A Quantitative Evaluation of Contemporary GPU Simulation Methodology

AKSHAY JAIN*, MAHMOUD KHAIRY*, and TIMOTHY G. ROGERS, Department of Electrical and Computer Engineering, Purdue University

Contemporary Graphics Processing Units (GPUs) are used to accelerate highly parallel compute workloads. For the last decade, researchers in academia and industry have used cycle-level GPU architecture simulators to evaluate future designs. This paper performs an in-depth analysis of commonly accepted GPU simulation methodology, examining the effect both the workload and the choice of instruction set architecture have on the accuracy of a widely-used simulation infrastructure, GPGPU-Sim. We analyze numerous aspects of the architecture, validating the simulation results against real hardware. Based on a characterized set of over 1700 GPU kernels, we demonstrate that while the relative accuracy of compute-intensive workloads is high, inaccuracies in modeling the memory system result in much higher error when memory performance is critical. We then perform a case study using a recently proposed GPU architecture modification, Cache-Conscious Wavefront Scheduling. The case study demonstrates that the cross-product of workload characteristics and instruction set architecture choice can affect the predicted efficacy of the technique.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Model verification and validation**; • **Hardware** → **Hardware accelerators**;

Additional Key Words and Phrases: GPGPU; Modeling and Simulation

ACM Reference Format:
Akshay Jain*, Mahmoud Khairy*, and Timothy G. Rogers. 2018. A Quantitative Evaluation of Contemporary GPU Simulation Methodology. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 2, Article 35 (June 2018), 28 pages. https://doi.org/10.1145/3224430

1 INTRODUCTION

Modeling has been a long standing challenge for computer architects. Architects create cycle-level simulators to evaluate future hardware designs with minimal effort. In order to achieve this design goal, architectural simulators must be flexible enough to make fundamental changes to the microarchitecture without excessive modeling effort. This drives simulator designers to model elements at a higher level of abstraction. However, the efficacy of any particular architectural change must be compared against a baseline model that is roughly equivalent to a contemporary design. Accurately modeling a contemporary part is hindered both by the desire for increased abstraction and the fact that many of the microachitectural details of contemporary hardware are not publicly available. In this work we explore the effect this tension between accuracy and abstraction has on commonly-accepted simulation methodologies in the GPU space.

*These authors contributed significantly to this work and are listed alphabetically.

Authors' address: Akshay Jain*; Mahmoud Khairy*; Timothy G. Rogers, Department of Electrical and Computer Engineering, Purdue University.

2476-1249/2018/6-ART35 $15.00
https://doi.org/10.1145/3224430

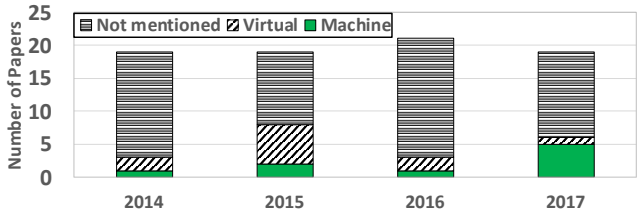


Fig. 1. A survey of published GPU simulation works in HPCA, ISCA and MICRO and their use of ISA.

Today, GPUs are commonly used for accelerating highly parallel compute workloads. They are widely used in both industry and academia for accelerating regular applications like those often found in machine learning. For the last decade, researchers have used cycle-level GPU architecture simulators to evaluate future designs. There are a number of open source GPU simulators available today. For example, Barra [12], GPUOcelot [14], Multi2Sim [44] and the recently released gpu-compute model [5] in gem5 [8] are all capable of simulating GPUs similar to those built by NVIDIA and AMD. However, GPGPU-Sim [6] is the most popular GPU simulator used in academia. Based on a survey of papers published in top-tier computer architecture conferences (ISCA, HPCA and MICRO) over the last 5 years, approximately 80% of papers on GPU architecture design utilize GPGPU-Sim. Given its pervasive use, this paper focuses on performing a detailed analysis of GPGPU-Sim's accuracy in modeling a modern NVIDIA Pascal TITAN X GPU. Although not presented in the paper, we also validate other publicly released card configurations in GPGPU-Sim. All the correlation results found in this paper, as well as those of other configurations can be found on our GPGPU-Sim Correlation Project Website [45].

GPGPU-Sim provides a flexible means of scaling the simulator to model new architectures, allowing the user to easily change the number of cores, cache configuration, memory system technology and a collection of other key parameters. There are a number of areas where the infrastructure trades off accuracy for abstraction. Certain microachitectural details, like the configuration of the register file, caches and interconnect are modeled at various levels of detail in order to maintain flexibility and decrease the overhead of reverse-engineering every detail of each new architecture. Based on the characteristics of the workload being studied, different areas of the simulator are more critical to performance than others. In this paper, we classify a group of commonly used GPU workloads based on their performance bottlenecks. Based on the workload type, we are able to draw conclusions about how well the simulator approximates the performance of the real machine it is meant to model.

The simulated Instruction Set Architecture (ISA) is another area where the tension between abstraction and accuracy are plainly apparent. GPUs employ a two step compilation process, where a high level language, is first converted into an intermediate language representation defined by the vendor, called a virtual Instruction Set Architecture (vISA). A just-in-time compilation step performed when code is launched to the GPU compiles the vISA into the mISA. In the case of the NVIDIA parts modeled by GPGPU-Sim, the mISA (called SASS) is poorly documented and can undergo frequent changes from generation to generation. However, the vISA (called PTX) is thoroughly documented and remains relatively stable. As a result, GPGPU-Sim currently supports simulating the most modern vISA and has limited support for simulating a decade old mISA (sm_13 from the GT200 architecture). Figure 1 presents a survey of all GPU papers published in HPCA, ISCA and MICRO over the past 4 years, indicating which ISA was used, if the authors specified it. Out of the 78 GPU works that used GPU simulators from 2014 to 2017, we found 11 specifically

Table 1. GPGPU-Sim performance error vs. hardware when modeling an NVIDIA Pascal Titan X.

App Type	vISA Error	mISA Error
Cache-Sensitive	104.8%	100.4%
Memory-Sensitive	31.6%	29.8%
Compute-Intensive	43.3%	21.9%
Compute-Balanced	58.5%	70.7%

cite using the vISA, 9 use the mISA and 58 that omit this detail from their paper. In this paper we demonstrate the impact ISA choice can have on simulator accuracy.

Table 1 breaks down the performance error relative to real hardware on our applications by class (detailed in Section 3). Our data demonstrates that the simulator is generally weak at modeling cache-sensitive applications (which tend to be irregular) and reasonable at modeling streaming, memory-sensitive applications. Regular, compute-intensive applications experience the lowest overall error, provided the mISA is used. In memory and cache-intensive workloads, the choice of vISA versus mISA is irrelevant. We also make the somewhat surprising observation that simulating the vISA is sometimes a better match to hardware than executing the mISA for an older machine. Changes in the mISA over product generations, such as reducing the number of instruction addressing modes, have made newer mISAs more similar in some ways to the vISA than to previous mISAs. Advances in the compiler have also improved the performance of the mISA, such that simulating fewer, more abstract instructions in the vISA results in a better performance estimation than older and less optimized mISA code.

Based on qualitative information, researchers have some intuition about which workloads the simulator accurately models. However, there is a lack of detailed analysis on the effects ISA and workload choice have on simulation error. This paper and our accompanying correlation project website [45] provide concrete data to back a researcher's assumption in all levels of the simulated system from the instruction stream through the core and into memory. This paper examines differences in the higher-level hardware performance counters by analyzing more detailed information obtained in the simulator to help understand the nature of workload-based deviations between simulation and hardware. Furthermore, we go on to analyze why the cache system in GPGPU-Sim is so inaccurate. We examine memory system issues in some detail, suggesting potential reasons and modifications that could improve how the simulator matches hardware.

Ultimately, the true test of a simulator is its ability to predict the performance of a future architecture. To quantify the effect different levels of abstraction can have on evaluating a new design, we perform a case study with a proposed research technique in GPU architecture, Cache Conscious Wavefront Scheduling (CCWS) [40]. From this case study, we show what effect workload characteristics and choice of ISA can have on the efficacy of CCWS.

This paper makes the following contributions:

- We examine the effect both workload type and ISA choice has on the modeling accuracy of a widely used GPU simulator, GPGPU-Sim, when modeling a contemporary Pascal Titan X GPU [1]. We ground these measurements by comparing them to real hardware counters that measure core, memory and system-level metrics.[2].

[1] Although we are modeling a more recent GPU, the trends and conclusions drawn are similar to what we observe in the Fermi (GTX480) configuration commonly found in many architecture papers. The Appendix provides detailed correlation data for gpgpu-sim using the GTX480 configuration and shows the same trends we see in the Pascal card are also present for the GTX480.

[2] Our correlation infrastructure and all the hardware source data is publicly available [46] and continuously run against pull requests to GPGPU-Sim on Github [39].

- We demonstrate that the average performance error obtained from GPGPU-Sim ranges from 22% to 105% and that this error is highly dependent on both the applications profiled and the ISA used to represent them. The most accurately modeled applications are streaming and their accuracy is independent of the ISA choice. The accuracy of compute-intensive applications is highly-dependent on the ISA choice, where the mISA reduces error by a factor of 2×. Applications whose performance is determined by the caching system see the highest error. Furthermore, the error introduced by the cache model is so high that it is difficult to determine if the choice of vISA or mISA matters on the cache-sensitive applications. We also make the somewhat surprising observation that there are instances where simulating the vISA results in less performance error than simulating the older mISA the simulator supports.
- We show that, while GPGPU-Sim model models in-core effects well, the memory system has serious deficiencies when executing irregular and cache-sensitive applications. We perform a detailed analysis of various performance counters in the memory system and suggest reasons for the cache model inaccuracies that should be addressed by additional simulator modeling.
- We perform a case study on academic work in GPU architecture, Cache Conscious Wavefront Scheduling (CCWS) [40] which is designed to exploit L1 cache locality. We demonstrate that selecting between mISA and vISA can have an impact on the result, depending on the choice of workload. While some apps benefit from CCWS in both mISA and vISA, other workloads only show performance improvement when using the mISA. This also suggests that even in the high-error cache-sensitive applications, the choice of ISA can still have an impact on the efficacy of an architectural technique.

The organization of this paper is as follows: Section 2 discusses the necessary background, Section 3 describes the experimental setup, Section 4 presents the results, Section 5 presents our case study of CCWS, related work is discussed in Section 6, and Section 7 concludes.

2 BACKGROUND

This sections provides concrete examples of areas where the simulator trades off abstraction for accuracy. Section 2.1 details assumptions made in the microarchitectural model that may create error and Section 2.2 details difference in the simulated mISA and vISA representations of the program.

