperforms another.

The simulation flow of Macsim Mini is shown in Fig. 1. First, instruction traces are collected from CUDA binaries

*Equal contribution.

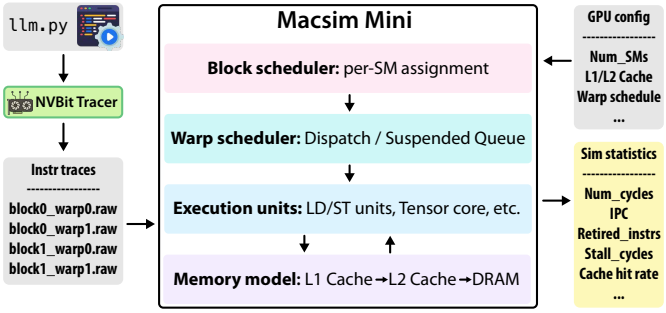

Fig. 1. Macsim Mini's simulation flow. Custom NVBit tool traces CUDA binaries, and the resulting instruction traces are fed into the simulator.

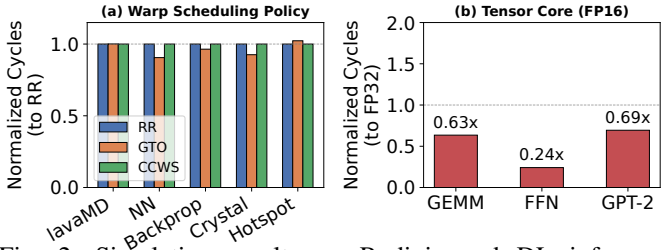

Fig. 2. Simulation results on Rodinia and DL inference benchmarks: (a) cycle counts under different warp scheduling policies, normalized to round-robin, and (b) FP16 tensor core cycle counts normalized to FP32 baseline.

using an NVBit-based SASS instruction-level tracer [29]. Macsim Mini is then run on the traces with varying parameters such as the number of SMs (Streaming Multiprocessors), L1/L2 cache sizes, and warp/block scheduling policies. The tracer automatically determines runtime properties such as blocks per SM based on shared memory usage.

B. Execution Model

In Macsim Mini's execution model, the block scheduler assigns thread blocks to SMs based on resource availability (shared memory and registers). Within each SM, a two-level scheduling scheme [18] manages warps: a *dispatch queue* for ready warps and a *suspended queue* for those awaiting memory responses. The scheduler selects one warp per cycle from the dispatch queue; cycles without a ready warp are recorded as stall cycles count, a key metric for observing how scheduling policies affect latency hiding. At the instruction level, compute operations are modeled using execution units with configurable widths and latencies that respect RAW (Read-After-Write) dependencies. The simulator uses a single top-level loop to advance all cores cycle by cycle, making the overall control flow easy to follow and modify.

C. Memory Model

Memory instructions from each core are first sent to a private L1 cache, then to a shared L2 cache upon a miss. For DRAM, Macsim Mini provides a modular wrapper that supports multiple backends: a fixed-latency model for simplicity; a lightweight model with bus, bank, and row-buffer modeling for more realistic behavior; or an interface to Ramulator [13], [17], depending on the user's educational or research needs.

III. Course Integration

Macsim Mini has been deployed in a graduate-level GPU architecture and programming course at Georgia Tech since Spring 2024. Over seven semesters, 976 students used the simulator as part of course assignments.

A. Assignment Design

The project consists of two tasks built on the same simulator codebase. In the first task, students implement three warp scheduling policies: round-robin (RR), greedy-then-oldest (GTO), and Cache-Conscious Wavefront Scheduling (CCWS) [24], and compare their impact on memory latency hiding and overall performance. In the second task, students add tensor core support by implementing execution-width-aware dispatch logic, enabling them to observe how dedicated low-precision matrix-multiply units affect throughput.

Students evaluate their implementations on a set of provided benchmarks spanning the Rodinia suite [2] and PyTorch model inference workloads (general matrix multiply (GEMM) [23], feed-forward network (FFN) [28], and GPT-2 [21]) at both full and half precision. This mix exposes students to both traditional high-performance computing (HPC) kernels and deep learning workloads.

