Swift-Sim: A Modular and Hybrid GPU Architecture Simulation Framework

Xiangrong Xu, Yuanqiu Lv, Liang Wang*, Limin Xiao*, Meng Han, Runnan Shen, Jinquan Wang
School of Computer Science and Engineering, Beihang University, Beijing 100191, China
{xxr0930, 20373273, lwang20, xiaolm, hanm, shenrn, derekjqwang}@buaa.edu.cn

***Abstract*—Simulation tools are critical for architects to quickly estimate the impact of aggressive new features of GPU architecture. Existing cycle-accurate GPU simulators are typically cumbersome and slow to run. We observe that it is time-consuming and unnecessary for cycle-accurate GPU simulators to perform detailed simulations for the entire GPU when exploring the design space of specific components. This paper proposes Swift-Sim, a modular and hybrid GPU simulation framework. With a highly modular design, our framework can choose appropriate modeling approaches for each component according to requirements. For components of interest to architects, we use cycle-accurate simulation to evaluate new GPU architectures. For other components, we use analytical modeling, which accelerates simulation speed with only minor and acceptable degradation in overall accuracy. Based on this simulation framework, we present two working examples of hybrid modeling that simulate the ALU pipeline and memory accesses using analytical models. We further implement two GPU performance simulators with different levels of simplification based on Swift-Sim and evaluate them using configurations from real GPUs. The results show that the two simulators achieve an 82.6x and 211.2x geometric mean speedup compared to Accel-Sim with insignificant accuracy degradation.**

***Index Terms*—GPU, architecture, performance modeling, simulation**

I. Introduction

GPUs have become indispensable computing platforms widely used in machine learning, high-performance computing, and data analytics workloads. As applications grow in complexity, the demand for innovative architectures to meet computational needs is on the rise [1]–[4]. Performance evaluation is critical for GPU architects to estimate the impact of aggressive new features and make correct design decisions [5]–[13].

Cycle-accurate simulation is the prevalent performance evaluation approach for exploring new GPU architecture design spaces. GPGPU-Sim [5], Accel-Sim [9], etc., are well-known cycle-accurate GPU performance simulators. GPGPU-Sim, the most representative cycle-accurate GPU performance simulator, carefully simulates the state of GPU components and ticks them each cycle. As the state-of-the-art cycle-accurate GPU performance simulator, Accel-Sim upgrades GPGPU-Sim's performance model to more faithfully simulate various features of modern GPUs [9]. These cycle-accurate simulators often accurately simulate the execution of applications, allowing architects to evaluate their innovative architectural designs through detailed simulation.

However, cycle-accurate simulators spend considerable time carefully simulating each component, making them run very slowly. For example, they take days or even months to complete the simulation for some large benchmarks [14]–[16]. In fact, much of the simulation time is unnecessary for architects to explore the architectural design space. In most cases, architects use GPU modeling tools primarily to evaluate and optimize specific GPU components, such as GPU cores or the memory hierarchy [1]–[4], [17]. In these situations, GPU performance simulators do not need to perform cycle-accurate simulations for the entire GPU. Instead, they can selectively use cycle-accurate simulations to model the specific modules of interest to the architects, while using non-cycle-accurate modeling methods such as mathematical models for other parts. This innovative approach not only meets the needs of architects to explore the design space but also improves the speed of GPU simulation.

In this work, we propose Swift-Sim, a modular and hybrid GPU simulation framework where each component is simulated by a separate module. The modular design allows architects to simulate each module using cycle-accurate simulations or other simplified modeling approaches. For modules of interest to architects, we use cycle-accurate simulation to evaluate new GPU designs. For other modules, we use non-cycle-accurate models to speed up the simulation. This hybrid framework accelerates GPU simulations with minor and acceptable degradation in overall accuracy.

To evaluate our design, we implement two GPU performance simulators with different simplification levels based on the Swift-Sim framework, named Swift-Sim-Basic and Swift-Sim-Memory. We compare the performance predictions of the two simulators with those of the state-of-the-art cycle-accurate simulator, Accel-Sim [9]. Our experiments demonstrate that the Swift-Sim-Basic simulator achieves a geometric mean speedup of 82.6x compared to Accel-Sim while maintaining comparable accuracy. The Swift-Sim-Memory simulator further simplifies memory access modeling using a classical analytical model, achieving a geometric mean speedup of 211.2x compared to Accel-Sim with only an insignificant degradation in accuracy. Additionally, we validate that the Swift-Sim framework exhibits high accuracy across three different real GPU architectures, demonstrating its effectiveness in exploring the architectural

[*] Liang Wang and Limin Xiao are corresponding authors.