2.1 Simulation Model

As architecture researchers, we have a strong desire to explore design spaces unencumbered by the practical constraints and minutia of modeling something exactly as it was built by company X. However, we must remain grounded. Like all things in engineering, a tradeoff exists here. On the one hand, it is not our job to model machines exactly as they are built, on the other if we do not model something that can or does exist in the real world we risk either designing techniques that were implemented years ago or worse designing techniques that are not useful.

GPGPU-Sim[6] provides a detailed full-system simulation model of a discrete GPU. Figure 2 depicts the architectural model that GPGPU-Sim simulates. It models massively multithreaded streaming multiprocessors (SMs) and memory partitions that are connected with each other via on-chip interconnection network (e.g. crossbar). A typical GPGPU kernel consists of numerous thread blocks and each thread block is composed of up to 2 thousand scalar threads. A thread block scheduler distributes the thread blocks among SMs in a load-balanced fashion. An SM runs thousands of threads concurrently. Instructions from groups of 32 consecutive threads are executed in lock-step and form what is known as a warp. At runtime, the execution of warps are scheduled and interleaved with each other using low-overhead context switching. This massive multithreading

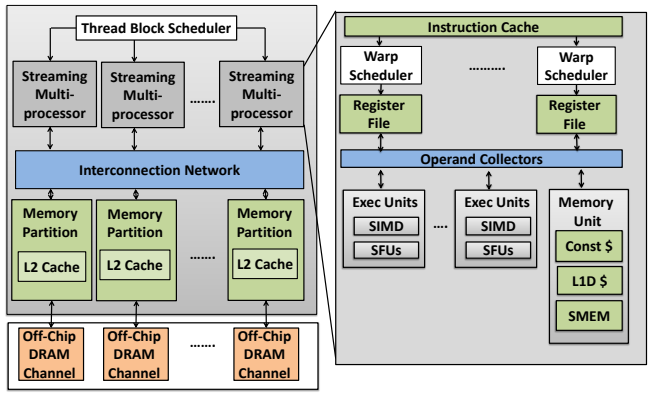


Fig. 2. GPGPU-Sim Architecture Model

hides long memory and execution latencies. When threads in the same warp execute different control flow paths, a hardware-based stack reconvergence mechanism is used to handle control flow divergence [18]. Each SM contains a private L1 data cache, constant cache, instruction cache and software-managed scratchpad, known as a shared memory. They also have a large register file that can sustain the massive threads' private data. Warps are executed on single instruction multiple data units (SIMD) and special function units (SFUs). The register file and execution units are connected via an operand collector [26] which is designed to mitigate bank conflicts in the highly banked register file. Memory partitions contain an L2 cache slice and an off-chip DRAM channel.

Researchers have performed detailed studies [24, 27, 28, 49] to demystify the GPU architecture model through micro-benchmarking. These studies revealed useful information about cache and execution units latency and organization. However, there are still some mysterious architecture details that are not disclosed yet, such as warp scheduling, cache replacement and allocation policy, register file organization, memory address mapping, interconnection arbitration and DRAM memory scheduling. These architecture details can have a crucial impact on the accuracy of GPGPU-Sim relative to hardware. Further, the last major modeling effort made in GPGPU-Sim was more than 5 years ago, when the simulator was updated to model the now 7 year old Fermi architecture [31]. Since then, the standard practice in GPGPU-Sim to model modern GPUs, such as Kepler [33] and Pascal [16], is to scale the configuration of the Fermi model by changing parameters like the number of cores, clocks, memory channels, etc.. without performing a rigorous validation of such a configuration against modern hardware and without making material changes to the simulator's baseline model in code. Another potential source of GPU simulation inaccuracy is the decision on which ISA to simulate.

2.2 Machine ISA vs Virtual ISA

GPUs from both NVIDIA and AMD [5, 17] make use of vISAs. This work focuses on the NVIDIA architecture, where code is compiled into Parallel Thread Execution (PTX). The PTX instruction set is well documented [4], making it easier for the open-source community to develop simulators for it. The Nvidia finalizer *ptxas* converts PTX into the machine language, commonly known as Source and Assembly (SASS). The implementation details of SASS are not well documented, and require reverse-engineering to fully implement in simulation. As a result, the support for SASS in open source simulators is lacking.

GPGPU-Sim [1, 6] is the most commonly used GPGPU simulator used in the architecture community and supports a large number of applications when simulating PTX. It also supports a subset of SASS sm_13 instructions from the old GT200 architecture (released in 2009) whereas the current Pascal architecture executes SASS sm_6x. To simulate SASS, GPGPU-Sim translates the mISA into a new format called PTXPlus. PTXPlus is a 1:1 representation of the GT200 SASS [3], translated to syntactically resemble PTX and ease the parsing of the two different ISAs in GPGPU-Sim's instruction parser.

Qualitatively, the following list summarizes the key differences between the vISA (PTX) and mISA (PTXPlus):

- **Register allocation**: PTX is not register allocated. Since PTX is a virtual ISA, it has no information about the hardware resources, and assumes infinite architectural registers. The register allocation is done in the finalization phase where the architecture specifics are known to *ptxas*. GPGPU-Sim uses an in-order scoreboard to avoid dependency hazards. PTX contains fewer write-after-write and write-after-read hazards, since PTX uses single static assignment.
- **Memory Addressing modes** PTX uses explicit load and store instructions to access memory, while GT200 SASS supports more memory addressing modes. In SASS, ALU instructions can directly access memory as one of the input operands. This allows the ALU operations to directly access global, shared or constant memory. Thus, a single SASS instruction could be doing the same work as performed by multiple PTX instructions. Table 2 shows a concrete example of this.

Table 2. Code demonstrating difference between parameter loads, and higher code density of PTXPlus. Code segment taken from invert_mapping in kmeans [11].

PTX	PTXPlus
mov.u16 %rh1, %ctaid.x;	mov.u16 $r0.hi, %ntid.x;
mov.u16 %rh2, %ntid.x;	cvt.u32.u16 $r1, $r0.lo;
mul.wide.u16 %r1, %rh1, %rh2;	mad.wide.u16 $r1, %ctaid.x, $r0.hi, $r1;
cvt.u32.u16 %r2, %tid.x;	
add.u32 %r3, %r2, %r1;	
ld.param.s32 %r4, [__cudaparm..npoints];	
setp.le.s32 %p1, %r4, %r3;	set.le.s32.s32 $p0\|$o127, s[0x0020], $r1;
@%p1 bra $Lt_0_2050;	@$p0.ne retp;
ld.param.s32 %r5, [__cudaparm..nfeatures];	
mov.u32 %r6, 0;	
setp.le.s32 %p2, %r5, %r6;	set.le.s32.s32 $p0\|$o127, s[0x0024], $r124;
@%p2 bra $Lt_0_2562;	@$p0.ne retp;
...	
$Lt_0_2562: $Lt_0_2050:	
exit;	

- **Parameter and thread ID initialization**: Thread ID is used in GPU kernels by every thread to figure out the part of work it needs to perform. As PTX is a virtual ISA, it is unaware of the application binary interface (ABI) used during the execution of the kernel on GPU. Hence, the thread ID is directly referred to as tid.x, tid.y and tid.z. However, in hardware, the runtime will initialize these and SASS assumes they are available in register $r0. GPU kernels

[3] In this paper, PTXPlus refers to GT200 sm_13 SASS mISA

Table 3. Example of SASS optimizing load-use delay, and using mov instructions for writing to shared memory. Bolded instructions and underlined registers show the difference in load to use delay between PTX and PTXPlus. This code segment is taken from needle_cuda_shared_1 in nw [11].

PTX	PTXPlus
ld.global.s32 **%r14**, [%rd15+4];	**ld.global**.u32 **$r5**, [$r1];
st.shared.s32 [%rd11+0], **%r14**;	add.u32 $r1, $r3, 0x00000004;
add.s32 %r15, %r13, %r6;	**ld.global**.u32 **$r4**, [$r1];
cvt.s64.s32 %rd16, %r15;	add.u32 $r1, $r6, 0x00000004;
mul.wide.s32 %rd17, %r15, 4;	**ld.global**.u32 **$r3**, [$r1];
add.u64 %rd18, %rd17, %rd14;	add.u32 $r1, $r7, 0x00000004;
ld.global.s32 **%r16**, [%rd18+4];	**ld.global**.u32 **$r1**, [$r1];
st.shared.s32 [%rd11+64], **%r16**;	mov.u32 s[$ofs1+0x04b4], **$r5**;
mul.lo.s32 %r17, %r6, 2;	mov.u32 s[$ofs1+0x04f4], **$r4**;
add.s32 %r18, %r13, %r17;	mov.u32 s[$ofs1+0x0534], **$r3**;
cvt.s64.s32 %rd19, %r18;	shl.u32 $r19, s[0x0020], 0x00000002;
mul.wide.s32 %rd20, %r18, 4;	shl.u32 $r18, s[0x0020], 0x00000003;
add.u64 %rd21, %rd20, %rd14;	mov.u32 s[$ofs1+0x0574], **$r1**;
ld.global.s32 %r19, [%rd21+4];	... 16 other ALU operations for calculating next 4 addresses.
...	

also receive parameters from the host, similar to a usual function call. These parameters are copied over by the CUDA runtime into GPU memory before kernel dispatch. In simulations, this needs to be taken care of by the simulator. GPGPU-Sim routes parameter loads through the constant memory path in PTX mode. In GT200 SASS, the ABI assumes that parameters are pre-loaded into shared memory prior to kernel launch.

- **Additional code optimizations** : Although not explicitly documented, our experiments have revealed that a number of common compiler optimizations are not performed on PTX, however they are performed in SASS. These include strength reduction and rescheduling for single-thread instruction-level parallelism. An explicit example of this reordering is shown in Table 3. For example, in the PTX, r14 is loaded then immediately reused in the next instruction. However, in the PTXPlus, the first load to r5 is separated from its use by 6 instructions, allowing the GPU to continue scheduling instructions from this warp until the true data dependency is reached. Control-flow management is also optimized in the SASS. Both the vISA and mISA are scalar ISAs and assume hardware support for branches in the SIMT execution model, however the SASS makes more use of explicit predicates than the PTX.
- **Architecture-specific instructions** : Since PTX is architecture-agnostic, SASS is able to use more highly-optimized arch-specific instructions, that do not make sense in the more abstract PTX definition.

It is worth mentioning that there have been noticeable differences in NVIDIA mISA versions over GPU generations. Based on our analysis, the recent Pascal SASS (sm_6x) has better instruction scheduling to exploit instruction-level parallelism and hide memory latency compared to the GT200 SASS (sm_13) PTXPlus simulates. Additionally, more recent versions of SASS have less addressing modes. This makes the modern mISA more RISC like and more similar to the PTX vISA. Thus, sm_6x and PTX both use explicit load and store instructions to access memory, whereas sm_13 accesses memory directly as operands.