B. Simulation Results

Fig. 2 shows representative results. For warp scheduling (a), GTO improves cache locality, reducing cycle counts by up to 10% compared to round-robin. In contrast, CCWS shows minimal improvement as inter-warp contention stays below its throttling threshold for the given traces. For tensor cores (b), FP16 yields substantial cycle reductions on GEMM (0.63×), FFN (0.24×), and GPT-2 (0.69×), demonstrating the throughput benefits of dedicated low-precision matrix units. GTO's advantage over round-robin on cache-sensitive workloads and the speedup from FP16 tensor cores on compute-dense kernels such as GEMM and FFN are consistent with prior findings [7], [10], [15], [22], [24].

C. Student Outcomes

Each assignment was completable within three weeks. Although most students had no prior simulator experience, completion rates consistently exceeded 90%, and average scores remained above 90%. These results suggest that Macsim Mini's compact codebase and clear control flow effectively lower the barrier to entry for cycle-level GPU simulation.

IV. Discussion & Conclusion

Macsim Mini's modular design makes it straightforward to extend: researchers and instructors can add execution unit types or swap the DRAM backend (e.g., Ramulator [13], [17]). We plan to support additional opt-in modules (e.g., sub-core partitioning, `LDGSTS`, and TMA engine) to enable more sophisticated modeling while keeping the baseline simple. By focusing on the memory hierarchy rather than detailed compute modeling, Macsim Mini has enabled students to implement and evaluate GPU microarchitectural components within a compact, instructive codebase.

REFERENCES

[1] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009, pp. 163–174.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2009, pp. 44–54.

[3] E. Chung, S. Na, S. H. Kang, and H. Kim, "Swift and trustworthy large-scale gpu simulation with fine-grained error modeling and hierarchical clustering," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 1397–1411.

[4] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.

[5] H. Grunbacher, "Teaching computer architecture/organisation using simulators," in *FIE'98. 28th Annual Frontiers in Education Conference. Moving from'Teacher-Centered'to'Learner-Centered'Education. Conference Proceedings (Cat. No. 98CH36214)*, vol. 3. IEEE, 1998, pp. 1107–1112.

[6] D. Hettiarachchi, "Tinygpu," https://github.com/deaneeth/tinygpu, 2026.

[7] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 152–163.

[8] R. Huerta, M. A. Shoushtary, J.-L. Cruz, and A. 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.

[9] A. Jain, M. Khairy, and T. G. Rogers, "A quantitative evaluation of contemporary gpu simulation methodology," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, pp. 1–28, 2018.

[10] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

[11] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, "Macsim: A cpu-gpu heterogeneous simulation framework user guide," *Georgia Institute of Technology*, pp. 1–57, 2012.

[12] H. Kim, R. Vuduc, and S. Baghsorkhi, *Performance analysis and tuning for general purpose graphics processing units (GPGPU)*. Morgan & Claypool Publishers, 2012, vol. 20.

[13] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[14] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[15] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers *et al.*, "Analyzing machine learning workloads using a detailed gpu simulator," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 151–152.

[16] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[17] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," *arXiv preprint arXiv:2308.11030*, 2023.

[18] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving gpu performance via large warps and two-level warp scheduling," in *Proceedings of the 44th Annual IEEE/ACM international symposium on microarchitecture*, 2011, pp. 308–317.

[19] NVIDIA, "Cuda programming guide," https://docs.nvidia.com/cuda/cuda-programming-guide/index.html, 2026.

[20] P. Prasad, A. Alsadoon, A. Beg, and A. Chan, "Using simulators for teaching computer organization and architecture," *Computer applications in engineering education*, vol. 24, no. 2, pp. 215–224, 2016.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[22] M. A. Raihan, N. Goli, and T. M. Aamodt, "Modeling deep learning accelerator enabled gpus," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 79–92.

[23] B. Research, "Deepbench," https://github.com/baidu-research/DeepBench, 2016.

[24] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 72–83.

[25] M. Sabry, "Softgpu: A simple educational gpu simulator," https://github.com/mhmdsabry/SoftGPU, 2024.

[26] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[27] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao *et al.*, "Mgpusim: Enabling multi-gpu performance modeling and optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 197–209.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[29] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 372–383.