Principal Kernel Analysis: A Tractable Methodology to Simulate Scaled GPU Workloads

ABSTRACT

Simulating all threads in a scaled GPU workload is impossible due to prohibitive simulation cost. As cycle-level simulation will always be several orders of magnitude slower than native silicon, the only tractable solution is to reduce the amount of work simulated while maintaining an accurate representation of the original program.

Existing solutions to simulate GPU programs either scale the input size, simulate only the first several billion instructions, or simulate a portion of both the GPU and the workload. These solutions lack validation against scaled systems, produce unrealistic contention conditions and frequently miss critical code sections. Existing sampling mechanisms proposed for CPUs, like SimPoint, focus on reducing the work done by each thread, and are ill-suited to GPU programs where reducing the number of threads is critical. Sampling solutions in the GPU space lack silicon validation, require per-workload parameter tuning, and do not scale to large workloads.

A tractable solution, validated on contemporary scaled workloads, is needed to provide credible simulation results. By studying scaled workloads with centuries-long simulation times, we uncover practical and algorithmic limitations of existing solutions and propose Principal Kernel Analysis: a hierarchical program sampling methodology that concisely represents GPU programs by selecting representative kernel portions using a scalable profiling methodology, tractable clustering algorithm and detection of intra-kernel IPC stability. We validate Principal Kernel Analysis on 147 workloads from 6 benchmark suites and three GPU generations using the Accel-Sim simulator, demonstrating a better performance/error tradeoff than prior work and that century-long MLPerf simulation times can be reduced to hours with an average cycle error of 27% versus silicon.

> There is no real ending. It's just the place where you stop the story.
>
> *Frank Herbert*

1. INTRODUCTION

Simulators carry out a cycle-accurate introspection of GPU workloads. These simulators are highly configurable and enable different analyses such as (i) debugging of workloads on specific simulated hardware to detect implementation errors [9], (ii) profiling of workloads to analyze performance bottlenecks [1, 6], or (iii) reconfiguration of the simulated hardware to analyze model changes [2]. Many such use cases are impossible in silicon due to limited profiling capabilities [37].

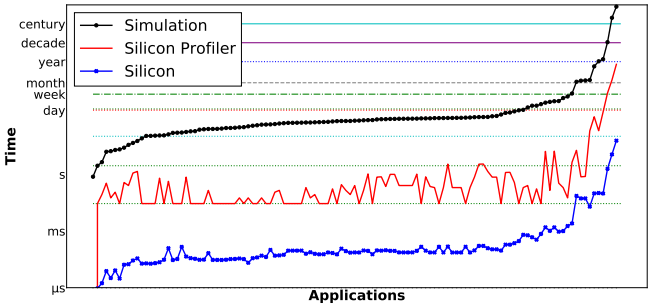


Figure 1: Projected hours to simulate, profile 12 statistics (Table 2) in-silicon [37] and raw execution time of the 147 workloads we study (Section 4) on a Volta V100. Projected simulation times are based on Accel-Sim's [30] simulation rate for each app.

The flexibility and introspection capabilities of simulation come at a price. Simulating complex hardware like modern GPUs incurs orders of magnitude of overhead for each simulated instruction. What takes seconds on a GPU would take millennia on a simulator. Due to the overhead in simulation, existing simulators cannot reasonably execute real GPU programs. Even high-performance industrial simulators used by GPU companies are still not fast enough to simulate applications that take seconds to minutes on contemporary GPUs [56]. *Any simulation platform must therefore restrict the number of executed instructions.*

Existing approaches restrict either the workload, the program, or the underlying simulation platform. Common approaches include: (i) scaling workload inputs [12, 21, 51], which reduces the applicability of the simulation results due to the extremely short runtimes (ignoring scaling effects), (ii) simulating the first several billion instructions of a scaled workload [8, 30, 55], which restricts the insights to a limited horizon of simulation (often limiting measurements to the warmup phase), and (iii) reducing the size of the GPU simulated [27, 38, 44], which forces the workload to adapt to different hardware. Each of these methodologies has limitations, and there is no public work validating them against the scaled workloads and systems they aspire to represent. Furthermore, no prior work has attempted to simulate or validate the truly representative large-scale workloads from suites like MLPerf [34, 43] to completion.