Table 4. Pascal Nvidia Titan X Configuration

#SMs	28
#Warps per SM	64
#Schedulers per SM	4
#Warps per Sched	16
RF per SM	64 KB
Shared Memory	96 KB
L1 cache (Off by default)	48 KB, 128B line
L2 cache	3 MB, 24 banks, 128B line
Freq(core:inter:L2:mem)	1417:1417:1417:2500 MHZ
Interconnection	28X24 crossbar, 32B flit
Memory BW	480 GB/sec

Table 5. Workload Category

Type	Abbr	IPC	L1 missrate	L2 missrate	Mem Util
cache-sensitive	CS	Low	High	Mod	Low
memory-sensitive	MS	Mod	High	High	High
compute-intensive	CI	High	Mod	Mod	Mod
compute-balanced	CB	Mod	Mod	Mod	Mod

3 EXPERIMENTAL METHODOLOGY AND SETUP

This section details how we created our workload classifications, how we collected our simulation and hardware results, and what metrics we used to evaluate the data.

Simulation: We use GPGPU-Sim [6, 25], a cycle-level microarchitectural model of an NVIDIA-like GPU for general-purpose computation. The simulations were done using development version 3.2.2 with a few minor bug fixes and additional performance counters. Our GPGPU-Sim version supports the latest PTX and GT200 (sm_13) SASS, represented as PTXPlus. For supporting SASS, it does a 1:1 translation of SASS into PTXPlus, which is an extension over PTX for supporting additional addressing modes, condition codes and instructions used in SASS. We configure GPGPU-sim to resemble a contemporary GPU, The Nvidia Titan X [37] that has the Pascal P102 architecture [36] and comes with high-bandwidth GDRR5x memory [42]. We use the generally-accepted practice of scaling GPGPU-Sim to represent the Pascal Titan X. Table 4 lists the Titan X configuration. These configuration parameters are adopted from publicly available documents and resources from Nvidia [35, 36].

Since the most recent SASS that GPGPU-Sim supports is sm_13 and contemporary cards run different ISAs, the current practice for running PTXPlus to model a new card is to compute the register consumption reported by ptxas for the newer mISA version to determine occupancy. It is critical that the occupancy of the simulated model matches hardware, hence both PTX and PTXPlus calculate occupancy based on the Pascal Titan X (sm_61) register usage. In all our simulations, PTXPlus is the mISA SaSS sm_13 representation of the program and PTX is the latest vISA description of the program. Both PTX and PTXPlus compute SM occupancy by using sm_61 register consumption.

Workloads: We used a total of 29 compute applications (which spawn a total of 1797 kernels) from Rodinia benchmark suite [11], Nvidia Cuda SDK 4.2 [32], SHOC [13], Parboil [43] and GPU

Table 6. Workload Characterization

Type	Workloads
cache-sensitive	backprop (BP)[11], BFS[11], kmeans(KMN)[11], streamcluster (SCL)[11], hybridsort(HYBR)[11], spmv[13], kmeans-noTex(KM-NT)[40], GMV[20]
memory-sensitive	BlackScholes(BS)[32], convolutionSeparable(CONV) [32], fastWalshTransform(FWT)[32], scalarproduct(SP)[32], vectorAddition(VEC)[32], SCAN [32]
compute-intensive	heartwall(HWL)[11], stencil (SRN)[43], sgemm(SMM)[43], cutcp(CP) [43], MRI[43]
compute-balanced	matrixMul(MM)[32], NN[11], gaussian(GAUS)[11], lud[11], SAD[43], SRAD1(SRD1)[11], SRAD2(SRD2)[11], NW[11], hotspot(HOT)[11], b+tree(B+T)[11]

Polybench [20]. The functional accuracy of the applications was verified by running the PTX and PTXPlus simulations, and comparing their output with hardware run.

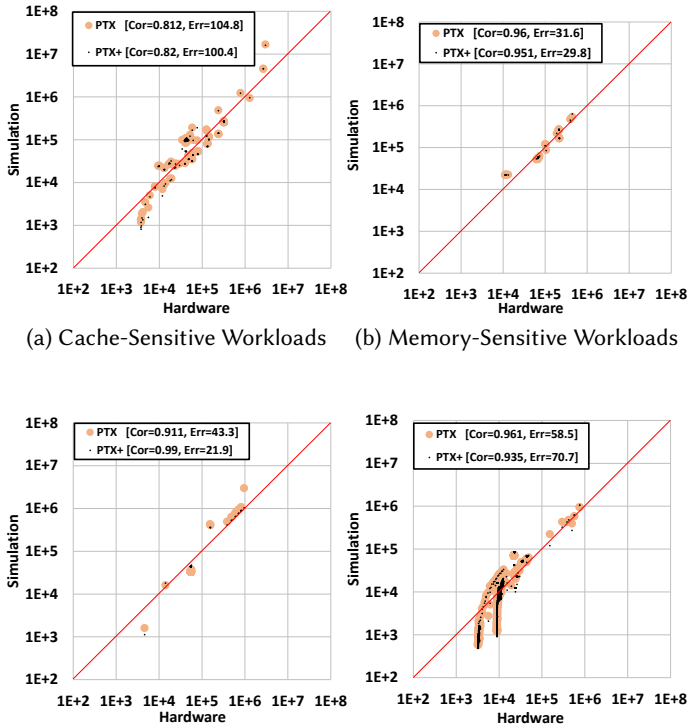
In this work, we classify our workloads as belonging to one of four categories: cache sensitive (CS), memory sensitive (MS), compute intensive (CI) and compute balanced (CB). Table 5 depicts our characterization methodology. CS workloads are highly sensitive to cache size. They suffer from cache thrashing and contention, therefore increasing the cache size for these workloads leads to significant performance improvement. For MS, they show streaming behavior with no locality to be exploited at the L1&L2 caches, thus they cause pressure on DRAM resources and are more sensitive to memory bandwidth. CI and CB workloads show moderate behavior throughout the memory system. CI workloads are characterized by a high IPC value and are limited by in-core compute resources. CB workloads exhibit a mix of the other 3 types. Table 6 lists the workload characterization that will be used throughout this paper.

Hardware Data Collection: For hardware data collection, we use the Nvidia profiler *nvprof* from the CUDA 8.0 package to collect performance data from a Pascal Titan X GPU. To make sure we exclude any noise in profiler measurements, hardware data is collected ten times for every application. We use the mean of each metric over the ten iterations to correlate the number with the simulator. Standard deviation is also calculated for each metric across the ten iterations. With these, we calculate Karl Pearson Coefficient of dispersion (COD), which is simply the ratio of the standard deviation to the mean. Then the average COD is calculated by taking the average across all the 1797 kernels. The average COD was small enough for most of the statistics, except for L1/L2 hit rates, which see more variations from one run to another.

Evaluation Metrics: Correlation between hardware and the simulator is calculated using CORREL function [3]. This tells us if the trends in simulator are similar to as seen in hardware, even though the absolute error may be high. For two vectors H and S representing hardware and simulator values respectively, the correlation is given by:

$$Correl(H,S) = \frac{\sum(h-\bar{h})(s-\bar{s})}{\sqrt{\sum(h-\bar{h})^2 \sum(s-\bar{s})^2}}$$

To measure error, we use mean absolute percentage error with respect to hardware. This tells us the relative extent to which the simulation values differ from hardware. We do not include the error numbers for statistics where there are many zero values from hardware (such as cache misses). This throws off the error calculations while normalizing with respect to hardware, if the simulator reports non-zero values. In these cases, we classify the absolute error as simply being HIGH.



(a) Cache-Sensitive Workloads (b) Memory-Sensitive Workloads

(c) Compute-Intensive Workloads (d) Compute-Balanced Workloads

Fig. 3. Execution Cycles Hardware vs. Simulation, Axes in log scale

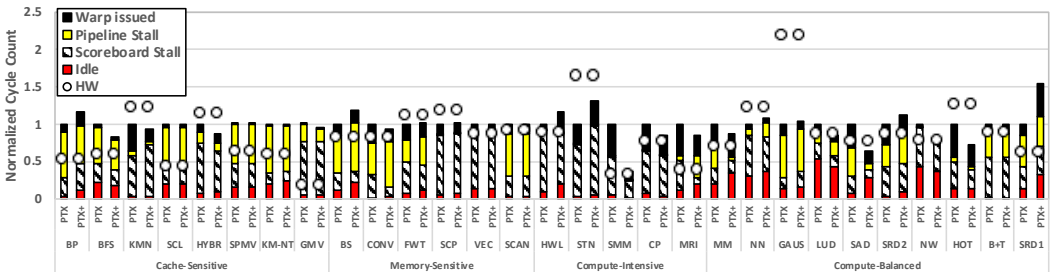


Fig. 4. Execution cycles relative to PTX. Figure shows a breakdown of stalls and useful execution. The hardware cycle count is shown by the white circles.

4 EXPERIMENTAL RESULTS

In this section, we look at how the the workload and ISA choice compare against a Pascal Titan X graphics card. We treat the hardware results as the reference point to guide us in deciding which workload type and ISA is providing results closer to hardware. We divide the results into three subsections, system, GPU core and memory system analysis.

4.1 System Analysis

In this section we study the higher-level effects of workload and ISA on simulation. In particular, we are interested in understanding how our choice of ISA affects the execution time and dynamic instruction count for each of our workload classes.

4.1.1 Execution Cycles . Figure 3 shows the execution cycle count for simulations versus the Pascal Titan X hardware, with the apps broken down into four categories. Each point corresponds to the reading from one kernel run (1797 for PTX and 1797 for PTXPlus), where the x-axis value is recorded from hardware, and the y-axis value is recorded from simulation. PTXPlus can be seen as the fine black dot, and PTX is the colored blob. The red line is drawn at x=y. A 100% correlation with no error would see the simulation data series track perfectly on the x=y line. The difference between PTX and PTXPlus for each kernel launch can be observed by measuring the distance from the center of each blob to the black dot at the same x value. A perfect match between PTX and PTXPlus simulation would see all the black dots in the center of all the blobs. Figure 3 also lists the final correlation number and relative error for both PTX and PTXPlus, per-category. We can see from the per-category figures that there are clear differences in how well GPGPU-Sim models each of the four classes of applications. Note that the axises on Figure 3 are in log scale. Starting with the cache-sensitive applications in Figure 3a, we can see the overall error in these applications is very large (> 100%). Our analysis of the cache system, as show in Section 4.3.1, reveals that the caching model for GPGPU-Sim yields vastly different results versus the hardware, which helps to explain the high error among these cache-sensitive applications. The memory sensitive apps (plotted in Figure 3b) are a different story. Here the correlation is quite high and the error relatively low. GPGPU-Sim does a fairly good job at modeling these streaming applications, implying that the DRAM bandwidth and latency model is reasonable. The compute-intensive apps (shown in Figure 3c) show a decent correlation with hardware, and demonstrate the first clear difference between PTX and PTXPlus. Although both PTX and PTXPlus track the hardware well generally, the error rate of PTXPlus is half that of PTX. The reasons for this are explored in the analysis of Figure 4. Finally, the compute balanced apps (Figure 3d) are different again. The correlation is high, but the relative error is also quite high. Here we note the somewhat surprising result that PTX has a slight advantage on absolute error. We explore this further in our detailed analysis of stalling and instructions executed below.

