Need for Speed: Experiences Building a Trustworthy System-Level GPU Simulator

Industry Track Paper

Oreste Villa* Daniel Lustig* Zi Yan*
NVIDIA *NVIDIA* *NVIDIA*
ovilla@nvidia.com dlustig@nvidia.com ziy@nvidia.com

Evgeny Bolotin
NVIDIA
ebolotin@nvidia.com

Yaosheng Fu
NVIDIA
yfu@nvidia.com

Niladrish Chatterjee
NVIDIA
nchatterjee@nvidia.com

Nan Jiang
NVIDIA
tedj@nvidia.com

David Nellans
NVIDIA
dnellans@nvidia.com

***Abstract*—The demands of high-performance computing (HPC) and machine learning (ML) workloads have resulted in the rapid architectural evolution of GPUs over the last decade. The growing memory footprint and diversity of data types in these workloads has required GPUs to embrace micro-architectural heterogeneity and increased memory system sophistication to scale performance. Effective simulation of new architectural features early in the design cycle enables quick and effective exploration of design trade-offs across this increasingly diverse set of workloads. This work provides a retrospective on the design and development of NVArchSim (NVAS), an architectural simulator used within NVIDIA to design and evaluate features that are difficult to appraise using other methodologies due to workload type, size, complexity, or lack of modeling flexibility. We argue that overly precise and/or overly slow architectural models hamper an architect's ability to evaluate new features within a reasonable time frame, hurting productivity. Because of its speed, NVAS is being used to trace and evaluate hundreds of HPC and state-of-the-art ML workloads on single-GPU or multi-GPU systems. By adding component fidelity only when necessary to improve system-level modeling accuracy, NVAS delivers simulation speed orders of magnitude higher than most publicly available GPU simulators while retaining high levels of accuracy and simulation flexibility. Building trustworthy high-level simulation platforms is a difficult exercise in balance and compromise; we share our experiences to help and encourage those in academia who take on the challenge of building GPU simulation platforms.**

***Index Terms*—GPU, Simulation, Performance Modeling, System Simulation**

I. Introduction

GPUs are the dominant platform for most data-parallel workloads in the high-performance computing (HPC), machine learning (ML), and data analytics domains. As dataset sizes and the desire to simulate more complex systems grow, both the individual size and aggregate number of GPUs being applied to these complex problems is increasing. To continue scaling performance GPUs are embracing increased complexity in their individual processing pipelines, while also increasing heterogeneity across the chip. GPUs continue to be manufactured near the reticle limit, with academic and industry groups proposing system-level designs to allow close coupling of multi-chip modules and aggregation of discrete GPUs across the data center [1], [2], [4], [15], [37], [50], [61], [63]. GPUs are also rapidly evolving, with recent generations featuring a wide variety of architectural enhancements including new data types [44], tighter integration with CPUs and their coherence protocols [8], [24], [31], [45], data center level GPU-optimized interconnects [43], and complex data transformations [42].

To support the needs of emerging workloads, GPU architects require design tools that can quickly evaluate the impact of aggressive new features across an increasingly diverse set of applications. Quickly exploring the design space of future GPUs is necessary for fast product iteration in a rapidly moving industry. Ideally, architectural level simulators should balance three primary goals during their development. First, simulators need to be both accurate and trustworthy—two distinct properties. While accuracy can be measured in comparison to existing silicon, trustworthiness reflects the ability of the simulator to model novel ideas not yet baked into silicon. Second, simulators must be able to run a wide variety of workloads including new and emerging workloads which may require new methodologies to capture their representative behaviors. Third, simulation turnaround must be fast. If the turnaround time for simulation is too slow, architects will often turn to other (possibly less appropriate) tools to make their engineering decisions.

With these three goals in mind, this work presents a retrospective of the choices made when implementing the NVIDIA Architectural Simulator (NVArchSim or NVAS), our in-house hybrid trace- and execution-driven single- and multi-GPU simulator. NVAS is one of many simulation tools used within NVIDIA and has been designed to fill a niche for which existing tools are either too slow, too targeted at a specific GPU component, or too difficult to employ for system-level studies.

* These authors contribute equally.

No two simulation platforms provide identical answers and even within NVIDIA no single tool provides a golden answer. NVAS is often the first tool used for architectural exploration, but rarely is it the last (or only) tool used for feature evaluation. It provides a balance of high-speed execution for rapid design iteration and sufficient accuracy that feature-level studies can occur quickly before being adapted, refined, and verified in more detailed (and slower) product-specific simulations.

NVAS was motivated by the need to capture system-level data reuse and communication patterns of applications ranging from small microbenchmarks to new large deep learning training and inference workloads containing billions of GPU warp instructions. To satisfy this need, we employ a number of strategies. 1) We employ a hybrid trace- and execution-driven simulation methodology. For GPU simulation, trace-driven execution tends to be faster than execution-driven simulation [29]. A partial execution methodology allows NVAS to overcome many of the limitations of trace-only driven simulation, such as value-dependent instruction execution, without sacrificing its speed benefits. 2) We utilize only *loose cycle accuracy* wherever this lack of fidelity does not substantially harm overall system-level simulation accuracy. To justify this approach, we provide three studies examining the simulation trade-offs when implementing differing levels of detail for SM instruction dependency tracking, on-chip interconnect modeling, and off-chip DRAM fidelity. 3) We emphasize the need to accurately capture trends across hundreds of GPU workloads rather than achieving ultra-high correlation for just a small handful. To demonstrate the effectiveness of this approach, we present NVAS simulation validation data for over 900 traces obtained from workloads run on single and multi-GPU systems with simulation turnaround times ranging a few seconds to up to 4 days.

Simulation development is not glamorous but it is necessary. NVAS has been developed over the last 7 years including a ground up re-write to implement the current methodological approach, which supports the new and shifting priorities within NVIDIA. While it is difficult to transition, abandoning old tools for new approaches is part of the natural evolution of chip, system, and data center modeling. Developing the right tool for the job can save countless engineering hours, far exceeding the development time invested. We hope that by providing increased visibility into our own development process, use cases, and evaluation metrics, our experience can help guide the architecture community as it continues to perform high quality GPU research.

II. Background and Motivation

Simulation is the primary tool within industrial and research communities to both project the performance of new designs and root cause performance bottlenecks in existing hardware. As a result, high simulation accuracy in relation to one or more physical design points is a key metric when implementing a simulation environment. In addition to accuracy, simulation platforms need to be stress tested across parameter variation

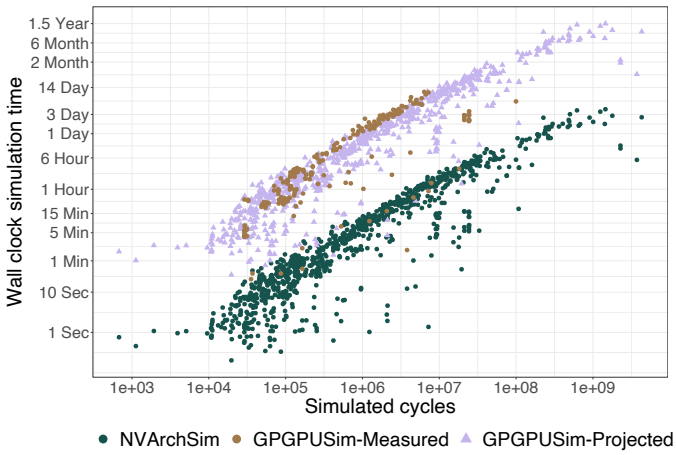


Fig. 1: Wall clock time required to simulate 955 benchmarks on NVAS. Where possible, identical workloads were simulated on GPGPU-Sim 4.0, with the remaining times projected based on average slowdown.

to provide confidence that the simulation is not just accurate but trustworthy at projecting performance.

For example, naturally skeptical engineers will consider simulation tools validated on too few workloads (but achieving high accuracy for those workloads) as likely to have little predictive value when considering new applications or invasive architectural changes. Similarly, if fudge factors that are not rooted in the architectural design are used to compensate for miscorrelation or simulator skew, the skeptical engineer will view them as obscuring real architectural bottlenecks and reducing the trustworthiness of the simulation platform. As a result, we have found that during simulator development we must be willing to *accept and embrace that some degree of inaccuracy is inevitable*. Focusing on achieving correct trends across hundreds of diverse workloads provides more predictive power for forward-looking studies performed on NVAS.