This work was supported in part by the National Key R&D Program of China under Grant No. 2023YFB4503100; in part by the National Natural Science Foundation of China under Grants 62272026 and 62104014; in part by the Fundamental Research Funds for the Central Universities; in part by State Key Laboratory of Complex & Critical Software Environment under Grant No. CCSE-2024ZX-10.

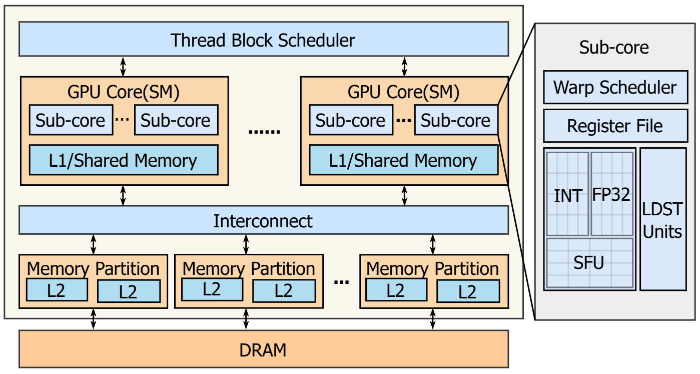

Fig. 1. The Modeled GPU Architecture.

design space.

This paper makes the following contributions:

- We observe that cycle-accurate simulation of the entire GPU is unnecessary when architects focus on specific components. Using analytical models for less critical components does not prevent design space exploration.
- We propose a modular hybrid GPU simulation framework that allows architects to simulate each module with either cycle-accurate or simplified models, accelerating simulations while meeting architectural exploration needs.
- We develop two GPU performance simulators with different simplification levels and evaluate them using real GPU configurations. Our two simulators achieve geometric mean speedups of 82.6x and 211.2x compared to Accel-Sim, respectively, with insignificant accuracy degradation.

II. Related Work and Motivation

A. GPU Architecture

As shown in Figure 1, modern GPUs consist of a number of streaming multiprocessors (SMs, also called compute units) that consist of multiple sub-cores. The sub-cores contain warp schedulers, register files, various execution units (e.g., INT units, SP units, SFU units, tensor cores, etc.), Load/Store units, etc. All sub-cores share the sectored L1 data cache and shared memory in the SM [11]. The SMs in a GPU share a banked L2 cache, and they are connected to the L2 cache via on-chip interconnects [3]. When a memory request misses in both levels of the cache, it is directed to DRAM to fetch the data, and the requested data is then filled back into the two-level cache.

In recent years, the rapid evolution of GPU architectures has prompted architects to quickly explore the design space of future GPUs to achieve architectural innovation [9], [11], [18].

B. Related Work

Architecture performance evaluation is essential for design space exploration [5], [9], [19]–[25]. With the rapid evolution of GPUs, architects require design tools that can quickly evaluate the impact of aggressive new features.

Cycle-accurate simulation is a widely used and highly accurate performance evaluation method. Cycle-accurate simulators like GPGPU-Sim [5] and Accel-Sim [9] accurately simulate each GPU component, periodically update the states of simulated components, and gather performance metrics critical to architects. Cycle-accurate simulators meticulously model each GPU component with thorough code, allowing architects to explore the design space of individual components and simulate innovative design ideas in detail. However, the speed of cycle-accurate simulations is still far from satisfactory. For example, the performance model of Accel-Sim is based on the GPGPU-Sim 4.0 simulator with fine-grained and accurate simulation. It carefully simulates streaming multiprocessors, interconnection networks, memory partitions, HBM/GDDR, and other GPU components cycle by cycle. As a result, it achieves high accuracy but runs very slowly. As mentioned in [14]–[16], it takes several days or even months for the cycle-accurate simulator to simulate large applications, and architects have to spend more evaluating architectural designs by running simulations on dozens of benchmarks.

Other studies have proposed analytical modeling techniques [11], [18], [26], [27]. The analytical modeling approaches utilize mathematical equations to calculate the GPU performance metrics, which reduce the modeling time significantly but they are not suitable for fine-grained architectural exploration. GPUMech [26], MDM [18], GCoM [11], and other analytical models calculate the additional latency due to various resource contention through mathematical equations to derive the overall GPU performance. However, analytical modeling supports fewer architectural design parameters due to the lack of detailed simulation of individual GPU components, making it difficult to evaluate small performance variations between different design alternatives. For example, GPU cache analytical model based on reuse distance theory [27] typically assumes that the cache replacement policy is *LRU*, which makes it difficult to simulate other replacement policies such as *FIFO* or *Random*. Additionally, models like MDM and GCoM often use queueing models to simulate on-chip networks. When the NoC topology changes, a new analytical model has to be created, resulting in poor scalability.