To demonstrate the extent of workload realism and simulation slowdown, Figure 1 plots the silicon execution

time and projected simulation time for 147 workloads from both contemporary GPU application suites typically used in simulation and 7 applications from the MLPerf benchmark suite where the datasets were publicly available. Benchmark authors adjusted the size of traditional workloads so that simulations complete in a matter of hours or days. However, these workloads execute in microseconds in silicon, rendering them impractical. The realistic workloads we study from MLPerf take several seconds to run in real silicon, and centuries to simulate. A *representative portion of the scaled GPU program* must be selected to make simulation practical.

Traditional mechanisms to select representative portions of CPU programs [11, 20, 22, 40, 58, 60] focus on selecting basic blocks from a single thread of execution. This approach is ill-suited to GPU programs. The control-flow graphs of individual threads in GPU programs are small since each thread performs a limited amount of work compared to a CPU thread. As a result, selecting per-thread basic block vectors without curtailing the number of threads does not significantly reduce simulation time. Prior work on selecting representative portions of GPU programs [24, 61, 62] lack silicon validation, require per-workload parameter tuning and do not scale to larger workloads.

Intelligent solutions like TBPoint [24] require full functional simulation to produce threadblock-level profiling information and rely on inter-kernel clustering mechanisms that does not scale to modern workloads such as MLPerf. As a result, no GPU workload sampling methodology has achieved general acceptance. In addition, no existing work has evaluated the practical implications of silicon profiling at scale. Figure 1 quantifies the slowdown experienced by detailed in-silicon profiling using NVIDIA's latest profiling tools [37]. Collecting even a limited number of statistics from long-running workloads quickly becomes impractical and any sampling methodology that relies on detailed profiling of the entire workload does not scale to contemporary workloads.

We propose an automated *Principal Kernel Analysis* (PKA) methodology which reduces the number and length of kernels used to represent fully scaled GPU applications. We base *Principal Kernel Analysis* on three key observations. First, even though realistic workloads can launch millions of kernel instances (5.3 million in MLPerf's SSD Training), these kernels can be characterized and grouped by a set of architecture-independent metrics using principal component analysis obtained from detailed in-silicon profiling.

Second, detailed, in-silicon profiling is impractical in truly scaled workloads and a hierarchical profiling mechanism is needed. For workloads with impractical silicon-profiling times, PKA performs detailed profiling on a subset of the application's kernels and lightweight profiling on the rest. Using a variety of classifying algorithms (Stochastic Gradient Descent, Gaussian Naive Bayes, Multi-layer Perceptrons) PKA maps the lightly profiled kernels into the groups identified in the detailed profiling phase. Using this per-kernel analysis, we perform *Principal Kernel Selection* to extract the minimum set of kernels necessary to obtain a target error on the projected execution-time.

Our third observation comes from the behavior of individual kernels. We observe that the instantaneous Instructions Per Cycle (IPC) within one kernel often stabilizes around a value that will be representative of the kernel's final IPC. Borrowing a method from the financial analysis sector that attempts to predict stock price stabilization over time [18], we track the standard deviation of a kernel's IPC throughout simulation. Once our online stabilization calculation reaches an appropriate confidence interval, we use occupancy information about the running kernel to make a *Principal Kernel Projection* based on the amount of work remaining. Our online mechanism can be executed quickly in simulation and validated against lightweight silicon profiling.

To evaluate the effectiveness of *Principal Kernel Analysis* and its effect on accuracy, we apply our selection and projection mechanisms to the cycle-level GPU simulator Accel-Sim [30]. Using silicon profiling data, we select a workload's representative kernels. These kernels are then simulated until IPC stabilization is detected in the simulator, at which point the resultant statistics are projected.

This work makes the following contributions:

1. We perform the first silicon-verified analysis on the validity of simulation on scaled GPU workloads, identifying characteristics we can exploit to create concise representations of real GPU programs, reducing their simulation time.