To better understand how the scatter plot of execution times in Figure 3 affects the overall application performance that architects wish to study, Figure 4 plots a per-application breakdown of execution cycles. Figure 4 breaks down the cycles into different types of stalls and the cycles where a warp instruction was issued.

Idle cycles are when there were no instructions ready to issue. If valid instructions are waiting for pending register writes, this is counted as a scoreboard stall. If instructions are ready and can not be issued due to a structural hazard, it is counted as a pipeline stall. Finally, the dark black at the top of the stack represents issued instructions. Also plotted in Figure 4 is the number of cycles reported by HW for all the kernels in the application. The hardware cycle count is plotted as a white circle. This figure gives us a sense of the variance in execution times, their cause and which of PTX or PTXPlus is closer to the HW.

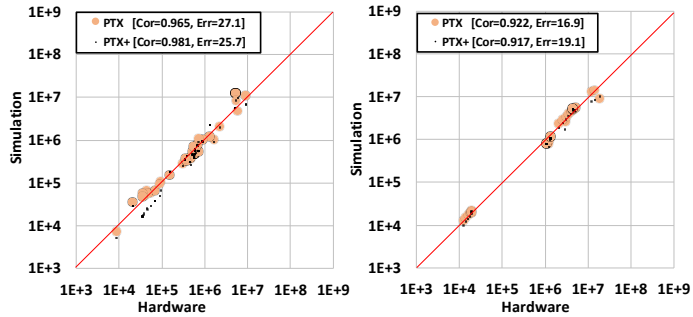
For the cache-sensitive apps in Figure 4, we can generally see that the simulator over-estimates the execution time (with the exception of HYBR and KMN). For the memory-sensitive apps, the hardware correlation is much better, irrespective of the ISA used. The story is much different in the compute-intensive apps, where PTXPlus is generally better. The most pronounced occurrence of this discrepancy is in SMM. Total instructions executed is the biggest factor in the compute-intensive workloads. Section 4.1.2 details the instruction breakdown in more detail but the synopsis is that the

hardware ISA is able to remove a large number of unnecessary move instructions that get generated in the PTX. The same holds true for CP and MRI, where the number of issued instructions falls. In STN, the instructions in PTXPlus rise due to a more realistic stalling model that comes from the deconstruction of complex PTX instructions into additional, less complex PTXPlus instructions. In all these cases, PTXPlus models the hardware more closely and produces less error. HWL is the only outlier in this category, where PTXPlus issues more instructions than PTX, resulting in PTX more closely matching the hardware. Further analysis shows that HWL is an app that relies heavily on integer division operations. In these apps, the difference in machine ISA between sm_13 (Used by PTXPlus) and sm_61 (used by hardware) comes into play. In the PTX, integer remainder and division operations are implemented by a single instruction each. However, in both PTXPlus and Pascal hardware, there are no such integer instructions and idiv/irem operations are implemented by a sequence of simpler integer instructions. This results in an instruction explosion (shown in Figure 5) in both PTXPlus and Pascal hardware versus PTX. Intuitively, PTXPlus should show better results since the instruction count is similar. However changes in the hardware instruction set and compiler instruction scheduling between sm_13 and sm_61 result in a much higher throughput on these instructions in the hardware than on PTXPlus's sm_13. Therefore, in a somewhat round-about way, PTX is able to provide a more generalized representation of the program than the code compiled for an older version of the hardware. The idealized representation of idiv/iremainder more closely matches the advances made in the architecture and compiler than the sm_13 implementation. We note that is discrepancy is also present in sm_20 (the fermi config) commonly used in the simulation papers in Figure 1. Finally, the compute-balanced apps are a mixed bag. The general inaccuracy of the cache model affects these workloads to a certain extent. We note that there are several instances (SRD2, HOT and SRD1) where PTX is actually closer to the hardware than PTXPlus. SRD1 heavily uses integer division and sees the same instruction explosion between PTX and PTXPlus (Figure 6) as HWL. However, HOT and SRD2 do not experience the same instruction explosion, yet PTX more closely matches the hardware. Section 4.1.2 explains that this difference is primarily due to the CISC nature of sm_13 decreasing instructions in PTXPlus that are not eliminated in the more RISC-like sm_61.

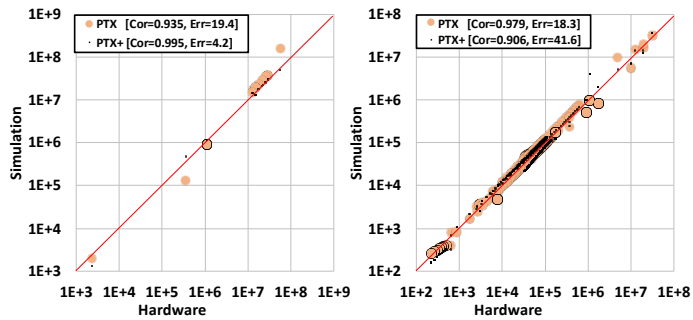
4.1.2 Dynamic Warp Instructions. To better understand the overall correlation of instructions executed for each class, this section first details the hardware correlation of dynamic warp instructions executed. We then delve deeper into these instructions, examining their composition and how closely each simulated instruction set matches hardware.

To understand the instruction mix, Figure 5 shows the dynamic warp instructions executed in simulation for PTX and PTXPlus vs. that in the Pascal Titan X hardware. Generally, across all four categories of apps, correlation is relatively high and error relatively low. This is interesting, considering there are three instruction sets at play (the vISA PTX, mISA sm_13 for PTXPlus and mISA sm_61 for the pascal hardware). Error in the PTX instruction count remains relatively stable ~20%, while PTXPlus is more volatile (varying from 41.6% in the CB workloads to 4.2% in the CI apps). We can also see that the general trends in execution time are mirrored in the instructions executed. That is, in CI apps, PTXPlus more closely matches hardware, PTX is closer in CB apps and both CS and MS apps show little variation between PTX and PTXPlus.

Figure 6 shows the dynamic instruction count for the workloads relative to PTX. It helps shed some light on the performance differences shown in figure 4. The figure also breaks down the instructions into various categories which gives us more insight into the code expansion/contraction due to differences in ISAs. In the two memory-system intensive classes (CS and MS), variances in the dynamic instruction count do not tend to translate into variances in the cycle count. For example, in BFS, KMN, VEC and CONV there is a clear advantage of one ISA versus the other.



(a) Cache-Sensitiave Workloads (b) Memory-Sensitive Workloads



(c) Compute-Intensive Workloads (d) Compute-Balanced Workloads

Fig. 5. Dynamic Instructions Executed Hardware vs Simulation, Axes in log scale.

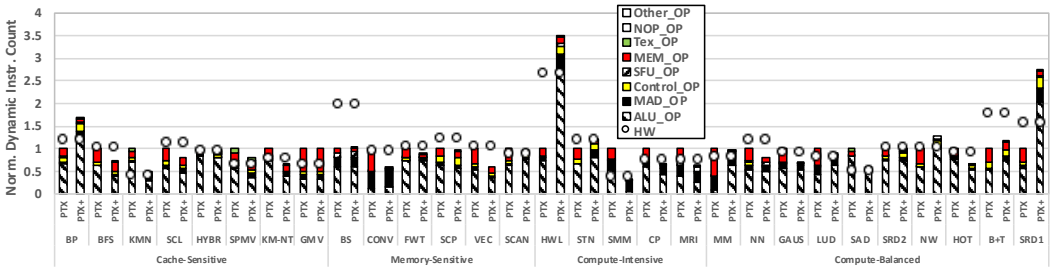


Fig. 6. Dynamic instruction count and breakdown relative to PTX. Hardware instructions executed are shown as white circles.

However, looking at the execution time of these apps in Figure 4, the ISA simulation with less dynamic instruction count error does not result in more execution time error. Fundamentally, this occurs because the performance of these apps is largely determined by the memory system (Figure 4 shows relatively little time issuing instructions, with the majority of the cycles spent idle or stalling). For the CI apps, there is a more 1:1 mapping between instructions executed and execution time. For example STN, SMM, CP and MRI show a clear advantage for PTXPlus in instructions executed, which translates into a more accurate representation of execution time. The only outlier here is HWL, where the instruction explosion associated with irem/idiv (detailed

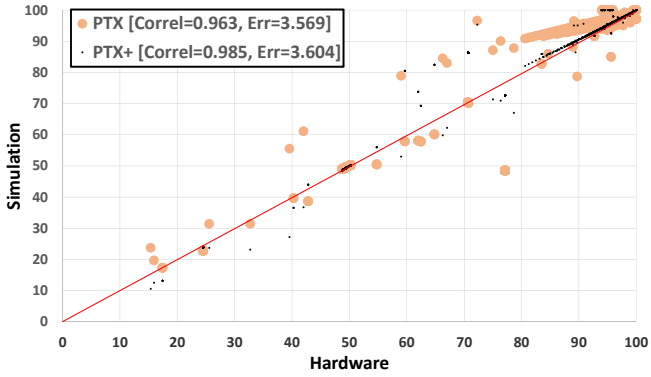


Fig. 7. SIMD Lane Occupancy in Percentage. Axises are linear.

in Section 4.1.1) results in an instruction explosion in the hardware and PTXPlus. As discussed earlier, although the PTXPlus instruction count in HWL matches hardware more closely, a marked difference in instruction throughput between PTXPlus and hardware results in a better execution time correlation for PTX. Another interesting observation from the CI workloads is a vast reduction in the number of memory instructions executed for PTXPlus versus PTX. For example STN, SMM, CP and MRI see a near elimination in MEM_OPs. The CISC-nature of the sm_13 ISA causes this difference. In particular loads from the shared memory space are mostly eliminated in PTXPlus, as they are passed as operands to ALU and MAD instructions. In some apps, we see a significant reduction in the number of ALU ops (SMM, CP and MRI for example). A closer examination of the code reveals that the PTX is particularly bad at dealing with unrolled loops. Although the loops are successfully unrolled in PTX, the process seems to wreak havoc on register allocation. The compilation flow seems to be relying on the register allocation, data flow analysis and unnecessary code removal steps in the mISA finalization phase to eliminate these move operations. Both sm_13 and sm_61 succeed in stripping out all the extra moves, resulting in a much better matching of instructions executed between HW and PTXPlus. This analysis further suggests that highly regular and optimized codes, which reach nearly peak IPC in hardware, really should be evaluated using a mISA. Finally, the compute balanced workloads show a mix of the characteristics found in the other classes. The outlier SRD1 experiences the same integer instruction explosion as HWL, while apps like SAD and LUD show similar characteristics to the CI apps, where PTXPlus is better at both instruction count and execution time. The compute balanced workloads also show the greatest number of apps where the execution time and instruction count of PTX is better than PTXPlus, in particular HOT, where the CISC nature of sm_13 results in too few operations in PTXPlus such that the reduced addressing modes in PTX are a much better representation of sm_61.