Furthermore, studies like [15], [16], [28]–[30] adopt sampling-based performance estimation methods where they simulate a small portion of the application and then estimate the desired GPU performance metrics based on existing metrics. They still rely on cycle-accurate simulation or analytical models for the sampled application. Therefore, these studies are orthogonal to our work.

C. Motivation

Existing cycle-accurate GPU simulators are still far from being able to quickly explore new architectures. To this end, we propose a simulation framework that combines cycle-accurate simulation with analytical models. The key observation behind our framework is that cycle-accurate simulation of the entire GPU is unnecessary for architects exploring specific GPU components.

Architects typically use simulation to explore only a few specific GPU components. For components that architects are not interested in, even if they use analytical models lacking sufficient architectural details, it does not affect architects'

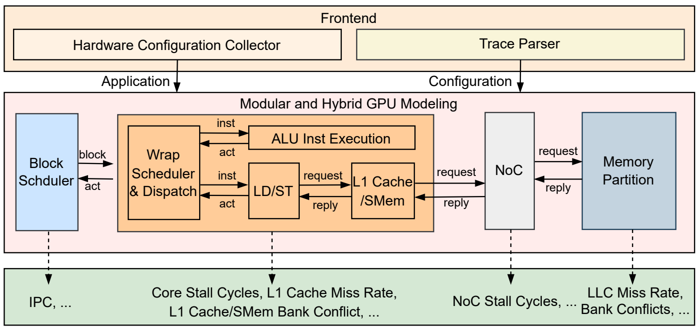

Fig. 2. Overview of Swift-Sim framework.

exploration of the design space of specific components. On the other hand, we find that using appropriate analytical models to model some components results in minor and acceptable degradation in accuracy. Therefore, architects can effectively evaluate their new designs as long as the modules of interest to the architecture are modeled using cycle-accurate simulation, while other modules can be simplified using analytical modeling.

This provides us with the opportunity to develop a hybrid and fast cycle-accurate simulator. For components that are of interest to architects, we use cycle-accurate simulation to evaluate new GPU architectures. For other components, we use analytical models, which speed up simulation with an insignificant impact on overall accuracy. However, it is challenging to effectively combine cycle-accurate simulation with analytical models if the simulator has a low modularity [31]. Therefore, it is imperative to develop a hybrid simulation framework that adopts highly modular modeling to minimize redundant interactions between components and supports adaptable modifications to the modeling approaches for each component. Through this framework, architects can simplify many modules in the simulator to significantly increase its simulation speed.

III. Design and Implementation

In this section, we present the design and implementation of Swift-Sim, a GPU architecture simulation framework with a highly modular design. As shown in Figure 2, Swift-Sim is composed of the following three parts:

(1) Frontend: The Frontend includes the Hardware Configuration Collector and the Trace Parser, handling the inputs of the performance model.

(2) Modular and Hybrid GPU Modeling: The Modular and Hybrid GPU Modeling models various GPU components and simulates the execution of applications based on the hardware configurations and application traces. It adopts a highly modular and flexible design, allowing architects to modify each component easily.

(3) Metrics Gatherer: The Metrics Gatherer collects various performance metrics, allowing architects to evaluate overall performance and analyze performance bottlenecks based on these metrics.

The design of Swift-Sim is detailed in the following subsections.

A. Frontend

The Frontend gathers hardware configurations for GPU architecture and application traces using the Hardware Configuration Collector and the Trace Parser, respectively. The Hardware Configuration Collector collects and parses modeling parameters from configuration files and provides them to the performance model. Architects can modify configuration file settings, such as GPU core count, L1 cache size, and latency of each execution unit, to simulate new GPU architectures. The Trace Parser collects GPU application traces captured from real hardware using an extension of the NVBit tool [32] and then translates these traces into a format recognizable by the simulator. It is worth noting that the traces in our simulation framework are independent of the GPU architecture and can be collected across various NVIDIA GPUs, regardless of the GPU architecture to be simulated.

B. Modular and Hybrid GPU Modeling

As shown in Figure 2, our GPU performance model, Modular and Hybrid GPU Modeling, receives hardware configuration from the Frontend, models each component of the GPU, and simulates the execution of applications.