2. We introduce *Principal Kernel Selection* an inter-kernel, architecture-independent principal component analysis that automatically clusters kernel instances with similar behavior. Using metrics obtained from two-level silicon profiling, we select a representative subset of kernels that we use to project the entire application's behavior. We demonstrate that the kernels we select from profiling one GPU generation generalize across Volta, Turing, and Ampere.

3. To reduce intra-kernel simulation time with low-overhead, we leverage an observation that the instantaneous IPC in many scaled, real-world kernels stabilizes near its final average. Inspired by methods that predict stock-price stability, we propose *Principal Kernel Projection*, which detects IPC stability and projects per-kernel metrics based on occupancy characteristics.

4. We propose a fully automated characterization and simulation methodology *Principal Kernel Analysis* that combines principal kernel selection and projection. Evaluated on Accel-Sim using 147 workloads, we demonstrate that PKA greatly reduces simulation time, while maintaining an error rate close to the baseline simulator. Centuries-long simulation times from MLPerf are reduced to hours with an average cycle error of 27% versus silicon execution of the scaled workloads.

Table 1: Landscape of Sampled Simulation Literature.

Sampling Methodologies	Control-Flow Reduction [22, 40], [49, 58, 60]	Synchronization Regions [11, 20]	GPGPU -MiniBench [61, 62]	GT-Pin [28]	TBPoint [24], Clustering [19]	***Principle Kernel Analysis***
Threaded	Single	CPU Multi-Threaded	GPU Multi-Threaded	GPU Multi-Threaded	GPU Multi-Threaded	GPU Multi-Threaded
Mechanism	Identify common basic blocks	Inter-barrier regions	Intra-threadblock control flow analysis	Unique kernels & control flow analysis	Threadblock reduction [24], kernel clustering	Threadblock/kernel reduction
Inter-kernel	NA	NA	✗	✓	✓	✓
Intra-kernel	NA	NA	✓	✗	[24]* Requires full functional simulation	✓
Sampling Clustering	Automated	Automated	Automated	Automated	Hierarchical hand-tuned	Automated
# GPU Workloads	NA	NA	23	25	12	147
Silicon Validated vs Century-Long Full-Simulation	✗	✗	✗	✗	✗	✓

2. BACKGROUND AND MOTIVATION

Characterizing programs to reduce simulation time through sampling techniques is a decades-old research area. Table 1 presents a survey of sampling techniques proposed for single-threaded CPU [22, 40, 58, 60], multithreaded CPU [11, 20, 33] and GPU [24, 61, 62] applications. Fundamental differences in the nature of CPU programs makes the direct application CPU techniques to GPUs difficult. CPU programs contain, at most, tens of threads. These techniques focus on reducing the work done by each thread, reducing the scope of the dynamic control-flow graph [22, 33, 40, 58, 60] or selecting portions of each thread between synchronization points [11, 20], without decreasing the number of threads.

Prior work has also explored sampling GPU applications. Kambadur et al. introduce GT-Pin [28], a dynamic binary instrumentation tool for OpenCL workloads on Intel GPUs that can be used to select representative portions of GPU programs. Using a clustering algorithm based on kernel name, arguments and basic block statistics, GT-Pin selects representative portions of the program with a kernel as the smallest granularity. In contrast, *Principal Kernel Analysis* focuses on both inter- and intra-kernel reduction, while basing its inter-kernel clustering on a name-independent feature-vector.

Zhibin et al. [61, 62] proposed GPGPU-MiniBench in which they profile an application's control-flow divergence. Their analysis is similar to SimPoint [22] in that they analyze and define the minimum number of intra-threadblock loops needed to represent a kernel. However, the MiniBench analysis does not address inter-kernel and inter-threadblock execution reduction and involves the creation of proxy-applications. Their analysis was also done on legacy workloads.

More closely related to PKA is TBPoint [24], which uses statistical modeling and pure-simulation results to reduce both the number and length of kernels used to represent a workload. Using 12 legacy workloads and statistics gathered from full functional simulation in GPUOcelot [16], they propose a hierarchical clustering mechanism to group and reduce the number of kernels in the program representation. Similarly, Goswami et al. [19] propose a hierarchical clustering mechanism.