4.2 GPU Core Analysis

This section is devoted to analyzing the effect ISA choice has on in-core characteristics. From our system-level analysis in Section 4.1, the compute model seems to be the most accurate component of the simulator. This section explores several in-core metrics, namely front-end metrics like the SIMD lane occupancy (Section 4.2.1) and instruction cache hit rates (Section 4.2.2) followed by the memory accesses generated by the core in Section 4.2.3.

4.2.1 Lane Occupancy . Figure 7 shows the SIMD lane occupancy in simulation vs hardware. Lane occupancy measures the average number of active threads per issued warp instruction, i.e. a

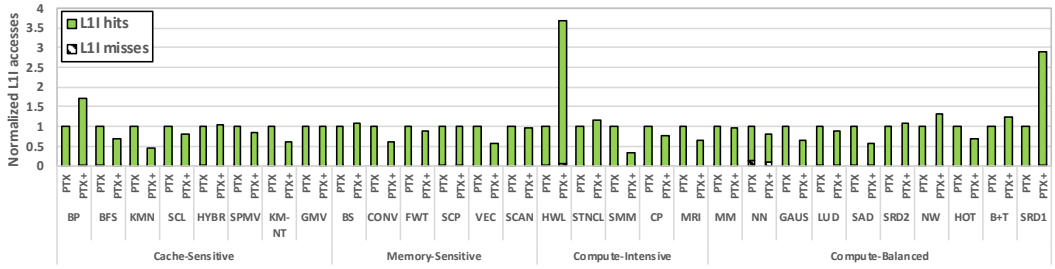


Fig. 8. L1 instruction cache hit rates

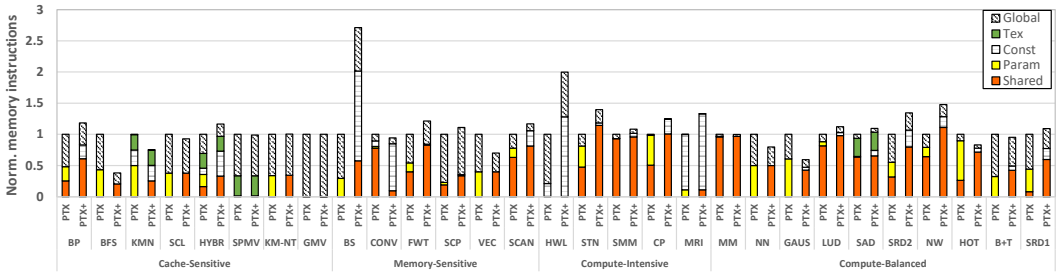


Fig. 9. Memory accesses accessed broken into region, relative to PTX

100% occupancy means that all the 32 lanes in a warp are actively executing every issued warp instruction. This can be seen in the top right corner, and represents highly regular workloads.

There was no clear differentiation in this metric among our workload classes, so we present only the aggregated graph in figure 7. For many of our kernels the lane occupancy remains very high (close to 100%) particularly in the memory-sensitive and compute-intensive categories. The cache-sensitive and compute-balanced workloads have several irregular applications that exhibit lower SIMD efficiency. There is very little variation between PTX and PTXPlus, with the exception of some outliers around 80%.

4.2.2 L1 Instruction Cache . The L1 Instruction cache (L1I) access count follows the same trend as dynamic instruction count. Figure 8 shows the L1I cache accesses broken down into hits and misses. We find that all the apps see a hit rate of more than 95%.This is interesting because, although the instruction counts can vary significantly in these workloads, there is so much locality among the many threads in GPU that frontend instruction fetch is never an issue. The total code size of GPU apps is small enough that even their modestly sized icaches capture enough locality. Even in apps with a huge number of instructions executed (like PTXPlus in HWL and SRD1) all the extra instructions are coming from larger loop bodies that get amplified by all the iterations of the loop.

4.2.3 Memory accesses generated . Figure 9 shows the count of memory region accesses made. These values are normalized with respect to PTX. Generally, the global accesses generated is held fairly constant between PTX and PTXPlus across all the apps. The parameter memory accesses in PTX are also completely eliminated in PTXPlus, since the parameters in sm_13 are loaded from shared memory.

Some notable outliers in terms of total memory accesses are BS and HWL, where PTXPlus generates many more total memory accesses which come largely from constant memory. Heartwall

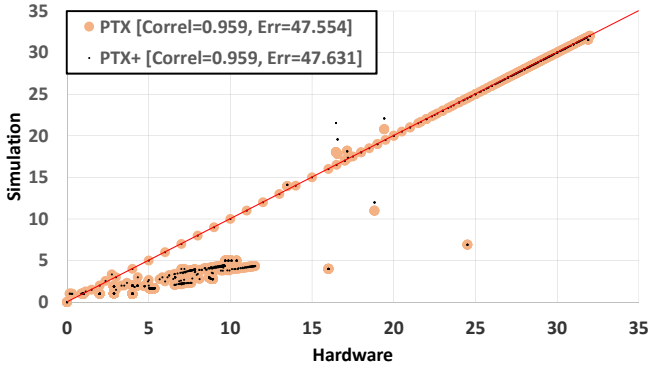


Fig. 10. Store instruction memory divergence. Average accesses generated per instruction.

makes almost 5x constant memory accesses in PTXPlus as compared to PTX. This is due to the register allocation performed in the finalization process. PTX code is over-optimistic and can work with reduced constant memory instructions because it uses 1868 32-bit registers per thread, which would prohibitively limit occupancy in a real machine. However, the same locations have to be accessed multiple times in PTXPlus as it uses only 25 32-bit registers. The same trend is seen in BS.

4.2.4 Global memory divergence. This section explores how the choice of ISA affects memory divergence. Memory divergence is defined as the average number of memory transactions generated by a single warp instruction. In the simulator, if requests from all the 32 threads in the warp access the same 128B chunk, it is called a fully coalesced access and can be serviced from a single memory transaction. On the other hand, if each of the 32 requests access a different cache line, it will result in 32 unique accesses to the memory system.

Figure 10 shows the trend for memory divergence for writes to global memory. Only writes are plotted, as reads exhibited similar behavior. We see the kernels concentrated along two major lines y=x and y=0.25x. Both PTX and PTXPlus show a similar correlation with respect to hardware and the ISA has no effect on global memory divergence.

There is an interesting observation we would like to discuss in light of the data in Figure 10. In the simulator, the GPU without the L1 cache (which is the behavior of our setup given that it is the default configuration in Pascal) will generate 32B requests when the access is diverged, and a single 128B request when the access is completely coalesced. The data in figure 10 shows that in highly coalesced kernels, the hardware is generating approximately 4× more accesses than the simulator. We believe this indicates that 32B transactions are always generated by the hardware, even when the access is fully coalesced. We also collected this data with the L1D enabled on Pascal and noticed a similar trend. This is due to fact that L1D cache has 32B sectors in the Pascal architecture [34]. The coalescer will always generate 32B requests no matter if L1 cache is enabled or not. This is a modeling detail that could be implemented in the simulator to improve memory system accuracy.

4.2.5 Global memory requests. Figure 11 shows the count of global memory transactions. This is not the raw access count from instructions, it is gathered after the coalescing logic has merged the individual lanes' requests.

We see that in figure 11, several kernels lie on the y=0.25x line, while some kernels lie on the y=x line. Others show a mixed behavior. Similar to the memory divergence behavior in Figure 10, this data indicates that up to four 32B accesses are being generated by the hardware where the simulator generates only a single 128B request to service coalesced accesses. There is little variation

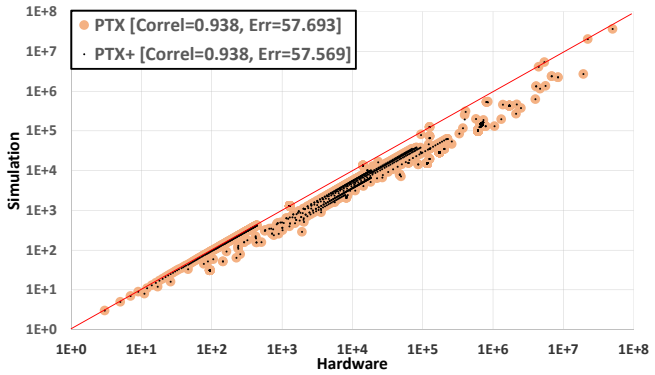


Fig. 11. Total global memory requests. Axes in log scale

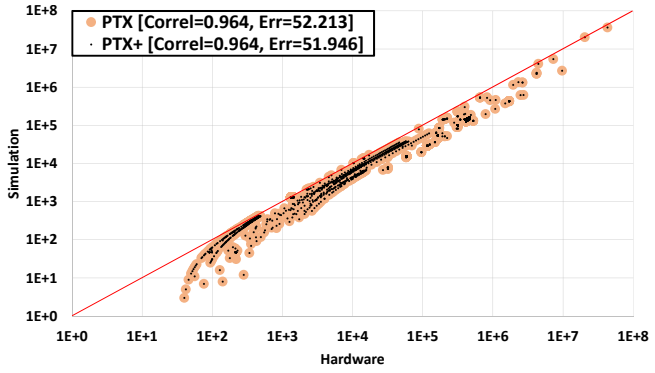


Fig. 12. L2 access count. Axes in log scale

between PTX and PTXPlus for global memory requests. The bulk of the error in here comes from the linear deviations caused by cache sectoring. This again suggests that modeling the memory coalescer and cache on a 32B sector basis will vastly reduce memory system error. These cache graphs also highlight a fundamental issue with observing correlation data only. The correlation of memory system accesses is quite high (greater than 0.93), however the 50% error help contribute to an excessive amount of error in the cache-sensitive applications. This ultimately results in a higher error in the execution time on applications where memory system accesses determine overall performance.

4.3 Memory System Analysis

In this subsection, we analyze and validate the correlation of the memory system, including L2 cache and DRAM memory. This analysis helps us to better understand the observed modeling deficiencies in the memory system.