1) GPU Execution Model: When an application consisting of many thread blocks is executed on the GPU, the Block Scheduler assigns the blocks to the SMs. After an SM receives the blocks, the Warp Scheduler & Dispatch selects an appropriate warp from the blocks to issue instructions. If the instructions are arithmetic, they will be executed within the execution units. For Load/Store instructions, the memory requests will be sent to the cache through the LD/ST units. Once all instructions in the application have been simulated, the GPU simulator generates and outputs corresponding performance metrics.

2) Modular Modeling Approach: To implement hybrid GPU modeling, it is essential to adopt a modular modeling approach. In our design, each component of the GPU is implemented as an independent module. For example, the Block Scheduler, Warp Scheduler & Dispatch, etc., each constitutes a separate module. To enable different components to be simulated using different modeling approaches, we decouple the modules and abstract out interfaces between them. Thus, when the inputs and outputs of the modules are fixed, our framework can easily modify the modeling approach within the modules.

For example, in Figure 2, the inputs of Warp Scheduler & Dispatch consist of blocks from the Block Scheduler and the instruction completion acknowledgment from the ALU pipeline and LD/ST unit. The outputs are the instructions that need to be executed. Therefore, Warp Scheduler & Dispatch only needs to ensure that the inputs are thread blocks and instruction completion acknowledgment, and the outputs are the instructions to be executed. Using either analytical models or cycle-accurate simulation to model Warp Scheduler & Dispatch does not affect other modules. Another example is that the execution units receive instructions from Warp Scheduler & Dispatch. After executing the instructions, they return the instruction completion acknowledgment to the scheduler. The execution units only need to ensure that their interfaces with

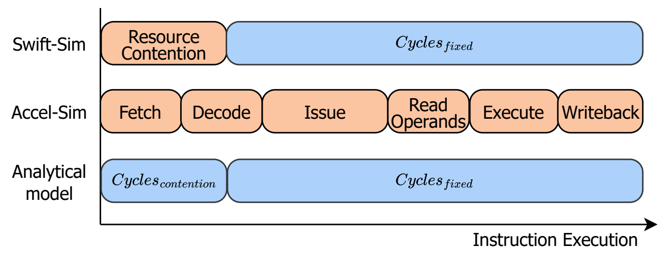

Fig. 3. Modeling the ALU pipeline using an improved analytical model. The orange blocks represent the process simulated cycle-accurately, while the blue blocks represent the process modeled by analytical models.

other modules are consistent. They can be modeled using either analytical models or cycle-accurate simulations.

Compared to cycle-accurate simulators like Accel-Sim, our modular design abstracts fixed interfaces for each module, reduces shared variables across multiple modules, and decreases module coupling. Therefore, we can more conveniently choose different modeling approaches for different modules and minimize unnecessary cycle-accurate simulations. In addition, the modular approach provides us with the opportunity for parallel simulation. We can leverage multithreading to simulate applications concurrently, achieving noticeable speedup. We evaluate the acceleration effects of multithreaded parallel simulation in Section IV-B2.

3) Hybrid Modeling: Based on the modular modeling approach, we can adopt various modeling methods for a single module, thus easily using the analytical model to simplify the cycle-accurate simulator. For example, in Section III-D, we discuss using analytical models to simplify the ALU pipeline and memory access modeling, leveraging the benefits of the modular modeling approach. It is worth noting that we have provided only two examples of using analytical models to simplify cycle-accurate simulators. Architects can also use analytical models for other modules as needed to increase simulation speed.

C. Metrics Gatherer

After modeling, architects can gather various performance metrics from Metrics Gatherer. Metrics Gatherer reads counters from each module and outputs metrics of interest to architects, facilitating performance evaluation of new architectures or diagnosing performance bottlenecks in applications. For example, it gathers total simulation cycles from Block Scheduler after all blocks have completed execution. It also gathers metrics such as core stall cycles, L1 cache miss rate, and cache bank conflicts from the components in SMs. Additionally, it gathers NoC and LLC performance metrics, such as NoC stall cycles and LLC miss rate. Thanks to our highly modular model, architects only need to update the code of the counter within modules to collect the desired metrics.

D. Working Example of Hybrid Modeling

In this section, we present two examples of the simplified cycle-accurate simulator based on our framework using the analytical model. Assuming we need to explore a new warp scheduling algorithm, Warp Scheduler & Dispatch needs cycle-accurate simulation to meet the evaluation requirements of the new architecture. For other modules, architects can choose appropriate modeling methods as needed. We provide two simple analytical models that can be easily integrated into our framework.