Although TBPoint attacks both inter- and intra-kernel reduction, the mechanisms do not scale to the century-long workloads we study in this paper, and require per-application parameter tuning. To reduce the number of kernels simulated, TBPoint performs hierarchical clustering on a feature vector derived from simulation. To reduce intra-kernel runtimes, per-threadblock simulation statistics are required for the entire kernel. Although precise, TBPoint cannot be applied to applications that cannot be fully simulated. To handle contemporary, scaled workloads we argue that the analysis must be done online, then validated against silicon execution. To demonstrate the efficacy of PKA on workloads where TBPoint is tractable, we perform a quantitative comparison against TBPoint in Section 5.

Despite the well-reasoned related work in this space, a GPU equivalent of SimPoint has yet to receive widespread adoption. We aim to fill this gap by introducing the *Principal Kernel Analysis* methodology and toolset [1], designed to have the following characteristics:

1. **Scalable:** By using two-level profiler data from real silicon execution of full-scale workloads, *Principal Kernel Analysis* is able to characterize workloads that would be untenable using prior GPU-centric sampling techniques like TBPoint.

2. **Automatic:** The inputs to *Principal Kernel Analysis* are the profiled results from silicon, a desired maximum error from the *Principal Kernel Selection* phase and a confidence interval for the *Principal Kernel Projection* phase. For all the applications we study, we apply the same desired error and confidence interval such that no per-workload tuning of opaque clustering parameters is required.

3. **Tunable:** There is always a tradeoff between simulation time and simulation fidelity. Allowing *Principal Kernel Selection* and *Principal Kernel Projection* are tuned according to desired error and confidence interval respectively.

[1]Fully-automated scripts, profiler data and simulator integration will be open-sourced upon the paper's publication

Table 2: A list of hardware-independent characteristics collected for PCA analysis.

Metric	Nsight metric name
Coalesced global loads	l1tex sectors pipe lsu mem global op ld.sum
Coalesced global stores	l1tex sectors pipe lsu mem global op st.sum
Coalesced local loads	l1tex sectors pipe lsu mem local op ld.sum
Thread global loads	smsp inst executed op global ld.sum
Thread global stores	smsp inst executed op global st.sum
Thread local loads	smsp inst executed op local ld.sum
Thread shared loads	smsp inst executed op shared ld.sum
Thread shared stores	smsp inst executed op shared st.sum
Thread global atomics	smsp sass inst executed op global atom.sum
#Instructions	smsp inst executed.sum
Divergence efficiency	smsp thread inst executed per inst executed.ratio
#Threadblocks	launch grid size

4. **Verified Against Silicon:** Simulator versus simulator validation limits the workloads that can be validated to those possible to simulate in their entirety. Applying PKA to Accel-Sim, we compare sampled simulation results to real silicon results, demonstrating that an open-source simulator can achieve an acceptable absolute and relative error on real workloads.

3. Principal Kernel Analysis

In this section, we introduce our two-level hierarchical method to create concise representations for GPU programs and speedup simulation. There are two different families of techniques to speedup GPU simulation, inter-kernel and intra-kernel, one acting at kernel-granularity and the other at thread block-granularity. Our first method, called *Principal Kernel Selection*, depicted in Figure 2a, uses several hardware-agnostic metrics to cluster similar kernels together and simulates only one representative kernel per cluster. The second method, called *Principal Kernel Projection*, depicted in Figure 2b, reduces the number of simulated thread blocks in the grid by detecting IPC stability. Figure 2c illustrates the mechanism of *Principal Kernel Analysis* when combining both techniques together to reduce the number of both simulated kernels and threadblocks.