Finally, simulators can be both accurate and trustworthy, but if they do not achieve high simulation speed, they are unlikely to be adopted because they cannot quickly simulate workloads of interest. Figure 1 provides an overview of the wall clock time required to simulate 955 benchmarks of interest on NVAS. Most of these workloads can be simulated in under a few hours, with only the longest requiring a multi-day simulation execution. This allows NVAS users to explore wider design spaces across a large and varied set of workloads. To illustrate the importance of simulation speed, with the help of the GPGPUSim team we were able to get a subset of identical workloads executing on GPGPUSim v4.0 [29]. Depending on application type GPGPUSim can be between 40-250× slower than NVAS while simulating identical workloads. This leads to infeasible simulation duration for many of our workloads, later shown in Table II, that would take months or even years to complete in traditional GPU simulators.

A. Cycle Accurate Simulation

The guiding principles for what we call "loose cycle accuracy" in the development of NVAS are based on the

observation that GPUs are throughput-oriented computing devices. CPUs are very sensitive to memory system latency and per-instruction scheduling due to high single-threaded instruction level parallelism (ILP). This makes getting exact cycle duration for events important in achieving good silicon correlation with simulation platforms. Conversely, GPUs tend to be latency tolerant thanks to their heavy use of massive multithreading. We have observed that overall GPU simulator accuracy is negligibly affected by small inaccuracies in microarchitectural latency, and thus near-cycle accuracy when implementing most GPU structures is not required. Exploiting this latency tolerance allows us to approach GPU simulation philosophically as a complex model of dynamically interacting roofline components, rather than a CPU-like system in which tracking the detailed cycle-by-cycle progression of all components is necessary.

B. GPU Computing Beyond HPC

While GPU computing is now a mainstay of HPC, GPUs are also seeing widespread adoption for data center machine learning training and inference. New benchmarks such as Baidu's Deepbench [5] are the first step towards producing representative ML benchmarks, and ultimately architects must utilize realistic ML training that captures both inter and intra-layer reuse. MLPerf [35], [49] is the industry standard for benchmarking single and scale-out GPU configurations (or other non-GPU platforms) for machine learning performance. To successfully capture and model just a single iteration of many MLPerf workloads, GPU simulation platforms must be able to execute approximately 11 billion warp level instructions in a reasonable time, and must also capture communication and reuse patterns between kernels rather than relying on snippets plucked from individual kernels. As shown in Figure 1, this level of workload complexity and duration is likely to be intractable on highly detailed simulators.

Large ML workload training demands often cannot be satisfied by a single GPU and scaling performance by employing multiple GPUs is commonplace. This places data and kernel placement decisions of high-level ML frameworks on the critical path for overall application performance, making it difficult to distill out a single (or even multiple) key inner loop(s) as have historically been used to optimize HPC performance on GPUs. Going forward, scalable GPU performance is increasingly likely to depend on the co-design of software runtimes and GPU architecture to account for application features like task distribution across GPUs, memory placement and migration among GPUs, and GPU network interaction. As a result, we believe GPU simulation platforms must evolve their capabilities and speed to successfully model realistic multi-GPU workloads rather than focusing on small single GPU benchmarks which are inherently easier to simulate.

C. Simulating GPU Assembly versus High-Level ISAs

To optimize for performance many NVIDIA mission critical libraries such as cuDNN [41] and cuBLAS [40] rely on highly optimized GPU assembly (SASS) rather than higher level programming languages or Parallel Thread Execution ISA (PTX) based routines. These per-product tuned routines may outperform naive versions that are implemented in CUDA/C++ by as much as 50%, resulting in dramatically different hardware utilization and bottleneck analysis. It is critical that architects examine the routines that are used in production environments rather than pedagogical examples, or they risk making incorrect architectural optimizations in future products. Further, even for high-level written code prior work has also shown that HSA-level simulation is less accurate than GPU assembly-level simulation because of the layer of indirection caused by code finalization [21].

Despite these new challenges that are largely unique to GPUs (versus CPUs), they are not insurmountable. Focusing development effort to find solutions that overcome simulation speed, scale, and accuracy issues is difficult within an industry environment and is undoubtedly harder for those outside industry. We applaud academic research teams that do their best to make informed estimations about the behavior of propriety hardware with limited development resources. We now present specific choices made while developing NVAS to help advance the community's GPU simulation best practices and allow them to focus on the GPU architectural issues that matter most while avoiding wasted effort on those that do not.

III. Simulator Design Choices in NVArchSim

NVArchSim is a hybrid trace- and execution-driven, loosely cycle accurate GPU simulator equipped with swappable components. These design features allow NVAS to meet our overarching goals; namely, achieving fast simulation speed without sacrificing accuracy and trustworthiness.

A. Architecture Model and Fidelity

As shown in Figure 2, a NVAS simulation implements a system level model composed of one or more GPUs that are driven via a high-level CPU model. The CPU model performs three main functions: 1) It executes the flow of CUDA driver APIs. 2) It interacts with a lightweight version of the Unified Memory (UM) driver for page allocation, release, and other memory management functions, if necessary. 3) It dispatches work (typically CUDA kernel calls or DMA memory copies) to the GPU(s). Each core in the CPU model is associated with a single simulated application, and NVAS allows concurrent execution of multiple simulated applications within a system.

Each GPU component contains a hardware task engine that receives and processes CUDA API calls from the CPU and schedules the resulting cooperative thread arrays (CTAs) on the various streaming multiprocessors (SMs), which ultimately are responsible for executing the warp-level instructions. Each GPU contains a hierarchy of graphics processing clusters (GPCs), each of which has multiple texture processing clusters (TPCs) containing multiple SMs and L1 caches. Additionally, there is both a private and shared TLB hierarchy assisted by a hardware page table walker (PTW). Each GPC is connected via an on-chip interconnect to multiple slices of a shared physically addressed L2 cache that typically is the GPU's point of coherence for physical addresses.

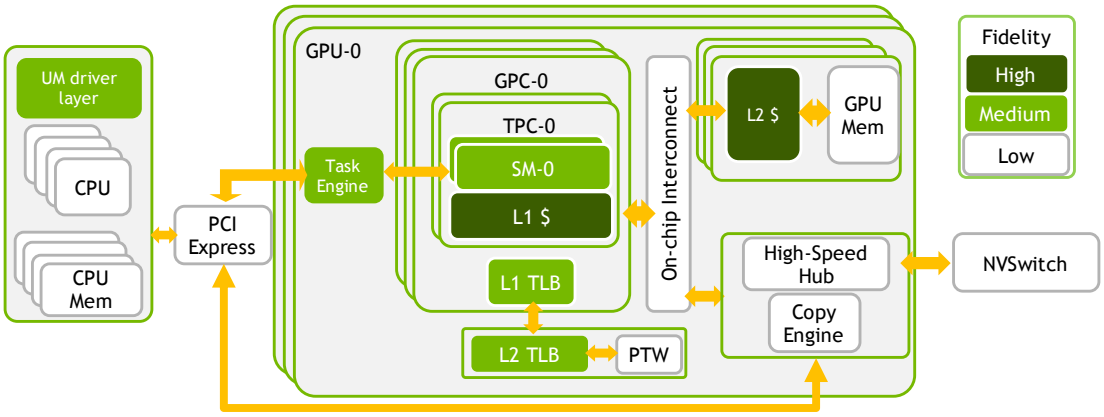


Fig. 2: Overview of NVArchSim with all major simulator components illustrated including fidelity.

All components in the entire system are interconnected using abstract channels that model adjustable bandwidth and latency. In multi-GPU scenarios each GPU can be connected to others either via PCIe, point-to-point NVLink connections, or via a unique topology of NVLink plus NVSwitch components. Users specify the number of instances, attributes of each component, and the bandwidth and latency of each channel using configuration files.