1) Analytical Modeling for ALU Pipeline: We now describe how to use analytical models to simplify arithmetic instruction execution. The execution process of arithmetic instructions in a GPU primarily includes *Fetch*, *Decode*, *Issue*, *Read Operands*, *Execute*, and *Writeback*, as shown in Figure 3 [9]. In a cycle-accurate simulator, these processes are accurately simulated cycle by cycle. For example, in Accel-Sim, the simulator updates the states of various components at different pipeline stages each cycle and determines the execution time and performance metrics during *Writeback*. This detailed and accurate simulation requires a lot of code execution, resulting in slower simulation speed.

We observe that cycle-accurate simulation is cumbersome since the execution time of arithmetic instructions remains constant without resource contention. Therefore, simulating the entire process cycle accurately is unnecessary. On the other hand, due to the extensive thread-level parallelism in GPUs, traditional analytical models using mathematical equations often introduce errors when modeling complex resource contention. We propose an improved analytical model for simulating the ALU pipeline. As illustrated in Figure 3, we use cycle-accurate simulation to model delays caused by resource contention. These delays are then added to fixed instruction delays to simulate instruction execution time. Through this design, we can reduce the amount of code execution, thereby increasing simulation speed, while ensuring that modeling accuracy suffers almost little degradation.

2) Analytical Modeling for Memory Accesses: We use a classic memory access analytical model as an example to illustrate how to apply this model to the memory access module in our framework. We calculate the expected latency of each Load/Store instruction using mathematical equations. In GPU applications, different warps execute almost the same instructions. It is a challenge to find a fixed value that objectively represents the latency of instructions for a given PC index across all warps. To this end, we utilize Equation 1 to compute the latency of the Load/Store instructions associated with the specified PC index [26].

$$L_{inst} = L_{L1} \times R_{L1} + L_{L2} \times R_{L2} + L_{DRAM} \times R_{DRAM} \quad (1)$$

In Equation 1, L_{inst} represents the latency of Load/Store instructions for a specific PC index. L_{L1}, L_{L2}, L_{DRAM} denote the access latencies for the L1 cache, L2 cache, and DRAM, respectively, while R_{L1}, R_{L2}, R_{DRAM} represent the cache hit rates obtained using a reuse distance tool or cache simulator, respectively. Equation 1 represents the expected latency for each Load/Store instruction without additional cycles due to NoC and other resource contention.

With this design, what we need to model is not the detailed memory access process, but only the additional latency due to resource contention. When processing a Load/Store instruction, we add the additional latency to the L_{inst} to get the overall instruction latency. It is worth noting that we only describe the method for calculating the latency of instructions accessing global memory. The latency of instructions accessing shared memory can also be calculated using a simple model (omitted due to space constraints).

3) Put All the Modules Together: After specifying the modeling methods for different modules, we combine cycle-accurate simulated modules with analytical model-based modules within our modular simulation framework to model the entire GPU. For example, we use cycle-accurate simulation for the Block Scheduler and Warp Scheduler & Dispatch. In each cycle, the Warp Scheduler & Dispatch issues instructions to the execution units and LD/ST units. Upon receiving the instructions, these units calculate the instruction delay (denoted as X) based on the analytical model and return the instruction completion acknowledgment after X cycles. After getting the instruction completion acknowledgment, the Warp Scheduler & Dispatch then issues the next instruction that depends on the completed instruction, continuing this process until all instructions are executed.

IV. Experimental Evaluation

A. Methodology

1) Experiment Setup: To evaluate the Swift-Sim framework, we compare its total cycle predictions and simulation speed against those of Accel-Sim, the state-of-the-art GPU cycle-accurate simulator. We validate Swift-Sim and Accel-Sim simultaneously against three real GPUs: Nvidia Turing GPU RTX 2080 Ti, Nvidia Ampere RTX 3060, and Nvidia Ampere RTX 3090 [33] whose configs are shown in Table I. We choose the RTX 2080 Ti for a detailed comparison. The main configurations of RTX 2080 Ti are shown in Table II. We use the NVIDIA Nsight Compute tool to collect GPU performance metrics on real hardware. We test the runtime of the simulation on a server with two Intel(R) Xeon(R) Gold 5218R CPUs. In our experiments, we used a maximum of 50 threads.