3.1 Principal Kernel Selection

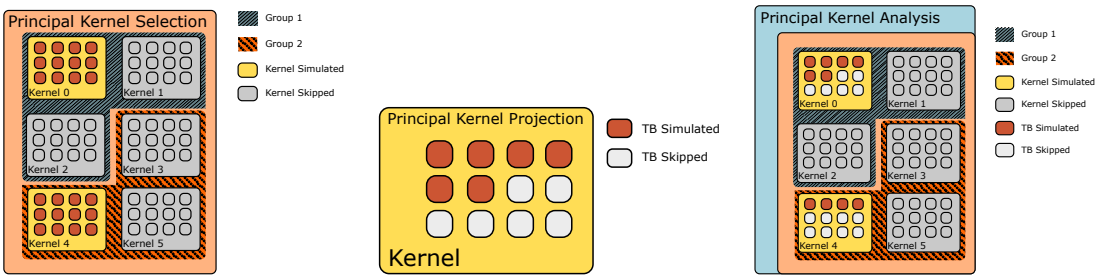
To reduce the number of simulated kernels, we group similar kernels together and only simulate the most representative (or principal) kernels. To achieve this, we profile each application in silicon. We tell the profiler only to report certain hardware-agnostic features, such as the number of global loads, stores, and atomic operations (Table 2). Note that these statistics are dependent only on the generated GPU code, not the specific GPU being profiled. One caveat is that different GPU generations use different machine ISA representations, therefore; the number of instructions and makeup of specific instructions can vary slightly across generations. We note that classic CPU methodologies have a similar issue between the x86 representation of the program and the micro-ops used by the pipeline. To further demonstrate that these metrics hold across different GPU generations, Section 5 evaluates the efficacy of *Principal Kernel Selection* across the three most recent NVIDIA GPU generations (Volta, Turing and Ampere). In our evaluation, *Principal Kernel Selection* is performed for Volta and those same representative kernels are selected to project the execution time in Turing and Ampere, without running *Principal Kernel Selection* on each machine.

Since GPGPU workloads may have different categories and special characteristics, we perform non-supervised machine learning techniques—Principal Component Analysis (PCA) and K-Means—to reduce these hardware-agnostic features' dimensions to a more manageable number from the components in Table 2. The PCA data are then clustered using K-Means. As a result, K groups of similar kernels are formed. It should be noted that clustered kernels often do not have the same name, groups are usually composed of several instances of differently named kernels. For each of these K groups, a single representative kernel is chosen. Because this kernel summarizes the entire group, we scale its runtime (in cycles) by the number of kernels (elements) to obtain a projection of each group's total runtime. This process is aggregated across all groups to obtain a projection for all the kernels in the program.

Algorithm for choosing K-Groups: Part of the reason why we choose PCA+K-Means is explain-ability. By applying PCA, we can trivially consider a broader set of characteristics. We know that the principal dimensions will have the most variance. With K-Means, we can directly change the number of groups. By combining PCA and K-Means, we avoid the curse of dimensionality when clustering. Another benefit is that the K parameter represents a less abstract notion than other clustering techniques, like sigma and hierarchy clustering used by TBPoint [24]. Most importantly, k-means clustering can scale to the millions of kernels in our large workloads, where hierarchical clustering demands an impractical amount of memory and runtime.

By varying the K parameter in K-means, the trade-offs are apparent. We start by sweeping across different values of K, typically from 1 to 20, and generating different clustering configurations. For each clustering configuration, we find the most representative kernels. We scale the number of cycles of each representative kernel by the number of elements in the group and aggregate them to obtain a total runtime projection. We end with K different projections. For each projection, we calculate its error with respect to the total silicon number of cycles. The smallest K value whose projected error falls below a threshold is chosen. A smaller K is preferred, since fewer groups results in a greater reduction in the number of kernels required to represent the program. The only input the user gives to the process is the desired execution time error. In all the data we present, we set the *Principal Kernel Selection* cycle error threshold to 5%. Less error will require more groups. Therefore, linked to choosing the number of groups is selecting the most representative kernel within each group.

To determine which kernels should be selected as the principal one we experimented with random selection,



(a) *Principal Kernel Selection* mechanism.

(b) *Principal Kernel Projection* mechanism.

(c) *Principal Kernel Analysis* mechanism when combined both *Principal Kernel Selection* and *Principal Kernel Projection*

Figure 2: An illustration of *Principal Kernel Analysis* by reducing the number of executed kernels and threadblocks.