4.3.1 L2 Cache . Figure 12 shows the L2 cache accesses in simulation vs. that in Pascal hardware. L2 cache statistics are collected while L1 is turned off. This includes traffic to all the memory regions and demonstrates that L2 accesses experience a large error with respect to hardware, regardless of the ISA used. We again see that the absolute numbers for many of the apps are off in simulation by

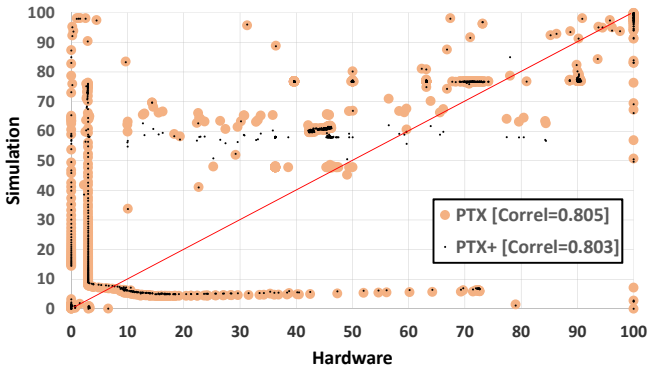
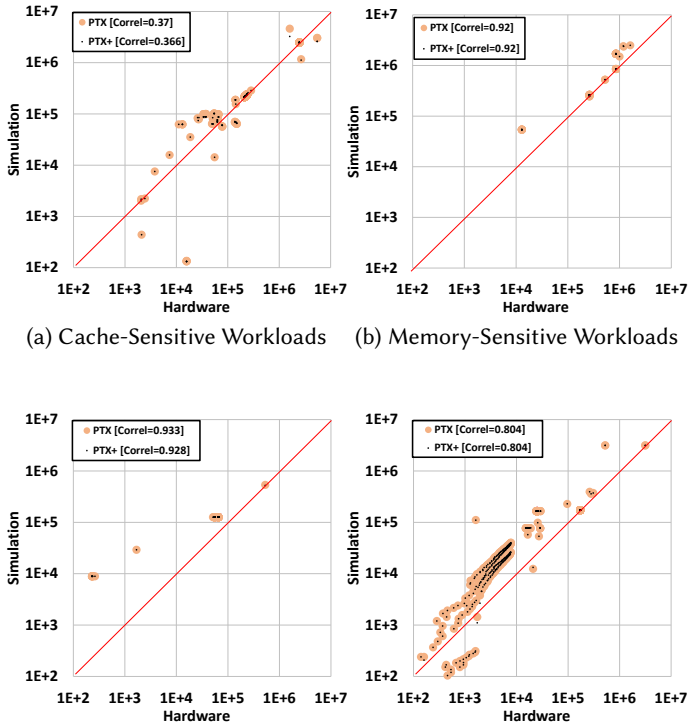


Fig. 13. L2 cache hit rate correlation

a ratio of 2:1 or 4:1. This would make sense if the hardware is reporting the count of 32B accesses to the L2 cache [2], while the simulator is assuming a non-sectored 128B cache line at the L2.

Figure 13 plots the L2 hit rate of PTX and PTXPlus simulations versus hardware. The hit rate performance in simulations does not match well with that in hardware. There are certain cases where either hardware or simulation report a negligible hit rate, while the other one sees many more hits. One of the reasons is that memory copies from CPU to GPU may be cached in GPU L2 cache [24], which is not modeled in the simulator. The high error can also be explained by a sectored cache [2, 38] and a more efficient L2 cache replacement policy used in Nvidia hardware (as disclosed in Nvidia patent [15]). These mechanisms are not currently modeled in GPGPU-Sim and appear to have existed in NVIDIA GPUs for several generations. Again, although they do not match hardware, PTX and PTXPlus are fairly close to one another and the ISA version does not play a large role here.

4.3.2 DRAM and Memory Controller. Figures 14 shows the DRAM read requests per workload category. Since there was no clear differentiation in DRAM writes among our workload classes, we present the DRAM reads only here. The simulator is reporting the number of 32B requests serviced from the DRAM. The error rates in some of the application classes are so high (because of some erroneous values being very close to zero) that we do not show them here. The CS workloads show the worst correlation at 37%, while MS, CI and CB workloads are much better, especially MS workloads that show the highest correlation with the smallest error among other classes. However, for the majority of CB and CI workloads, the simulator reports more read requests than the HW. This is because we have a poor correlation at L2 cache hit rate (as shown in the previous section). This discrepancy affects both PTX and PTXPlus simulations. Thus, using either ISA should not affect the results. The detailed results from the memory system indicate that some key errors in the cache model are creating a vast discrepancy between hardware and simulation, independent of ISA. We have uncovered some of the potential reasons for these differences. Namely, (1) The caches in hardware are sectored [2, 38], whereas GPGPU-sim only models non-sectored caches. (2) Hardware employs an efficient L2 cache write allocation policy [15]. (3) Memory copies from CPU memory to GPU memory are also cached in GPU L2 cache [24]. GPGPU-Sim does not model this, so some compulsory misses in simulation may not actually be misses in hardware. (4) Instruction scheduling in the hardware SASS sm_61 can be much better than simulated SASS sm_13 or the vISA, creating different locality patterns in each warp. We leave improving the GPGPU-Sim model as future work.



(a) Cache-Sensitive Workloads (b) Memory-Sensitive Workloads

(c) Compute-Intensive Workloads (d) Compute-Balanced Workloads

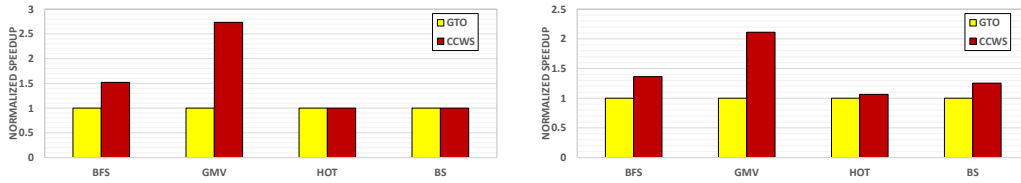
Fig. 14. DRAM Reads access count. Axes in log scale

Table 7. Summary of GPGPU-Sim Error and Correlation with respect to Pascal TITAN X hardware

Statistic	Mean Abs Error		Correlation	
	PTX	PTX+	PTX	PTX+
Execution Cycles	65.5%	71.9%	79.6%	79.2%
Dyn. Warp Insts	19.9%	35.4%	90.9%	96.1%
Lane Occupancy	3.5%	3.6%	96.2%	98.4%
Global Ld Reqs	57.6%	57.4%	93.8%	93.8%
Global St Reqs	47.2%	47.2%	90.7%	90.7%
Global Ld Div.	57.6%	57.5%	82.6%	82.6%
Global St Div.	47.5%	47.6%	95.9%	95.9%
L2 Reads	51.1%	50.7%	96.5%	96.5%
L2 Writes	55.2%	55.2%	90.8%	90.8%
L2 Hit Ratio	High	High	80.5%	80.3%
DRAM Reads	High	High	38.7%	38.2%
DRAM Writes	High	High	96.6%	96.6%

4.4 Correlation summary

Table 7 presents the absolute percent error and correlation numbers for the comparison of GPGPU-Sim reported statistics with respect to hardware for all system, core, cache and memory metrics.



(a) Performance with PTX ISA

(b) Performance with PTXPlus ISA

Fig. 15. Performance of GTO and CCWS normalized with respect to GTO [40]

We do not include the error numbers for statistics where there are many zero values from hardware (such as cache misses). This throws off the error calculations while normalizing with respect to hardware, if the simulator reports non-zero values. Instead, we replace the error with "High" in the table which means high relative error exists but cannot be quantified. We can see that in general, the simulator in-core statistics, such as warp instructions and lane occupancy, track the hardware numbers well. However, there is a considerable error between the simulator and hardware at all levels of memory hierarchy.

5 CASE STUDY

The true test of a simulator is its ability to predict the performance of a future architecture. Can one trust the results provided by the simulator for a new technique where correlation with a design is not possible? To quantify the effect different levels of abstraction can have on evaluating a new design, we perform a case study with a proposed research techniques in GPU architecture, Cache Conscious Wavefront Scheduling (CCWS) [40]. CCWS is an issue-level GPU warp scheduling technique designed to improve intra-warp cache locality. It operates by dynamically throttling the number of active warps in a core, based on predicted cache locality. The original paper compared the technique to a number of other warp schedulers including a Greedy-Then-Oldest (GTO) scheduler that was used as the baseline. GTO greedily schedules instructions from one warp until the warp can no longer issue, then it switches to the another warp. The warps candidate warps are prioritized by age with the oldest warps having the highest priority. The rationale behind the GTO scheduler is that cache locality for older warps is maintained since younger warps will have lower priority and their accesses will not interfere with the cache locality in younger warps. CCWS captures more locality than GTO because it will choose to not issue anything from younger warps (even if they are ready) when the mechanism detects older warps have locality. In this case study, we demonstrate the choice of ISA may show different results in evaluating the efficacy of CCWS. The original work was implemented using GPGPU-Sim version 3.1.0, and used the PTX instruction set for performance evaluation. We added support for running it with PTXPlus, and evaluate the proposed performance gains in the new mode. The aim of this study is to see what would have happened if the authors had used mISA instead of the vISA. Would the performance improvement be reduced if the mISA was used, and is CCWS more effective in mISA than vISA or vise versa?

Figure 15 shows the performance results of CCWS compared to the baseline GTO scheduler [40] for two workloads from the original paper (BFS and GMV) and 2 new workloads (HOT and BS) in PTX and PTXPlus modes. We can see the performance improves with CCWS in both the modes for BFS and GMV, however it only improves the performance for HOT and BS in PTXPlus mode by 7% and 25% respectively. This is because HOT and BS are limited by register usage per thread to increase the overall GPU occupancy (number of concurrently running threads). Limiting

the register usage per thread may lead to registers spilling to memory due to the insufficient hardware register resources. These memory accesses are cached in the L1 to alleviate register spilling overhead. However, in some cases, the register spilling footprint of the running warps can be large and they may leading to poor performance [29]. As we showed in Section 2.2, PTX assumes an infinite register space and has no information about the hardware resources, thus cache thrashing due to register spilling does not occur in PTX mode. For this reason, CCWS is more effective in PTXPlus for HOT and BS wherein cache thrashing occurs due to register spilling. The cache thrashing in BFS and GMV occurs due to global memory loads which are found in both PTX and PTXPlus.

6 RELATED WORK

This section details prior work both on correlating simulators to real hardware (Section 6.1) and on studying the effects of simulating different ISA representations of the same program (Section 6.2).

6.1 Correlation of simulations and hardware

In this section, we first examine those works that study CPU simulators, followed by GPU simulators.

6.1.1 CPU correlation. Several studies [7, 10, 41] have correlated various CPU simulators with real hardware. However, the methodologies used for CPU simulators cannot be directly applied to GPU simulators. The biggest difference is that work is dispatched in the form of kernels with varying behaviors, and thus it is important to look at the trends on a per-kernel basis.