2) Benchmarks: We validate our GPU performance simulation framework using applications from five benchmark suites: Rodinia [34], Polybench [35], Mars [36], Tango [37], and Pannotia [38]. These applications cover areas such as pattern recognition, graph computing, linear algebra, stencil computations, web data analysis, deep learning, etc. The names of the applications are shown on the axes of Figure 4.

TABLE I
Comparison of Three NVIDIA GPUs.

NVIDIA GPUs	RTX 2080 Ti	RTX 3060	RTX 3090
Architecture	Turing	Ampere	Ampere
Graphics Processor	TU102	GA106	GA102
SMs	68	28	82
CUDA Cores	4352	3584	10496
L2 Cache	5.5MB	3MB	6MB

TABLE II
NVIDIA RTX 2080 Ti GPU configuration.

Parameter	Value
# SMs	68
# Sub-Cores/SM	4
Resources/Sub-core	Warp Scheduler: 1x, GTO
	Exec Units: INT:16x, SP:16x, DP:0.5x, SFU:4x
	LD/ST Units: 4x
L1 in SM	Sectored, streaming, write-through, 4 banks, 128 B/line, 32 B/sector, 256 MSHR entries, 8 maximum merge / MSHR, LRU, 32 cycles
L2 Cache	Sectored, write-back, 128B/line, 32B/sector, 192 MSHR entries, 4 maximum merge/MSHR, LRU, 188 cycles
Memory	22 memory partitions, 227 cycles

3) Performance Simulators: To validate the accuracy and speed of our framework, we develop two GPU performance simulators using the Swift-Sim framework. We first build a simulator called Swift-Sim-Basic based on the Swift-Sim framework by using an analytical model for the ALU pipeline and simplifying less critical modules like instruction cache, constant cache, etc. as described in Section III-D1.

Based on Swift-Sim-Basic, we replace the memory data access modules with the analytical model described in Section III-D2, resulting in Swift-Sim-Memory.

B. Verification Results

1) Performance Prediction: The bar chart in Figure 4 represents the error in predicting the execution cycles of applications on the RTX 2080 Ti GPU by the simulators based on the Swift-Sim framework and Accel-Sim. As shown in Figure 4, the mean error of Swift-Sim-Basic in these applications is 22.6%, which is comparable to Accel-Sim's 20.2%.

In Swift-Sim-Memory, the mean error between the predicted execution cycles and the actual hardware cycles is 24.3%. This is because replacing the entire memory access module with a simplified analytical model introduces minor errors compared to the cycle-accurate simulation, especially in cases of intense resource contention.

It is important to note that we have only used two basic analytical models to illustrate the functionality and benefits of the Swift-Sim framework. If more advanced analytical models are integrated into our framework, the error can be further reduced.

2) Simulation Time: We compare the simulation time of the Swift-Sim-based simulators and Accel-Sim with the same machine configuration. The speedup of the two Swift-Sim-based simulators over Accel-Sim is depicted by the scatter plots in Figure 4. The modeling speed of Swift-Sim-Basic shows an 82.6x geometric mean improvement compared to Accel-Sim, while Swift-Sim-Memory shows a 211.2x geometric mean speedup over Accel-Sim. For the NW, ADI, SM, and GRU applications, Swift-Sim-Memory achieves a speedup of over 1000x compared to Accel-Sim.