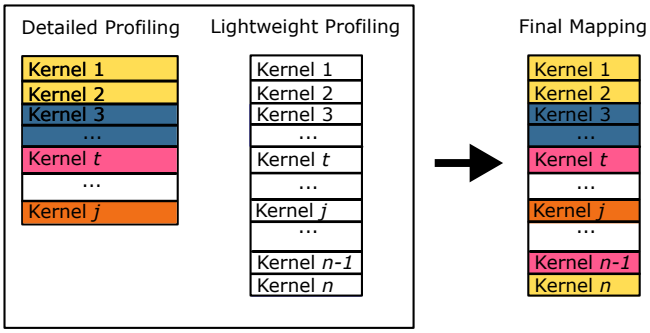


Figure 3: The hierarchical profiling mechanism

selecting the closest to the center and selecting the first chronologically. We empirically determined that random selection has a high error. In contrast, the performance difference between choosing the cluster-center and first-chronologically kernels to be mostly negligible. Selecting the first chronological kernel has practical advantages in reducing tracing and profiling times, and we utilize this methodology for *Principal Kernel Selection*.

Two-level profiling: For workloads where detailed silicon profiling is intractable (i.e. if the profiling takes more than one week), we propose a novel two-level profiling approach. Figure 3 shows a visualization of our approach. We perform detailed profiling on the first j kernels and create our k-groups based on their characteristics (i.e. we apply *Principal Kernel Selection*). For the remaining kernels, we profile them using low-overhead profiler (Nsight-systems), where only the kernel name and grid dimensions are collected. We determined that utilizing this small amount of information was not enough to train the classifiers, so we supplemented the data using Nvidia's PyProf, a tool that provides extra information for PyTorch workloads, like tensor dimensions, program traces, which layer launched any particular kernel, etc, etc. The previously identified k groups were transferred from the detail-profiled data to the low-overhead one. We use three different classification models (Stochastic Gradient Descent, Naive Bayes Gaussian, and Multi-Layer perceptrons) to map between the groups and the augmented data. For this technique to work, we need to make sure that the kernels from the Nsight Compute and Nsight Systems are the same. We ensure this happens by checking for same names, grid sizes, (and if available same PyProf details).

Table 3: An example of *Principal Kernel Selection* analysis output for sample workloads. The last two columns show the set of kernel IDs selected to represent each group and the number kernels per group.

Suite	Workload	Selected Kernel ID	Group Count
Rodinia	gaussian_208	0	414
	bfs 65k	0	20
Parboil	histogram	0,1,2,3	20,20,20,20
	cutcp	0,1,2	2,3,6
Polybench	fdtd2d	0,2	1000,500
	gramschmidt	0, 1, 2, 1439, 2783, 4127	2048, 2273, 479 448, 448, 448
Cutlass	2560 x 128 x 2560 wmma	0	7
	4096 x 4096 x 4096 sgemm	0	7

Group Selection Examples: Table 3 depicts an example of *Principal Kernel Selection* analysis output for a few selected workloads with a target error of 5%. In the example the analysis of the application *gaussian* clusters 414 similar kernels in one only one group. In this scenario, only one kernel (kernel id=0) is selected. In *gramschmidt*, the clustering analysis determines six different groups are necessary (ranging in size from 448 kernels to 2273 kernels), and thus six representative kernels (out of 6411 kernels) are selected.

In Figure 4, we depict the per-group kernel composition after applying *Principal Kernel Selection* to the long-running ResNet 5.0 workload from the MLPerf suite. As shown in the figure, we end up with nine different groups and each group contains hundreds of kernels. It worth mentioning that kernels in the same cluster have different names/code implementations. We notice that compute-intensive kernels (e.g., convolution operations and fully connected layers) are combined in the same group, whereas memory-intensive kernels (e.g., element-wise operations) are clustered in the same group. In