6.1.2 GPU correlation. Wong et al. performed a detailed study of GT200 architecture through micro-benchmarking [49]. Their study revealed useful information about latency and organization of ALU and memory pipelines. The work does not correlate the hardware study with a simulation infrastructure though.

In terms of hardware correlation, the closest work to ours is the original correlation presented with GPGPU-Sim [1, 6], PTXPlus is shown to have a better correlation of IPC than PTX, when compared to GT200 and GTX480 GPUs. Multi2sim [19] is a CPU-GPU heterogeneous simulator that models Kepler GPU SASS and shows same trend in execution cycles with Nvidia K20. Nugteren et al. [30] investigate how to accurately model GPU cache based on reuse distance theory. The new model is more accurate than GPGPU-sim simulator and their results show a mean absolute error of 6% and 8% compared to a real hardware. Jia et al. [23] characterize the different types of locality and contentions that occur in the cache hierarchy on a real Tesla C2070 hardware. In comparison to these, we look at a more detailed correlation, with more number of GPU kernels from the diverse set of applications. Our work presents a detailed comparison between the virtual and machine ISAs, and offers insights on how these affect the simulation results. We cover control flow and data divergence characteristics, the two major sources of inefficiencies in SIMT based throughput processors with lockstep execution.

6.2 Comparison of virtual and machine ISAs

This sections is also subdivided between those works exploring CPU ISAs and those examining GPU ISAs.

6.2.1 CPU ISAs. The GPU vISA versus mISA situation is similar to the CISC x86 versus machine micro-ops that exist in many contemporary CPUs. CPUs have had virtual ISAs for a long time, and a few studies have explored the differences in vISA and mISA. Blem et al. compare the x86 CISC ISA with ARM RISC ISA [9]. They perform performance and power measurements using Intel and ARM processors, and claim the ISA being RISC or CISC is largely irrelevant for today's mature

microprocessor design world. However, they do not analyze the effects of using different ISAs on the same micro-architecture. Another work [22] looks at the most frequently used x86 instructions and maps them to the internal micro-ops. They claim micro operation analysis provides very good focus for optimization, and enables them to squeeze out 17.4% reduction in micro operation cycles.

6.2.2 GPU ISAs. T Volkov et al. [48] analyze dense matrix multiplication on Nvidia G80 GPUs. They use machine instructions generated by decuda disassembler [47] and claim that provides more insights than using PTX. Another work [19] proposes a mISA-level GPU simulator for the Kepler architecture. They show a short comparison between dynamic instructions count for mISA and vISA without hardware validation.

Concurrent work by Gutierrez et al[21] examines the effects of HSAIL and AMD Graphics Core Next 3 (GCN3) ISAs in a closed-source simulator. The HSAIL vISA model is publicly available, but the GCN3 mISA model is not. They perform an analysis that compares the scalar/vector GCM3 mISA against the register-allocated, HSAIL vISA, which is a CISC-like SIMT ISA without scalar instructions. The paper performs a validation study on a limited number of applications, showing that their mISA simulation is more accurate. In contrast, our paper performs a detailed, categorized performance and hardware counter analysis on a widely used simulation infrastructure across a diverse set of applications. Furthermore, the insights gained from the HSAIL/GCN3 comparison are orthogonal to our PTX/SASS exploration, since the tradeoffs made in the vISAs, mISAs and underlying architectures are fundamentally different.

7 CONCLUSION

Validating a performance simulator is a constant challenge for researchers in both academia and industry. This paper performs a detailed evaluation of contemporary GPU simulation methodology. We rigorously correlate the commonly used GPGPU-Sim simulator with contemporary hardware. We conclude that the accuracy of the baseline machine model is highly dependent on the characteristics of the applications being evaluated. Both intrinsic workload characteristics, such as cache-sensitivity, and the ISA representation (virtual ISA versus machine ISA) play a key role in determining how well the simulator matches the baseline machine.

Conclusions from this analysis show that, although the correlation coefficient between hardware and simulation performance is relatively high, the absolute error is also somewhat high (between 22% and 105%). The magnitude of this error is highly dependent on both workload characteristics and the ISA used to represent the program. Generally, streaming applications experience low error, while apps whose performance is determined by the cache system have the highest error. Compute-intensive applications are most accurately modeled by the simulator, provided a mISA representation of the program is used.

We also demonstrate that the caches modeled by GPGPU-Sim are very different from the caches in real hardware. A significant source of error in the simulator comes from its cache model. Through our analysis we predict that sectoring the caches and improving the replacement policy will have a significant impact on reducing both the error observed in cache statistics and in the absolute performance of applications where the cache model is critical. In particular, we have identified several aspects of the memory system that should be enhanced to improve simulator accuracy and we leave implementation and correlation of these enhancements as future work.

Finally, the intent of this study is not to dissuade researchers from using GPGPU-Sim, or simulators in general. On the contrary, we hope that a detailed understanding of simulator correlation motivates the community to contribute to simulation projects and rigorously validate their accuracy.

8 APPENDIX

For completeness, we have included all the data we collected correlating the Fermi-based GTX 480 to GPGPU-Sim, since GTX 480 is the most prominently used configuration in recent GPU simulation papers.

Table 8. Summary of GPGPU-Sim Error and Correlation with respect to GTX480 hardware

Statistic	Mean Abs Error		Correlation	
	PTX	PTX+	PTX	PTX+
Execution Cycles	45.3%	57.2%	97.1%	97.3%
Dyn. Warp Insts	29.8%	44.3%	83.7%	91.3%
Lane Occupancy	9.3%	6.8%	92.9%	97.2%
Global Ld Reqs	19.2%	19.2%	99.5%	99.5%
Global St Reqs	20.2%	20.2%	99.7%	99.7%
Global Ld Div.	19%	19%	96.9%	96.9%
Global St Div.	20.9%	20.8%	97.3%	97.0%
L1D Hit Ratio	High	High	72.1%	72.0%
L2 Reads	73.6%	73.5%	99.5%	99.2%
L2 Writes	40.6%	41.6%	94.0%	94.0%
L2 Hit Ratio	High	High	82.3%	82.2%
DRAM Reads	High	High	77.6%	53.1%
DRAM Writes	High	High	98.2%	98.1%