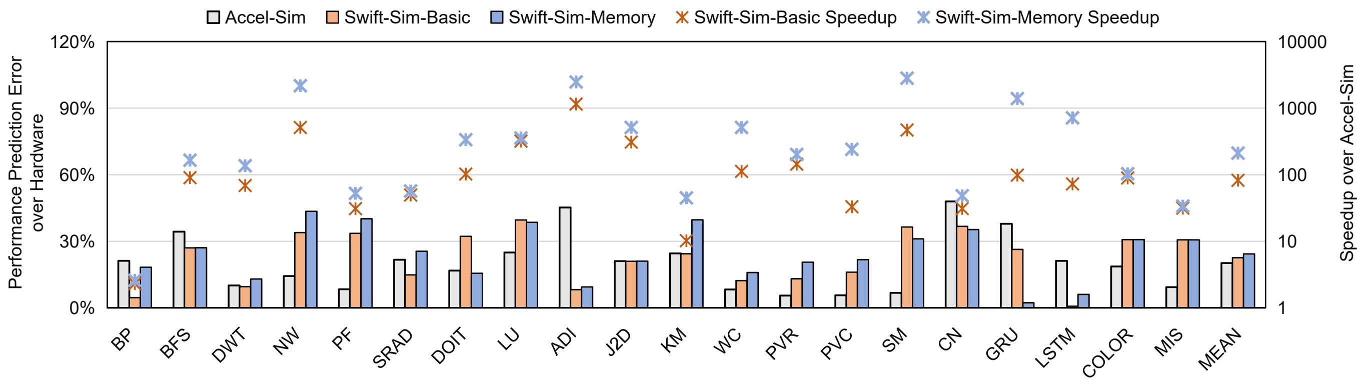

Fig. 4. Comparisons of performance prediction error and simulation speed. The bar chart represents the prediction error of execution cycles for applications on the RTX 2080 Ti GPU by Swift-Sim and Accel-Sim. The scatter plot shows the speedup of Swift-Sim compared to Accel-Sim.

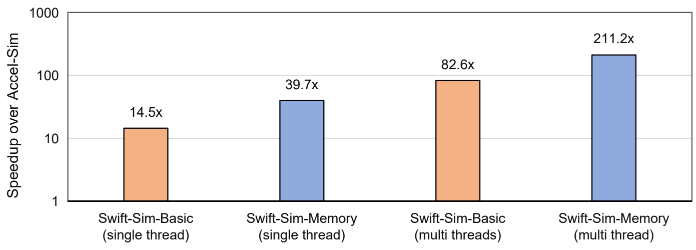

Fig. 5. Contribution analysis of speedup for Swift-Sim over Accel-Sim.

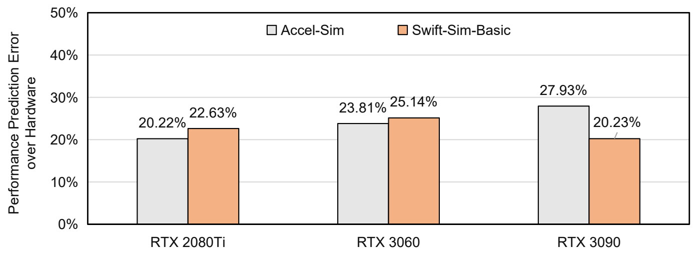

Fig. 6. Performance prediction errors of Swift-Sim-Basic and Accel-Sim on three GPUs.

We further analyze the speedup achieved by various factors, as shown in Figure 5. For all applications, Swift-Sim-Basic achieves an average speedup of 14.5x compared to Accel-Sim under single-thread execution. Swift-Sim-Memory provides an additional 2.7x speedup over Swift-Sim-Basic by simplifying memory access modeling, resulting in a total speedup of 39.7x over Accel-Sim under single-thread execution. In addition, the highly modular design of our simulation framework makes it easy to run parallel simulations on a single machine. This leads to further speedup gains of about 5x for both Swift-Sim-Basic and Swift-Sim-Memory, resulting in total speedups of 82.6x and 211.2x respectively compared to Accel-Sim. With the above speedups, Swift-Sim can complete simulations in a few minutes which would take Accel-Sim over one day to complete.

3) Design Space Explorations: We use the Swift-Sim-Basic simulator to evaluate the variation in the modeling errors of the Swift-Sim framework on different GPU architectures. As shown in Figure 6, we compare the errors between the application execution cycles predicted by Accel-Sim and Swift-Sim-Basic with the actual hardware cycles across various NVIDIA architectures. On the NVIDIA RTX 3060 GPU, the predicted cycles from Swift-Sim-Basic have a mean error of 25.14%, while Accel-Sim's have a mean error of 23.81%. On the NVIDIA RTX 3090 GPU, our predicted cycles have a mean error of 20.23%, while Accel-Sim's have a mean error of 27.93%. We find that Accel-Sim performs poorly on applications such as BFS, ADI, and LU when modeling the NVIDIA RTX 3090 GPU. Upon analyzing Accel-Sim's results for these applications, we observe a number of cache reservation failures. We hypothesize that the discrepancies in predicted execution cycles are due to differences between Accel-Sim's cache policy modeling and the actual hardware, leading to greater errors in these applications.

Due to unique disclosed hardware parameters in different GPU architectures, the error of the GPU performance simulator varies a bit from architecture to architecture. We observe that Swift-Sim maintains high accuracy across GPU architectures with significant variations in GPU configuration. We believe Swift-Sim performs well in exploring various GPU architectures.