One unique feature of NVAS is that these individual architectural components all implement very different levels of fidelity in terms of how faithfully they model the corresponding hardware components found in production GPUs. As Figure 2 illustrates, we intentionally maintain a low level of detail in many components. The NVArchSim development approach is to add increasing detail to components *only when that detail makes a significant improvement to system-level performance accuracy, and simulation speed is not substantially affected*. For example, the SM pipelines model the right high-level architecture and throughput as controlled via simulator knobs, but not the detailed microarchitecture that determines those knobs. The caches are modeled with tags and data banks of sizes and associativity matching real hardware, with customization for atomics and other features as is needed to reflect their actual performance. On the other hand, our on-chip interconnect model is nearly trivial. Most channels are simple latency plus bandwidth models, but a few incorporate more specifics regarding packet overhead wherever that overhead is an important factor. In our experience, minimizing per-component detail also enables much faster modification and development of new architectural components and interconnects, particularly by non-expert users.

The memory-related components in NVAS, such as caches, tend to have increased fidelity compared to most other components because well-optimized GPU workloads tend to be both memory-intensive and latency-insensitive. NVAS' SM models contain a medium level of accuracy: enough to capture the throughput and latency of each instruction, and to properly maintain instruction dependencies. Studies that require substantially more SM microarchitectural detail are typically deferred to other tools within the company. Having flexibility with the SM model allows us to, for example, easily model the impact of new instructions or in-core accelerators such as NVIDIA Tensor Cores [42] that dramatically change instruction throughput.

Components that have not yet proven to impose a significant system-level roofline effect in applications are implemented with only the minimal detail required to allow functional operation of the simulator. Implementing unnecessary detail within components not only slows down simulations due to needing to track and execute additional microarchitectural state, but it also raises the risk that a specific implementation choice unintentionally becomes a roofline that does not correspond to any real hardware (e.g., improperly sizing some low-level buffer within a component). Section V later provides several examples of fidelity, accuracy, and speed trade-offs we have explored in NVAS.

B. Simulation Engine

While Figure 2 shows the high-level conceptual components in NVAS, simulations typically have thousands of hierarchically organized sub-components, plus channels interconnecting those components. Ticking these components each cycle is too expensive, so like other event-driven simulators [7] NVAS modules are only ticked on the cycles in which they have work to perform. The simulation starts with the CPU component reading the first CUDA driver API event from a trace and it naturally terminates when there are no more events to simulate. The simulator is able to restrict areas of interests to a set of CUDA driver API events or even warps, in cases where a user only cares about a specific portion of the workload.

NVArchSim utilizes a single threaded, but highly optimized, event-based execution engine capable of dispatching several millions of events per second. Although recent simulators [52], [55] show that multithreading can deliver speedups in the range of 2–3×, the NVArchSim engine remains single-threaded to reduce code complexity, facilitate easy debugging, and ensure reproducible simulations. High simulation speed is achieved primarily through judicious component fidelity. Each component in a NVAS simulation enqueues events with the simulation engine to be executed some number of cycles in the future. Examples of these events may be a packet being

released from a channel to be received from another component, or an instruction completion based on its configuration defined static average latency.

Dynamic system latencies are modeled using queues within virtual or physical channels. NVAS policy explicitly forbids components from having any form of "retry logic" that proactively queues events in the future for retry without being signaled by another component first. Allowing retry logic can result in an explosion of unnecessary simulation-slowing events, which can be avoided by requiring triggering events instead. For example, a component should not continually try to queue a packet onto a channel each cycle, only to find the channel remains full. Instead, the channel component must signal a not full condition to the enqueuing component before it is allowed to retry. Failure in queuing may still result, for instance another packet occupies that channel before the component queues its own packet, but the number of fruitless attempts (and simulation overhead) is drastically reduced.

C. Tracing Infrastructure

To achieve high simulation throughput in systems composed of thousands of SIMT pipelines we believe functional execution of all ISA instructions is infeasible for large workloads. GPU instruction set architectures are becoming increasingly complex both semantically and in the number of distinct instructions they implement as products evolve to support new workload classes. Fortunately, most GPU programs do not make complex control flow decisions based on the state of dynamically generated data. This programmatic determinism allows the vast majority of the instructions to be traced on existing GPU hardware and then "replayed" on the simulated architectural models by accounting for their throughput, latency, hazard dependencies, and interactions with the memory system. Thus, the NVAS application trace format mostly captures static instruction decoding information (as in the binary), a sequence of dynamically executed instructions per warp and the list of the memory addresses accessed those instructions (if any). The trace does not record instruction timestamps or other event timing information, nor does it record any architectural state not explicitly required for execution. For example, while hazard dependencies are encoded in the trace and are part of the semantics of the application, latency and throughput are architectural parameters that can be modified in the simulated target architecture.

Application traces are collected at the SASS (GPU assembly) level using NVTracer, a binary instrumentation tool developed on top of NVBit [57], and stored in highly compressed files as shown in Figure 3. Each trace, representing a CUDA application, contains one stream recording the high-level CUDA driver API invocations and additional streams recording the SASS execution of each logical warp in the application. A workload that executes a single kernel launch with 100 thread blocks, composed of 256 threads each, will result in a NVAS trace containing 1 CUDA driver API stream with a `cudaLaunchKernel` event along with the streams

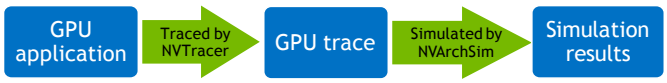


Fig. 3: The baseline trace-based NVAS workflow.

Suite	All	Determ-inistic	Simple Spinloops	Complex Spinloops	Other Non-det.
Microbench	58	58	0	0	0
Rodinia	17	17	0	0	0
Polybench	10	10	0	0	0
Industry HPC	204	187	5	8	4
CUBLAS	35	31	4	0	0
CUTLASS	410	410	0	0	0
Deepbench	548	218	330	0	0
MLPerf	24	8	8	0	8

TABLE I: Sources of non-determinism ("non-det.") in our library. Workloads with more than one form of non-determinism are placed into the rightmost applicable category.

of SASS instructions (and virtual addresses accessed) executed by each of the 800 logical warps[1].

Several layers of compression in the format, based on similarities of executed instruction sequences and addresses (within warps and kernel invocations), combined with a brute force binary compression, enable very compact traces with a typical average size of less than 1 Byte for each warp level instruction (including all virtual memory addresses if present). On top of this, an indexing layer allows fast traversal (i.e., *O(1)*) of individual warp instruction streams, independent of other executing warps. This is necessary because while the order of instructions within a logical warp will not change during replay, the relative order in which independent warps make forward progress is based on the properties of the simulated GPU. For example, a GPU containing 100 SMs, each with 32 warp slots, will execute the stream of warps in a grossly different order than another GPU comprising a single SM with only 8 warp slots.

D. Supporting Hybrid Trace and Execution Simulation

One well established issue with trace-based simulation is that non-deterministic workloads cannot be accurately modeled. There can be many reasons why non-deterministic GPU executions may differ from run to run, including (but not limited to) producer/consumer synchronization, in-memory spinloops, etc. For all these cases, capturing one particular execution of an application may not be representative of the application behavior across many executions.

Classifying GPU Workload Non-Determinism: Table I summarizes the causes of non-determinism we observe in our application library into several categories. While most applications remain deterministic, some application suites have a

[1]A 256-thread thread block consists of eight 32-thread warps, so 100 thread blocks contain 800 logical warps.

large number of workloads that contain what we call "simple" spinloops (e.g., patterns consisting of an easily identifiable "load, compare, branch" sequence). A much smaller set of applications contain "complex" spinloops that perform extra integer or floating-point calculations within the loop. Finally, a few applications contain enough sources of non-determinism such that non-trivial portions of the application are executed differently across successive executions on real GPUs. Because applications can be composed of many kernels (of which some may be deterministic and others non-deterministic), we classify applications into each category if at least one of its invoked kernels falls in that category. For instance, MLPerf has 8 traces that are classified as "Other Non-Determinism" although they primarily contain just simple spinloops.