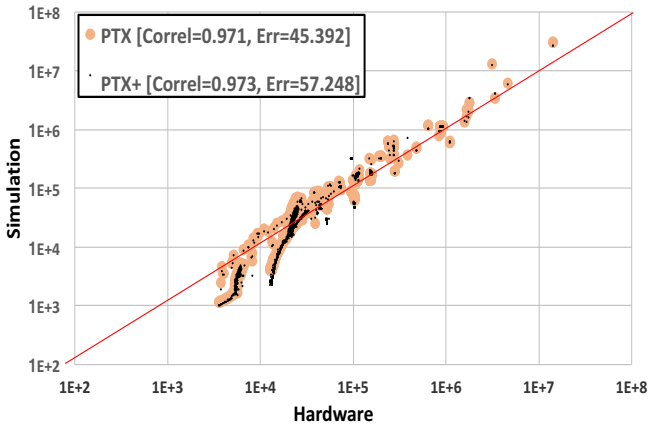


Fig. 16. GTX480: Execution cycles

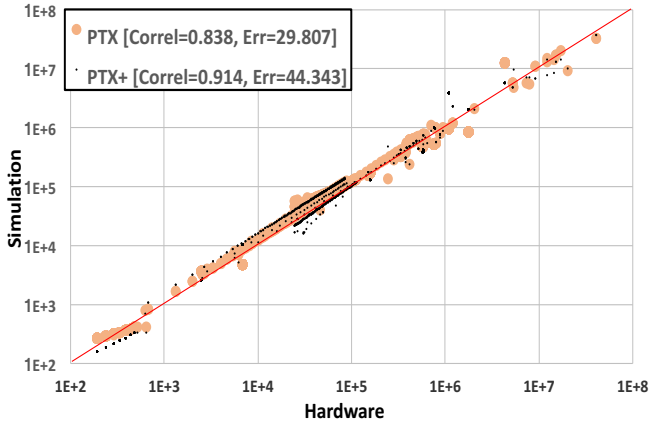


Fig. 17. GTX480: Dynamic Warp Instructions Executed

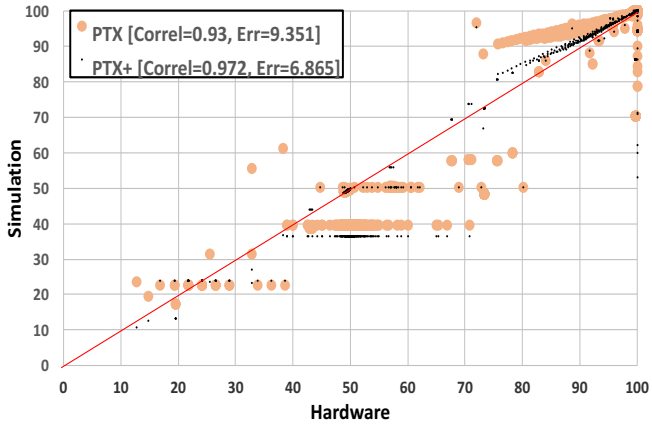


Fig. 18. GTX480: SIMD Lane Occupancy in Percentage

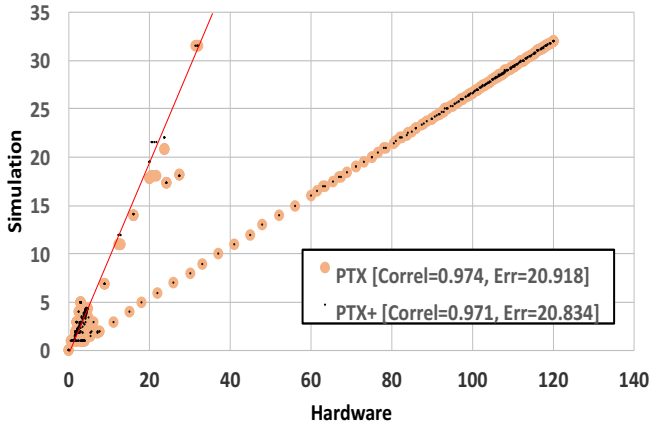


Fig. 19. GTX480: L1 store memory divergence

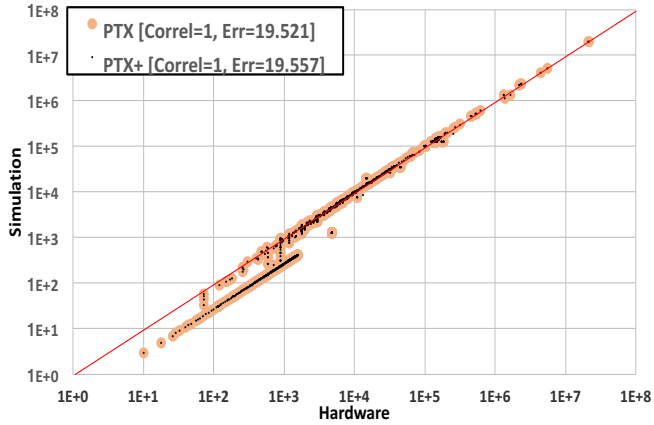


Fig. 20. GTX480: L1 global memory requests

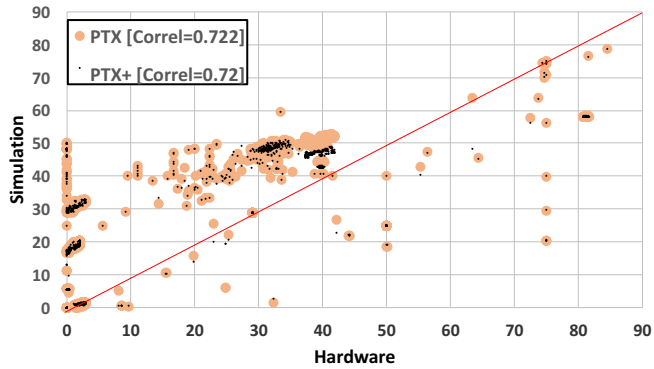


Fig. 21. GTX480: L1D Cache Hit Ratio in Percentage

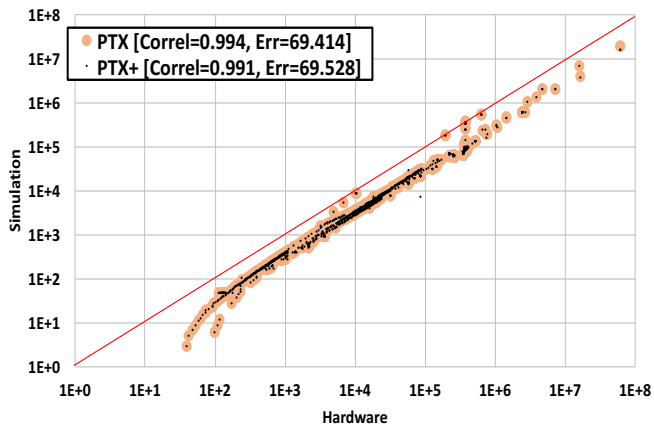


Fig. 22. GTX480: L2 access count

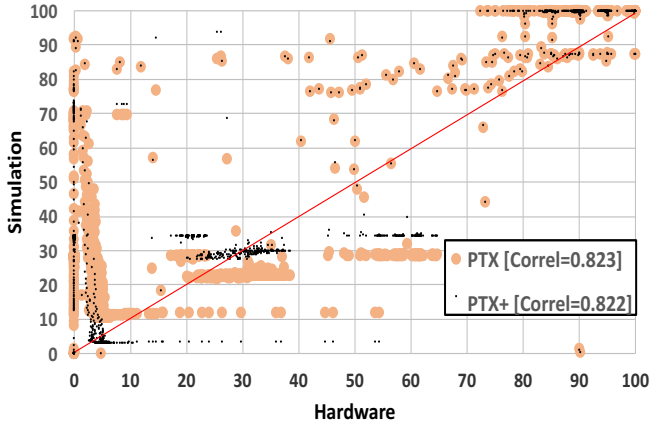


Fig. 23. GTX480: L2 cache hit rate correlation

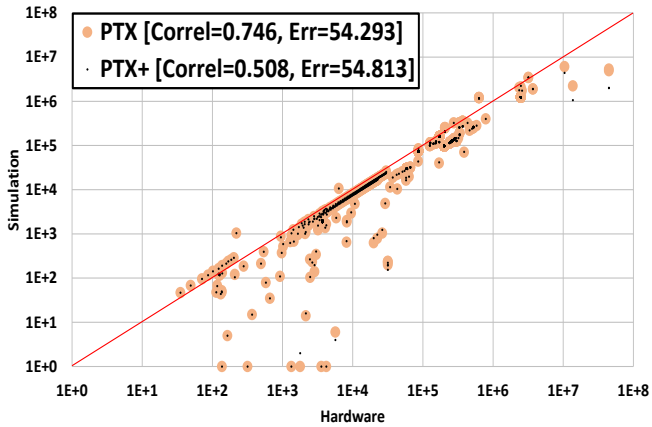


Fig. 24. GTX480: DRAM access count

ACKNOWLEDGMENTS

We thank our shepherd Prof. Murali Annavaram and the anonymous reviewers for their insightful comments and feedback on the paper. We also thank Dr. Bradford Beckmann at AMD Research for providing the motivation to pursue this study and for his valuable feedback during its execution.

REFERENCES

[1] 2011. GPGPU-Sim 3.x manual. http://gpgpu-sim.org/manual/index.php/Main_Page
[2] 2017. CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[3] 2018. CORREL function. https://support.office.com/en-us/article/CORREL-function-995dcef7-0c0a-4bed-a3fb-239d7b68ca92
[4] 2018. PTX ISA :: CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/parallel-thread-execution/index.html
[5] AMD. 2015. The AMD gem5 APU Simulator: Modeling Heterogeneous Systems in gem5. http://www.gem5.org/wiki/images/f/fd/AMD_gem5_APU_simulator_micro_2015_final.pptx
[6] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software, 2009. ISPASS 2009.*
[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer*

Architecture News 39, 2 (2011), 1–7.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[9] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA) (HPCA '13)*. IEEE Computer Society, Washington, DC, USA, 1–12. https://doi.org/10.1109/HPCA.2013.6522302

[10] Doug Burger and Todd M Austin. 1997. The SimpleScalar tool set, version 2.0. *ACM SIGARCH computer architecture news* 25, 3 (1997), 13–25.

[11] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. [n. d.]. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*.

[12] Sylvain Collange, Marc Daumas, David Defour, and David Parello. 2010. Barra: A Parallel Functional Simulator for GPGPU. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*. IEEE Computer Society, Washington, DC, USA, 351–360. https://doi.org/10.1109/MASCOTS.2010.43

[13] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*.

[14] Gregory Frederick Diamos, Andrew Robert Kerr, Sudhakar Yalamanchili, and Nathan Clark. 2010. Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*.

[15] John H Edmondson, David B Glasco, Peter B Holmqvist, George R Lynch, Patrick R Marchand, and James Roberts. 2013. Cache and associated method with frame buffer managed dirty data pull and high-priority clean mechanism. US Patent 8,464,001.

[16] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[17] HSA Foundation. 2016. HSA Standards to Bring About the Next Level of Innovation. http://www.hsafoundation.com/standards/

[18] Wilson WL Fung, Ivan Sham, George Yuan, and Tor M Aamodt. 2007. Dynamic warp formation and scheduling for efficient GPU control flow. In *MICRO*.

[19] Xun Gong, Rafael Ubal, and David Kaeli. 2017. Multi2Sim Kepler: A detailed architectural GPU simulator. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 153–154.

[20] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. 2012. Auto-tuning a high-level language targeted to GPU codes. In *Innovative Parallel Computing (InPar), 2012*.

[21] Anthony Gutierrez, Bradford M. Beckmann, Alexandru Dutu, Joseph Gross, John Kalamatianos, Onur Kayiran, Michael LeBeane, Matthew Poremba, Brandon Potter, Sooraj Puthoor, Matthew D. Sinclair, Mark Wyse, Jieming Yin, Xianwei Zhang, Akshay Jain, and Timothy G. Rogers. 2018. Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level. In *24th IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2018*.

[22] Jer Huang and Tzu-Chin Peng. 2002. Analysis of x86 instruction set usage for DOS/Windows applications and its implication on superscalar design. *IEICE Transactions on Information and Systems* 85, 6 (2002), 929–939.

[23] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. 2012. Characterizing and improving the use of demand-fetched caches in GPUs. In *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 15–24.

[24] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *arXiv preprint arXiv:1804.06826* (2018).

[25] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *proc. of ISCA*.

[26] Samuel Liu, John Erik Lindholm, Ming Y Siu, Brett W Coon, and Stuart F Oberman. 2010. Operand collector architecture. US Patent 7,834,881.

[27] André Lopes, Frederico Pratas, Leonel Sousa, and Aleksandar Ilic. 2017. Exploring GPU performance, power and energy-efficiency bounds with Cache-aware Roofline Modeling. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 259–268.

[28] Xinxin Mei and Xiaowen Chu. 2017. Dissecting GPU memory hierarchy through microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2017), 72–86.

[29] Paulius Micikevicius. 2011. Local memory and register spilling. *NVIDIA Corporation* (2011).

[30] Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, and Henri Bal. 2014. A detailed GPU cache model based on reuse distance theory. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 37–48.
[31] NVIDIA. 2009. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf .
[32] NVIDIA. 2011. CUDA C/C++ SDK Code Samples. http://developer.nvidia.com/cuda-cc-sdk-code-samples.
[33] NVIDIA. 2012. NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110. nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf. (2012).
[34] NVIDIA. 2015. Pascal L1 cache. https://devtalk.nvidia.com/default/topic/1006066/pascal-l1-cache/.
[35] NVIDIA. 2016. Pascal P100. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.
[36] NVIDIA. 2016. Pascal P102. https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf.
[37] NVIDIA. 2017. Pascal Titan X. https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/.
[38] NVIDIA. 2017. Pascal Tuning. https://www.olcf.ornl.gov/wp-content/uploads/2017/01/SummitDev_Pascal-Tuning.pdf.
[39] University of British Columbia. 2018. GPGPU-Sim Public Github. https://github.com/gpgpu-sim/gpgpu-sim_distribution/tree/dev.
[40] Timothy G Rogers, Mike O'Connor, and Tor M Aamodt. 2012. Cache Conscious wavefront scheduling. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 72–83.
[41] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture-ISCA*, Vol. 13. Association for Computing Machinery, 23–27.
[42] JEDEC Standard. 2013. GDDR5X. *JESD232A* (2013).
[43] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* 127 (2012).
[44] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*.
[45] Purdue University. 2018. GPGPU-Sim Correlation Project. https://engineering.purdue.edu/tgrogers/group/correlator.html.
[46] Purdue University. 2018. GPGPU-Sim Simulations Github Repository. https://github.com/tgrogers/gpgpu-sim_simulations.
[47] W.J. van der Laan. 2010. Decuda and cudasm, the CUDA binary utilities package. https://github.com/laanwj/decuda
[48] Vasily Volkov and James W. Demmel. 2008. Benchmarking GPUs to Tune Dense Linear Algebra. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC '08)*. IEEE Press, Piscataway, NJ, USA, Article 31, 11 pages. http://dl.acm.org/citation.cfm?id=1413370.1413402
[49] Henry Wong, M-M Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. 2010. Demystifying GPU microarchitecture through microbenchmarking. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*. IEEE, 235–246.

Received February 2018; revised April 2018; accepted June 2018