V. Conclusion

In this paper, we discuss existing GPU performance simulators that are still cumbersome and slow to run. We propose a modular and hybrid GPU simulation framework, called Swift-Sim. Swift-Sim adopts a highly modular design, with each component simulated by an independent module. Architects can choose the modeling methods for these modules based on the requirements of the architectural design space and hence take full advantage of the respective strengths of cycle-accurate simulation and analytical models. We experimentally demonstrate that simulators with different levels of simplification based on the Swift-Sim framework achieve geometric mean speedups of 82.6x and 211.2x, respectively, compared to Accel-Sim. Swift-Sim is accurate, fast, and suitable for architectural exploration. The source code of Swift-Sim is available at https://github.com/xurongxiang/Swift-GPUSim.

References

[1] S. Zhang, M. Naderan-Tahan, M. Jahre, and L. Eeckhout, "Sac: Sharing-aware caching in multi-chip gpus," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[2] P. Dalmia, R. Mahapatra, and M. D. Sinclair, "Only buffer when you need to: Reducing on-chip gpu traffic with reconfigurable local atomic buffers," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 676–691.

[3] X. Zhao, A. Adileh, Z. Yu, Z. Wang, A. Jaleel, and L. Eeckhout, "Adaptive memory-side last-level gpu caching," in *Proceedings of the 46th international symposium on computer architecture*, 2019, pp. 411–423.

[4] S. Darabi, E. Yousefzadeh-Asl-Miandoab, N. Akbarzadeh, H. Falahati, P. Lotfi-Kamran, M. Sadrosadati, and H. Sarbazi-Azad, "Osm: Off-chip shared memory for gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3415–3429, 2022.

[5] 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.

[6] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012, pp. 335–344.

[7] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills gpusimpow: A gpgpu power simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 97–106.

[8] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, 2014.

[9] 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.

[10] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 738–753.

[11] J. Lee, Y. Ha, S. Lee, J. Woo, J. Lee, H. Jang, and Y. Kim, "Gcom: a detailed gpu core model for accurate analytical modeling of modern gpus," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 424–436.

[12] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 3, pp. 1–25, 2014.

[13] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.

[14] O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, N. Chatterjee, N. Jiang, and D. Nellans, "Need for speed: Experiences building a trustworthy system-level gpu simulator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 868–880.

[15] Y. Li, Y. Sun, and A. Jog, "Path forward beyond simulators: Fast and accurate gpu execution time prediction for dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 380–394.

[16] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled gpu workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 724–737.

[17] Y. Oh, K. Kim, M. K. Yoon, J. H. Park, Y. Park, M. Annavaram, and W. W. Ro, "Adaptive cooperation of prefetching and warp scheduling on gpus," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 609–616, 2018.

[18] L. Wang, M. Jahre, A. Adileho, and L. Eeckhout, "Mdm: The gpu memory divergence model," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1009–1021.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[20] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[21] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 74–85.

[22] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Multi-processor system-on-chip design space exploration based on multi-level modeling techniques," in *2009 International Symposium on Systems, Architectures, Modeling, and Simulation*. IEEE, 2009, pp. 118–124.

[23] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2013, pp. 86–96.

[24] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "Dramsim: a memory system simulator," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 100–107, 2005.

[25] S. J. Wilton and N. P. Jouppi, "Cacti: An enhanced cache access and cycle time model," *IEEE Journal of solid-state circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[26] J.-C. Huang, J. H. Lee, H. Kim, and H.-H. S. Lee, "Gpumech: Gpu performance modeling technique based on interval analysis," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 268–279.

[27] C. Nugteren, G.-J. Van den Braak, H. Corporaal, and H. Bal, "A detailed gpu cache model based on reuse distance theory," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 37–48.

[28] J.-C. Huang, L. Nai, H. Kim, and H.-H. S. Lee, "Tbpoint: Reducing simulation time for large-scale gpgpu kernels," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 437–446.

[29] C. Liu, Y. Sun, and T. E. Carlson, "Photon: A fine-grained sampled simulation methodology for gpu workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1227–1241.

[30] H. SeyyedAghaei, M. Naderan-Tahan, and L. Eeckhout, "Gpu scale-model simulation," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1125–1140.

[31] 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.

[32] 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.

[33] NVIDIA, "Nvidia graphics cards." https://www.nvidia.com/en-us/geforce/graphics-cards/, 2024.

[34] 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.

[35] L.-N. Pouchet and S. Grauer-Gray, "Polybench: The polyhedral benchmark suite." http://web.cs.ucla.edu/\~pouchet/software/polybench/, 2012.

[36] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 260–269.

[37] A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 137–138.

[38] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 185–195.