With an understanding of the sources of non-determinism across a wide range of GPU applications we chose to add support for simple spinloop detection and execution within NVAS as shown in Figure 4. The use of in memory spinloops is becoming more common on GPUs typically to support inter-CTA synchronization patterns, such as reductions. In addition, application suites such as LoneStarGPU [9] and HeteroSync [54] are specifically dedicated to studying more advanced forms of synchronization patterns and irregularities. As the importance of these applications grows in the marketplace, we will have to re-evaluate, extend, and perhaps alter the NVAS approach to for handling complex non-deterministic applications.

Functional Execution of Spinloops Within a Trace: Spinloop non-determinism can have a large impact on simulated GPU performance because of the overheads incurred during trace collection on real GPUs. Specifically, the relative number of program instructions spent executing the spinloop can increase many orders of magnitude due to a consuming thread waiting on a producer thread that is executing additional instructions, e.g., due to instrumentation overhead during the trace generation process. Consequently, we have considered two ways for NVAS to support applications containing spinloops.

The first is to simply identify and fast-forward simulations through these overly repetitive spinloop sequences. We have found this technique a useful and accurate representation of hardware execution when simulating applications where the spinloop itself is used strictly as a performance optimization, designed to align the execution phases of different warps to avoid hardware resource contention. In these cases, despite changing the trace SASS semantics, the approach is sufficiently representative from the broad application perspective.

The second and more general-purpose approach of *partial functional execution* allows NVAS to simulate spinloops without violating functional correctness. In brief, this involves building a small GPU execution engine that can functionally execute a limited subset (less than 1%) of GPU instructions within NVAS. This approach requires two primary tasks: identifying the instructions that require simulation and supporting their execution in the simulator.

To identify the set of instructions in a GPU execution that need to be functionally executed, we perform the following steps. First, we instrument the binary and run the application multiple times on real hardware and for each execution generate a histogram of the executed instructions counts per program counter. An example is shown in Figure 5 where for each instruction in the disassembled binary, the number of times that instruction was executed in that particular program execution, is annotated on the right. If the instruction counts are identical across multiple executions, we consider them deterministic[2]. If there are differences in instruction count, the histograms identify the subset of instructions that are responsible for the workload nondeterminism.

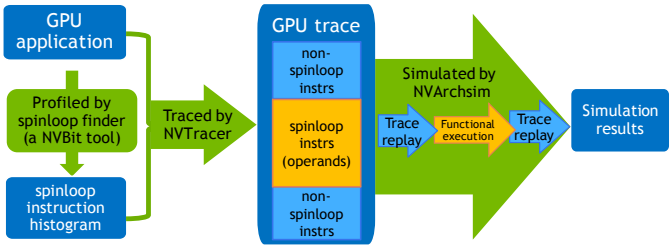


Fig. 4: The NVAS partial execution workflow where operand values are recorded in the trace during non-deterministic regions, then used for functional execution during simulation.

After we have identified one or more of these regions, we then perform a trace analysis to identify other instructions that access the same memory address(es) as the memory instructions contained within the spinloop(s). This is necessary to ensure that we can capture the production of the value the spinloop is waiting on, even if the producer instruction is executed a deterministic number of times. After this analysis is completed, we have identified all the instructions that need to be functional emulated. With this information in hand, we then retrace the application while also recording all register and predicate operands for the identified instructions, including the initial state of the memory locations accessed by them.

To execute the functional instructions recorded in NVAS' operand-augmented trace, we made the following changes to NVAS. First, we enhanced the SM to support the functional execution of a very limited number of instructions. Second, we extended the memory hierarchy capability to allow the passing of values contained at labeled memory locations throughout the simulated memory hierarchy. Finally, we modified the trace reading component of NVAS to detect the presence of instructions that need to be functionally simulated and to initialize the appropriate source register and predicate values based on the values listed in the augmented trace file.

When NVAS encounters an instruction in the trace that is marked as functional, the new functional execution engine begins execution. This functional execution continues until the thread returns to executing a non-functional instruction, at which point the traditional NVAS trace-based execution continues its work. If the program counter and active thread

[2]In theory this approach can produce false negatives, but for a large enough number of executions, we found this not to be a concern in practice.

Instructions	Exec. 1	Exec. 2
...	...	...
1 MOV R10, RZ	432	432
2 ISETP.NE.AND P5, PT, R10, R12, P5	208822	208304
3 @P5 LDG.E.STRONG.GPU R10, [R102]	208822	208304
4 @P5 BRA.U 2	208822	208304
5 ISETP.LT.AND P5, PT, R91, R80, P1	432	432
...	...	...
6 @P5 MEMBAR.GPU	432	432
7 @P5 STG.E.STRONG.GPU [R100], R12	432	432
8 EXIT	432	432
...	...	...

Fig. 5: Histograms showing the number of times each instruction in a particular application is executed. The instructions with different execution counts (2–4, highlighted in yellow) identify a clear spinloop. Analysis of the trace virtual addresses then shows that instances of 7 access the same memory locations as instances of 3 (highlighted in green), even though the registers differ. Therefore, 7 is the producer of the value on which the spinloop is waiting and hence must also be functionally executed, even though its instruction count does not vary across executions.

mask of this first non-functional instruction match the exit state of the functional simulation, then no unsupported types of non-deterministic execution have occurred. If this condition is not met, however, NVAS must abort and report a new form of unsupported non-determinism has been identified. In these cases, simulator developers must examine the trace to investigate if NVAS can be augmented to support the non-determinism, or if we will choose not to support this trace within the library. Thus far, we have experienced that a very small number of functional execution instructions are necessary to support the vast majority of the applications which exhibit non-deterministic execution via spinloops.

IV. NVAS Accuracy and Trustworthiness

We evaluate NVAS on the wide range of benchmarks shown in Table II. This suite includes commonly used academic benchmark suites such as Rodinia [13], [14] and Polybench [19], internal microbenchmarks and HPC workload traces, and modern machine learning workloads such as CUBLAS routines [40], CUTLASS [39], Deepbench [5], and MLPerf v0.6 inference and training [35], [49]. For MLPerf, we use the spinloop simulation technique discussed in Section III-D. As mentioned earlier, we continue to actively add increased support for other non-deterministic workloads on a case-by-case basis. We validate the results against the NVIDIA Tesla GV100 SXM2 GPU described in Table III.

Our workloads range from a six-instruction "hello, world!" microbenchmark to large applications with tens of billions of instructions. Applications from different suites tend to produce very different SM instruction mixes, thread divergence patterns, memory access patterns (e.g., fully coalesced, gather/scatter, random access, or mixtures thereof), different memory access sizes, and so on. As such, applications from different suites tend to stress different sets of rooflines in our architecture model. The more diversity we add, the more confidence we have that our GPU model faithful represents all the major rooflines within a real GPU microarchitecture. No small subset of the suite can easily achieve this objective. This led us to conclude that establishing trust within a general-purpose GPU simulator indeed requires evaluating hundreds of diverse workloads, rather than just a handful.

We evaluate all benchmarks end-to-end and collect the total execution time of all GPU kernels. We then compare this total to the corresponding number collected from real hardware using standard NVIDIA GPU profiling tools. Like many other simulator papers, we present mean absolute error (MAE), Pearson correlation, and simulation speed statistics. However, following that we also present some additional sensitivity studies that we used to build our own confidence in the simulator's predictive power.

A. Diagnosing Sources of Simulator Inaccuracy

One widely used metric to evaluate a hardware simulator is to compare its performance predictions to real hardware using microbenchmarks intended to stress specific microarchitectural components, such as those shown in Figure 6. While it may come as a surprise that a simulator developed with access to internal product details does not match silicon performance nearly exactly, we contend that highly correlated microbenchmarks do not necessarily lead to highly accurate simulations of large workloads and are often over emphasized during simulator development.

For example, NVAS strongly prefers to use design specification parameters within its individual component models rather than fine-tuning simulator knobs through an empirical or black box process to improve microbenchmark correlation. While we could choose to tune simulation parameters to match silicon microbenchmarking performance nearly exactly, we have not chosen to do this. We believe that using the correct architectural parameters without change vastly improves our simulator development and debugging process. Specifically, by leaving these parameters at their nominal architectural values we ensure that knobs changes made to improve microbenchmark correlation are not merely masking flaws in other portions of the simulation model. Simulation developers must remember that the end goal of simulation tools is not to achieve high perceived accuracy for a given configuration (or generation), but instead to maximize the predictive value simulations will deliver when considering new architectural features.

Furthermore, some minor discrepancies between NVAS microbenchmark results and silicon performance are by design. For example, NVAS chose to make little distinction between the GPU's shared memory scratchpad and L1 cache, because these structures are (for the most part) physically merged in recent GPU generations. Because of slight access path and addressing issues, microbenchmarking often will reveal subtle performance discrepancies. However we contend that fairly often these small discrepancies tend to not propagate up into notable system level effects and choosing to live with microbenchmarking miscorrelation in exchange for reduced

Benchmark Set	Number of Benchmarks	Instruction Count (Min)	Instruction Count (Median)	Instruction Count (Average)	Instruction Count (Max)	Mean Absolute Error (MAE)	Pearson Correlation
Microbenchmarks	58	6	297,856	3,308,650	22,068,224	14%	0.995
Rodinia	17	324,656	321,775,425	11,221,288,859	78,524,397,760	15%	0.996
Polybench	10	6,378,048	416,959,760	1,070,392,228	5,438,720,000	12%	0.993
Industry HPC	187	260,580	49,182,512	2,439,293,716	54,550,132,311	19%	0.935
CUBLAS	31	4,334,080	102,914,560	355,912,387	1,695,767,040	10%	0.999
CUTLASS	410	50,336	29,077,952	482,594,905	11,618,640,000	19%	0.988
Deepbench	218	80,480	14,217,117	276,199,106	3,752,390,356	14%	0.996
MLPerf	24	18,176,675	3,909,825,077	11,952,352,871	52,202,026,615	13%	0.969
All	955	6	27,158,732	1,270,964,114	78,524,397,760	17%	0.953

TABLE II: All benchmark sets used for NVArchSim with minimum, maximum, average, and median of the numbers of warp level instructions from each benchmark set and mean absolute errors (MAE) of simulation correlation.

Architecture	Volta
Process node	12nm
SM count	80
Shared memory/L1 cache per SM	128KB in total (up to 96KB shmem)
Total L2 cache	6MB
DRAM	16GB, 897 GB/s

TABLE III: NVIDIA Tesla GV100 GPU SXM2 specification used to correlate NVAS against real silicon.

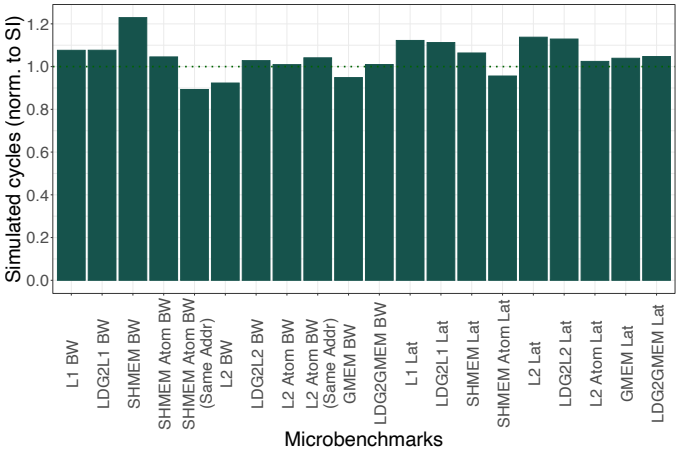


Fig. 6: NVAS achieves good accuracy for a wide range of microbenchmarks (subset shown here) by using known architectural parameters even if correlation could be improved through hand tuning of parameters.

code and simulation complexity is the appropriate simulator trade-off.

Most recent GPU simulators are choosing to report both the Pearson correlation and mean absolute error (MAE) across a range of benchmarks, as we show in Table II. Improving a simulator's MAE will improve its overall correlation to silicon; however, the Pearson correlation can be misleading when too few benchmarks are used, they do not span large differences in hardware cycle count, or there is systematic performance skew within the simulator. Instead, we evaluate NVAS through the lens of a wide range of benchmarks that are sorted by their relative error, as shown in Figure 7. This sorting provides an easy-to-understand snapshot of simulator accuracy (how flat is the s-curve), systematic skew (how shifted from the 1.0 midpoint is the s-curve), and which applications should not be trusted to provide high confidence results when using NVAS.

Table II shows that NVAS' correlation coefficient can vary from 0.935 to 0.999 across different benchmark suites and the mean absolute error ranges from 0.104 to 0.193. Yet clearly there are significant outliers from these averages and even from the 90th percentile of workloads. We typically approach NVAS improvements by hunting these outliers one by one. By looking at these individual benchmarks in isolation and comparing the simulator performance to real hardware, we often find a component area that is modeled with insufficient fidelity or heavily utilizes a corner case performance optimization within the GPU's microarchitecture. When fixing and improving these components improves NVAS' overall correlation and/or predictive value (see Section IV-B below), without substantially hurting simulation speed, we do so. In other cases, if we root cause the issue to a more obscure microarchitectural detail or optimization that does not affect the overall trends, and/or when doing so would require substantial added complexity or slowdown, we have two options. We might choose to simply defer detailed study of such benchmarks to other internal tools that model the components in question in more detail, or we might swap in more detailed module implementations for specific studies (See Section V). This allows NVAS to by default remain focused on its primary goal of enabling fast system-level GPU architecture simulation.

Figure 8 shows NVAS' accuracy improvement achieved by using the hybrid trace and execution methodology described in Section III-D on the MLPerf v0.6 workloads [35], [49]. These 24 traces are split between training and inference at both large (L) and small (S) batch sizes which correspond to single GPU and scale out GPU configurations respectively. Figure 8 demonstrates that spinloop-unaware tracing and execution can result in a large correlation error with respect to real hardware. With spinloop correction, NVAS' accuracy for these many-kernel, full iteration benchmarks becomes very similar to NVAS' accuracy trends across other workload types. Partial functional execution for the spinloop instructions has negligible impact on simulation speed because the spinloops generally form only a small portion of the overall workload. So while this execution methodology will not solve general non-determinism issues in trace driven simulation, it fulfills a critical gap to enable fast simulation of large ML workloads

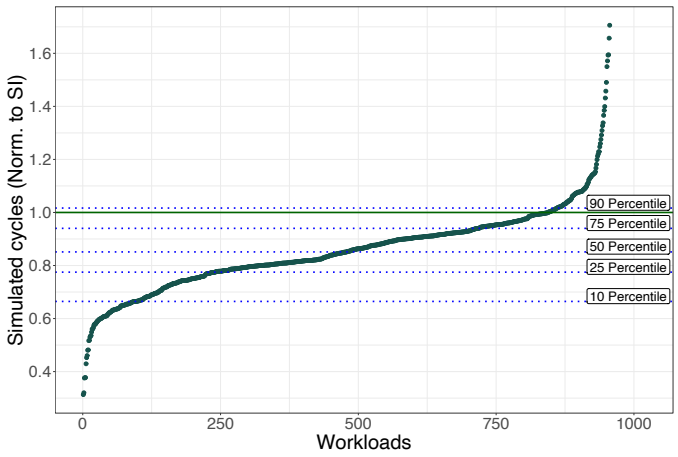


Fig. 7: In addition to correlation and mean average error, NVAS' accuracy is evaluated using sorted S-curves of simulation results.

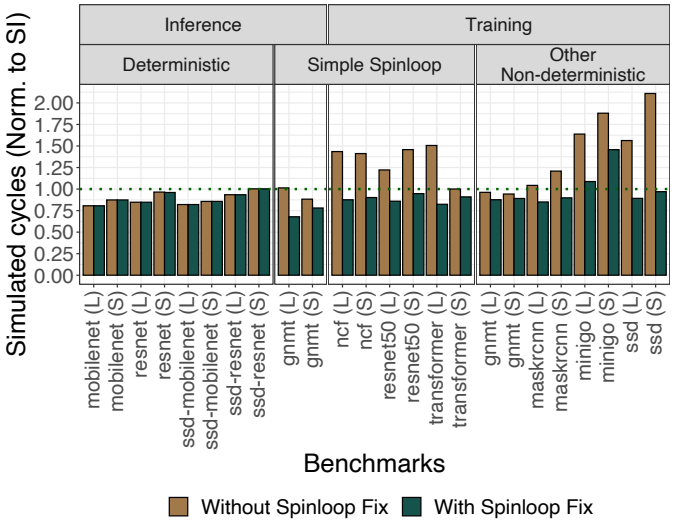


Fig. 8: Improvement in NVAS correlation on MLPerf workloads by implementing partial execution. Simple spinloops are the major source of instruction count mismatch, even when other non-determinism exists.

within NVAS.

B. Simulator Correlation vs Predictive Value

To maximize predictive value in a simulator, it is important to validate that simulated configurations are not overly constrained by superfluous details or overly aggressive microarchitectural rooflines. We do this in NVAS by making sure that changes in our simulation configuration accurately capture the behaviors and trends of real hardware whenever possible (and testable). A surprising number of simulation mistakes can be caught through basic testing such as sweeping frequencies (of core, interconnect, DRAM, etc), interconnect and I/O bandwidth and latency, and cache and memory capacities, in both silicon and the simulated model. During the development of NVAS we have seen more than one case in which simulated GPU performance moved in the opposite direction of the expectation when varying an architectural parameter, due to simulation bugs that were previously undetected.

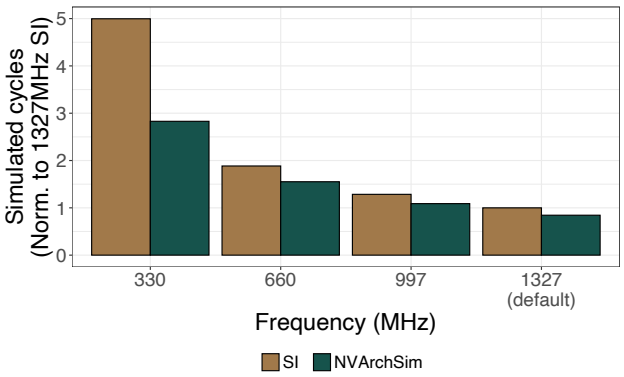
Figure 9a shows an example of this validation on NVAS by comparing expected performance while varying the SM frequency from 25% to 100% of the GPUs nominal value. Similarly, Figure 9b shows real silicon and simulated performance when restricting the number of SMs from 80 (100%) down to 20 (25%). For these configurations, NVAS scales performance at nearly the same rate as silicon maintain a slight underestimate on projected cycle count regardless of parameter scaling. For the 330 MHz point in Figure 9a where NVAS' performance project diverges from the broader trend, we were able to root cause this difference to implementation specific issues when aggressively underclocking this particular GPU configuration on real hardware. Excluding this data point, NVAS' projected performance remains at a nearly consistent (and overly optimistic) value of 16% and 20% faster than silicon, when sweeping frequency and SM count respectively. We encourage other simulator validation studies to perform similar experiments wherever possible.

C. Simulator Correlation across GPU Generations

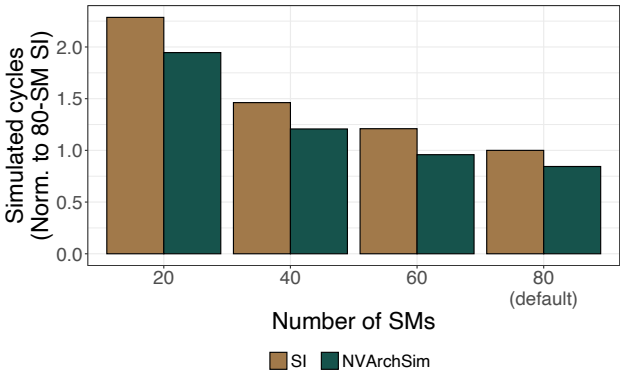
For correlation purposes, we mainly focused on a Tesla GV100 SXM2 GPU from the Volta Generation, as this was the state of the art NVIDIA GPU during the main development phase. However, to evaluate how well NVAS adapts to other generations, we developed a model of a Titan RTX GPU from the Turing generation and evaluated the same set of benchmarks. We did minimal work on top of existing NVAS; we simply 1) adjusted instruction latency and throughput information and added new instructions like updated Tensor Core operations, and 2) changed architectural parameters like channel latency and bandwidth between different components in the GPU. With only this small effort, NVAS achieves good correlation results for Turing: Pearson correlation is 0.992 and MAE is 17%. This places it in a similar range as our Volta results, and in fact the results show less systematic skew than the Volta results, which confirms that NVAS adapts well across GPU architecture generations.

V. Trading off Component Fidelity versus Simulation Speed

Much of NVAS' high simulation speed is achieved through the judicious use of simulation detail and a development process that only includes high per-component fidelity when necessary. By utilizing a pluggable component infrastructure with components that by design do not expose a complex or overly detailed public interface, it is easy to swap in or out multiple implementations of the same component. This allows feature developers to use more accurate models of components when studying specific microarchitectural designs, without burdening all NVAS users with excessive detail and slow simulation speeds. Given this capability, we present data from three differing component studies to illuminate where it is and is not important to employ high fidelity when simulating NVIDIA GPUs.



(a) Silicon and simulator performance (relative to baseline frequency) as core SM and L1 frequency is reduced.

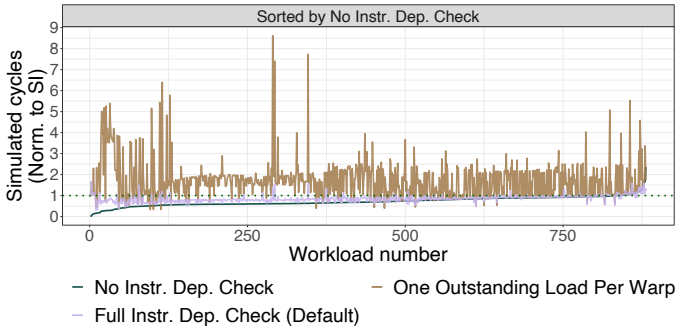


(b) Silicon and simulator performance (relative to baseline SM count) while decreasing the number of SMs that can be scheduled onto via the hardware task engine.

Fig. 9: Comparing simulator response to varying architectural parameters is a good way to ensure that your simulator is not overly or incorrectly constrained.

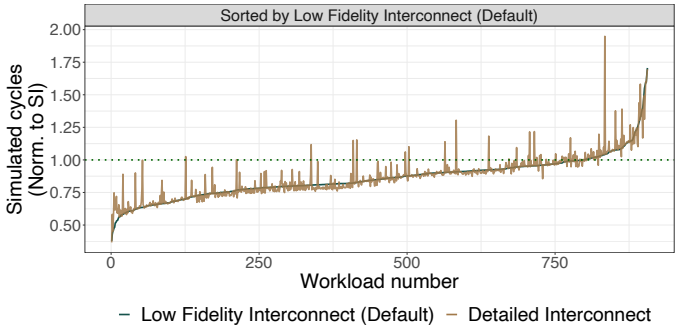
Instruction Dependency Tracking: One of the earliest fidelity features we considered in NVAS was how to best model dependency tracking within the SMs' SIMT pipelines. We initially started with no instruction dependency tracking, knowing that it was unlikely to provide good simulation accuracy. We then implemented a crude proxy for dependency tracking that only allows one outstanding load per warp, which causes the scheduler to finely interleave warps within the SM. Figure 10 shows that neither of these models provides sufficient simulation accuracy, with performance varying greatly from overly optimistic to overly pessimistic. We then developed a full dependency tracking mechanism in which an active warp yields only when a true register data dependency is encountered. The full dependency tracking improves correlation substantially and despite the added code complexity it also surprisingly improves simulation speed. This is due to the reduced number of pending memory references to track and/or to the reduced number of warp switches when compared to the other two modes. Consequently, full instruction dependency checks are used by default in NVAS.

GPU On-chip Interconnects: Another area where we have explored fidelity versus simulation speed tradeoffs is within



	No Instr. Dep. Check	One Outst. Load per Warp	Full Instr. Dep. Check
Mean Avg. Error	30%	81%	17%
Simulation Speed	0.87×	0.97×	1.00×

Fig. 10: Simulation accuracy when implementing 3 variants of instruction dependency tracking.

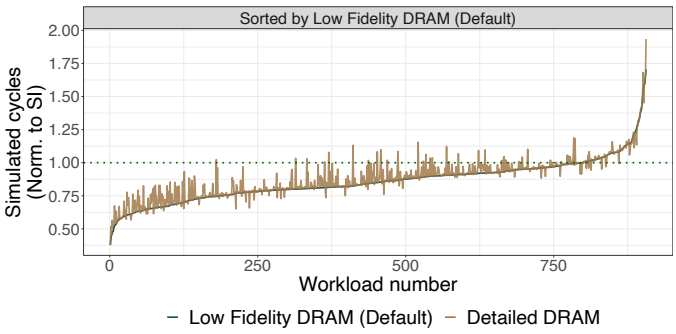


	Low Fidelity Interconnect	High Fidelity Interconnect
Mean Avg. Error	17%	17%
Simulation Speed	1.00×	0.47×

Fig. 11: Simulation accuracy when moving from low to high fidelity on-chip interconnects.

GPU on-chip interconnects between the SMs and the memory system. By default, NVAS implements this interconnect as a monolithic crossbar network that is parameterized to match the basic performance characteristics of the GPU (e.g., static latency, port bandwidth, and transaction overheads). Additionally, the default model emulates packet contention delay and packetization overheads without implementing the specific buffering and arbitration components in detail. To test the impact of interconnect fidelity on simulation accuracy, we replaced the default model with a flit-level, cycle-accurate network simulator based on BookSim [28]. Using this network simulator, we created a high fidelity interconnect model that matches the topology, router architecture, and arbitration algorithms employed on modern GPUs.

Figure 11 shows the comparison between the two models. Overall, the high fidelity model showed a less than 1% change in mean absolute error compared to the low fidelity default NVAS model. In workloads where the simulation ran faster than hardware, the detailed model tends to increase the simulation runtime. This is expected, as the detailed model



	Low Fidelity DRAM	High Fidelity DRAM
Mean Avg. Error	17%	16%
Simulation Speed	1.00×	0.88×

Fig. 12: Simulation accuracy when moving from low to high fidelity off-chip DRAM model.

can capture more interconnect contention effects such as head-of-line blocking and arbitration inefficiencies. These in turn increase the latency of some memory transactions, leading to an increase in simulation cycles.

Nevertheless, the reason that the high fidelity interconnect model has such a small impact on simulation accuracy is in part due to the throughput-oriented nature of the GPU. Because GPU applications tends to be throughput-dominated and are designed to stream traffic from the memory system, GPU interconnects are generally designed to be devoid of any systematic bandwidth bottlenecks caused by either topology or router architectures. Therefore, a simpler interconnect model that adequately captures the bandwidth and average latency of the hardware is sufficiently accurate for most simulation use cases. Furthermore, the average simulation speed when employing the high fidelity model is 0.47× that of the default model. Such a large slowdown in simulation speed compared to accuracy improvement has motivated us to maintain a simpler GPU interconnect as the default model within NVAS. For studying designs that significantly alter the performance characteristics of the interconnect or in areas of known bottlenecks (e.g., GPU off-chip interconnects) NVAS users can selectively deploy the detailed model.

Off-chip DRAM Memory System: Similar to the on-chip GPU interconnect, by default NVAS implements a very simple model for the DRAM subsystem in which memory requests have a fixed minimum latency (reflecting the empty-pipe latency) and are serviced in arrival order, while being constrained by the maximum bandwidth the memory system can sustain. With high memory bandwidth being a key driver of GPU adoption (in addition to math throughput), developers typically optimize their algorithms to saturate DRAM bandwidth whenever possible. Consequently, good memory system utilization is often the key to high overall system performance.

The performance of the DRAM system heavily depends on access patterns, the request scheduling algorithm, and the timing characteristics of the DRAM device. As such, we implemented several new NVAS memory system components at much higher fidelity including a GPU memory controller and HBM devices similar to prior work [12], [46]. Fidelity improvements come from detailed modeling of the DRAM resources (e.g., address/command and channel bandwidth, bank and row-buffer state, command interactions) and associated timing constraints, and from modeling the memory-controller request arbitration and command scheduling policies that aim to maximize throughput while maintaining fairness and low latency. In keeping with the rest of NVAS, the detailed memory model is event-based, unlike most other standalone cycle-accurate DRAM simulators [11], [30], [33].

Overall, the high fidelity NVAS DRAM model reduces simulation mean absolute error by 1.7% relative to NVAS' default simple DRAM model. As shown in Figure 12, many workloads show marked improvements, as their simulated execution time with the detailed model increases to approach the execution time measured on hardware, indicating that the simple model consistently is overestimating the achievable bandwidth in the system. Unsurprisingly, adding increased microarchitectural rooflines within the memory subsystem reduces the achieved bandwidth in these workloads. These workloads typically have two characteristics that make them sensitive to high fidelity DRAM simulations; namely, they are bandwidth-sensitive and have access patterns that cause bank conflicts leading to bandwidth under-utilization. In other words, even though these applications generate enough requests to keep the memory system busy, the controller cannot find enough row-buffer hit requests and bank-parallelism in the access stream to saturate the DRAM's bandwidth limits.

While still under active development, high fidelity DRAM simulation can provide dramatic accuracy improvements to a specific class of workloads in the NVAS application library. Without this simulation fidelity we have yet to uncover an easy way to estimate the realistic DRAM bandwidth utilization of these applications, because ultimately the off-chip memory system throughput is decided by a complex interaction of the DRAM bank states, the arrival order of requests, and the address patterns. Whereas the high fidelity on-chip GPU interconnect causes simulation times to more than double, NVAS' high fidelity DRAM components increase average simulation time by just 14%. Further testing and optimization are necessary before we can decide if this slowdown is worth the accuracy improvement for all NVAS users by default.

VI. Related Work and Outreach

There has been an enormous amount of work put into simulation to help researchers to explore their new hardware designs without building the actual hardware. We acknowledge the countless researchers and developers who have contributed to CPU and GPU simulators in the past and apologize if we omit any work due to the enormity of literature.

Numerous GPU simulators are in use in the field [3], [6], [10], [16]–[18], [20], [23], [25], [26], [29], [32], [38], [48], [58], [60], [62]. GPGPUSim is one of the more widely used examples; it is an execution-driven and cycle-level simulator that executes publicly documented NVIDIA PTX instructions.

Accel-Sim adds new GPU simulation models up to NVIDIA Turing and a new trace-driven simulation mode to GPGPUSim [29]. The added trace-driven mode relies on the SASS-level traces generated from Accel-Sim tracer, an NVBit tool, and its simulation speed more than doubles its execution-driven mode [29], [57]. For AMD GPUs, Multi2Sim supports both AMD's GCN1 ISA, while gem5-gpu and MGPUSim are able to simulate the GCN3 ISA [21], [22], [55], [56]. All continue to improve regularly.

In general, CPU simulators can be categorized into many intersecting classes: application-level vs full-system, trace-driven vs execution-driven, functional execution vs timing execution [36]. Unlike GPU simulation, most CPU simulators must also account for the other components in a system besides the CPU itself, thus full-system simulation is often used [7], [47], [59]. In addition, due to the latency sensitive nature of CPU execution, CPU simulators often use the execution-driven and cycle-accurate approach for high accuracy [7], [47]. Thanks to the public PIN tool, fast instrumentation-based simulation can be used to feed the simulations as well [27], [34], [51]. To speed up simulation, parallelization, sampling, fast forwarding using virtualization, and checkpointing are often used [7], [51], [53], [59].

During the development of NVAS, we got a lot of help from and were inspired by the knowledge shared by the research community. In response, we are trying to share what we have learned back with the community by writing this paper and by collaborating with researchers building academic GPU simulators. In particular, we worked directly with the developers of Accel-Sim [29] to support them in the creation of Accel-Sim Tracer, a methodology which also uses NVBit to feed the simulator's new trace-driven execution mode.

VII. Conclusion

In this paper, we share our experiences building a simulator that is fast enough to model large emerging (multi-)GPU workloads, while remaining accurate and trustworthy enough for the exploration of novel GPU designs within NVIDIA. NVArchSim has been successfully used for our internal GPU research across several different domains and continues to serve as our primary tool for future GPU systems research. NVAS is designed to augment, rather than to replace, more detailed simulation flows and is thus often the first tool used to explore a new idea, but rarely the last. NVArchSim remains under active development; we share our experiences thus far in the hopes that other system-level GPU simulators can learn from our observations and mistakes to further enhance the high-quality GPU research occurring in the architecture community.

References

[1] N. Agarwal, D. Nellans, E. Ebrahimi, T. F. Wenisch, J. Danskin, and S. W. Keckler, "Selective GPU Caches to Eliminate CPU-GPU HW Cache Coherence," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[2] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page Placement Strategies for GPUs within Heterogeneous Memory Systems," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[3] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[4] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[5] Baidu Research, "DeepBench," https://github.com/baidu-research/DeepBench, [Online; accessed 28-Jul-2020].

[6] A. Bakhoda, G. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads using a Detailed GPU Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, Aug. 2011.

[8] T. Burd, N. Beck, S. White, M. Paraschou, N. Kalyanasundharam, G. Donley, A. Smith, L. Hewitt, and S. Naffziger, "'Zeppelin': An SoC for Multichip Architectures," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, 2019.

[9] M. Burtscher, R. Nasre, and K. Pingali, "A Quantitative Study of Irregular Programs on GPUs," in *International Symposium on Workload Characterization (IISWC)*, 2012.

[10] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[11] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," University of Utah Technical Report UUCS-12-002, 2012. [Online]. Available: https://niladrish.org/pubs/usimm.pdf

[12] N. Chatterjee, M. O'Connor, G. Loh, N. Jayasena, and R. Balasubramonian, "Managing DRAM Latency Divergence in Irregular GPGPU Applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[13] 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 *IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[14] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2010.

[15] E. Choukse, M. B. Sullivan, M. O'Connor, M. Erez, J. Pool, D. Nellans, and S. W. Keckler, "Buddy Compression: Enabling Larger Memory for Deep Learning and HPC Workloads on GPUs," in *International Symposium on Computer Architecture (ISCA)*, 2020.

[16] H. Dai, C. Li, Z. Lin, and H. Zhou, "The Demand for a Sound Baseline in GPU Memory Architecture Research," in *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2017.

[17] V. M. del Barrio, C. Gonzalez, J. Roca, A. Fernandez, and Espasa E, "ATTILA: a Cycle-Level Execution-Driven Simulator for Modern GPU Architectures," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2006.

[18] X. Gong, R. Ubal, and D. Kaeli, "Multi2Sim Kepler: A Detailed Architectural GPU Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017.

[19] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-Tuning a High-Level Language Targeted to GPU Codes," in *Innovative Parallel Computing (InPar)*, 2012.

[20] A. A. Gubran and T. M. Aamodt, "Emerald: Graphics Modeling for SoC Systems," in *46th International Symposium on Computer Architecture*, 2019.

[21] A. Gutierrez, B. M. Beckmann, A. Dutu, J. Gross, M. LeBeane, J. Kalamatianos, O. Kayiran, M. Poremba, B. Potter, S. Puthoor, M. D. Sinclair, M. Wyse, J. Yin, X. Zhang, A. Jain, and T. Rogers, "Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[22] T. Gutierrez, S. Puthoor, B. Beckmann, and T. Ta, "The Updated AMD gem5 APU Simulator: Modeling GPUs Using the Machine ISA," *International Symposium on Computer Architecture (ISCA) Tutorial*, 2018.

[23] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model," in *International Symposium on Computer Architecture*, 2010.

[24] HSA Foundation, "HSA Platform System Architecture Specification - Provisional 1.0," http://www.slideshare.net/hsafoundation/hsa-platform-system-architecture-specification-provisional-verl-10-ratifed, 2014, [Online; accessed 28-Jul-2020].

[25] J. Huang, J. H. Lee, H. Kim, and H. S. Lee, "GPUMech: GPU Performance Modeling Technique Based on Interval Analysis," in *47th Annual International Symposium on Microarchitecture (MICRO)*, 2014.

[26] A. Jain, M. Khairy, and T. G. Rogers, "A Quantitative Evaluation of Contemporary GPU Simulation Methodology," *ACM Measurements and Analysis of Computing Systems*, 2018.

[27] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob, "CMP$im: A Pin-Based On-the-Fly Multi-Core Cache Simulator," in *Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, 2008.

[28] 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 *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[29] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[30] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 15, 2015.

[31] Lawerence Livermore National Laboratory, "CORAL/Sierra," https://asc.llnl.gov/coral-info, 2016, [Online; accessed 31-Jul-2020].

[32] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *International Symposium on Computer Architecture*, 2013.

[33] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters*, pp. 1–1, 2020.

[34] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2005.

[35] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. Janapa Reddi, T. Robie, T. St John, C.-J. Wu, L. Xu, C. Young, and M. Zaharia, "MLPerf Training Benchmark," in *Machine Learning and Systems 2020*, 2020.

[36] C. J. Mauer, M. D. Hill, and D. A. Wood, "Full-System Timing-First Simulation," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, Jun. 2002.

[37] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the Socket: NUMA-aware GPUs," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[38] T. Nowatzki, J. Menon, C. Ho, and K. Sankaralingam, "gem5, GPGPUSim, McPAT, GPUWattch, "Your favorite simulator here" Considered Harmful," 2014.

[39] NVIDIA, "CUTLASS 2.4," https://github.com/NVIDIA/cutlass.

[40] NVIDIA, "NVIDIA CUDA basic linear algebra subroutines (BLAS)," https://developer.nvidia.com/cublas.

[41] NVIDIA, "NVIDIA CUDA Deep Neural Network library (cuDNN)," https://developer.nvidia.com/cudnn.

[42] NVIDIA, "NVIDIA Tensor Cores," https://www.nvidia.com/en-us/data-center/tensor-cores/.

[43] NVIDIA, "NVIDIA NVLink High-Speed Interconnect," 2016, accessed: 2016-06-20. [Online]. Available: http://www.nvidia.com/object/nvlink.html

[44] NVIDIA, "NVIDIA Turing Architecture In-Depth ," 2018. [Online]. Available: https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/

[45] Oak Ridge National Laboratory, "Summit," https://www.olcf.ornl.gov/summit/, 2018, [Online; accessed 31-Jul-2020].

[46] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[47] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore X86 CPUs," in *Design Automation Conference (DAC)*, 2011.

[48] 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*, 2015.

[49] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "MLPerf Inference Benchmark," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[50] X. Ren, D. Lustig, E. Bolotin, A. Jaleel, O. Villa, and D. Nellans, "HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[51] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[52] S. Secchi, J. B. Manzano, O. Villa, and A. Tumeo, "Fast and Accurate Simulation of the Cray XMT Multithreaded Supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, Dec 2012.

[53] T. Sherwood, E. Perelman, and B. Calder, "Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[54] M. D. Sinclair, J. Alsop, and S. V. Adve, "HeteroSync: A Benchmark Suite for Fine-Grained Synchronization on Tightly Coupled GPUs," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2017.

[55] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, H. Barclay, A. K. Ziabari, Z. Chen, R. Ubal, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, "MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization," in *International Symposium on Computer Architecture (ISCA)*, 2019.

[56] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[57] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[58] X. Wang, K. Huang, A. Knoll, and X. Qian, "A Hybrid Framework for Fast and Accurate GPU Performance Estimation through Source-Level Analysis and Trace-Based Simulation," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[59] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, no. 4, Jul. 2006.

[60] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[61] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, "Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[62] Y. Zhang and J. D. Owens, "A Quantitative Performance Analysis Model for GPU Architectures," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[63] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards high performance paged memory for GPUs," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.