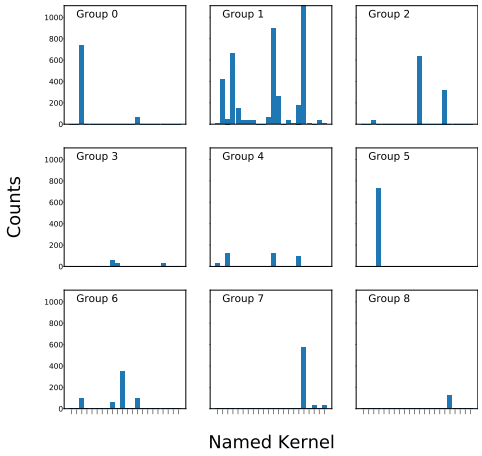
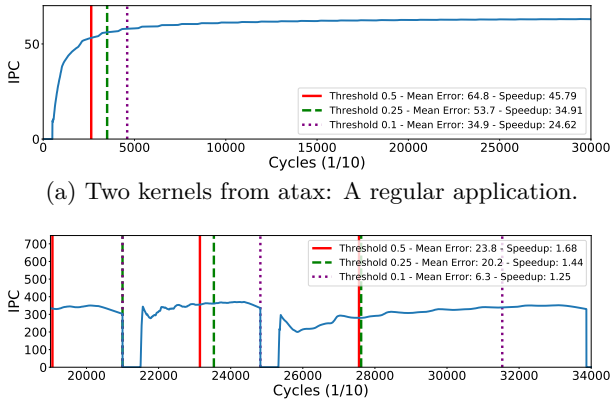


Figure 4: Per-group kernel composition after applying PKS to ResNet 5.0. The y-axis shows the number of kernel launches for each kernel instance.



(a) Two kernels from atax: A regular application.

(b) Three kernels from BFS: An irregular application.

Figure 5: IPC versus time for a subset of kernels from two applications. Lines indicate Principal Kernel Projection stopping points using different s thresholds.

addition, some kernels with the same name are sorted into different groups. This often happens when the kernel is launched several thousand times with different grid and/or threadblock dimensions. Note that since we use unsupervised learning to create this clustering, these groupings naturally fell out from silicon profiling.

3.2 Principal Kernel Projection

Principal Kernel Selection only addresses the number of kernels an application runs; long-running kernels may still be a bottleneck. To reduce the runtime of individual kernels, we introduce *Principal Kernel Projection*. The key insight exploited in *Principal Kernel Projection* is that each thread in the grid executes the same code and the code tends to have few phases, as the lifetime of threads are much shorter than their CPU peers. As a result, we have observed that the IPC of most GPU kernels stabilizes around the final average, even in some irregular applications like graph processing. Therefore, *Principal Kernel Projection* is designed to detect a stable IPC when it occurs, ending simulation then and projecting final statistics.

To detect stability, we calculate the rolling average and standard deviation of the last n cycles (we use 3000 across all our workloads). If the standard deviation falls below another user-defined variable s, the IPC is considered quasi-stable. The standard deviation variable s is the only user-facing input variable to *Principal Kernel Projection* and the user can select a value of s to reflect the confidence interval desired in IPC stabilization. To show that *Principal Kernel Projection* does not require extensive hand-tuning to generate reasonable results, we select an s value of 0.25 for all our experiments. A smaller value will increase the confidence that the IPC has stabilized at the expense of more simulation time.

To ensure resource contention is properly captured, we impose an additional constraint on classifying the signal as stable; the number of finished thread blocks must be more than the amount that fills the GPU's cores i.e. enough thread blocks to reach the highest-possible occupancy of the kernel in question. We call this quantity a wave. Once enough thread blocks have finished to complete the wave where quasi-stability occurs, we consider the signal stable. To project the number of cycles it would take to finish the kernel, we take the number of unfinished thread blocks and linearly project the number of cycles left. If a kernel launches less threadblocks then a wave, we ignore this conditional and stop the kernel as soon as stability occurs. Since kernels with few CTAs do not experinece CTA ending/beginning phases, we find that removing this condition for low-CTA kernels results in acceptable error.

To illustrate the operation of *Principal Kernel Projection*, Figure 5 presents a visualization of application IPC versus time for a regular workload (ATAX in Figure 5a) and an irregular workload (BFS in Figure 5b). The stopping conditions for *Principal Kernel Projection* at different s values are indicated with vertical lines. The results in the regular workload are unsurprising in that it quickly ramps up to its peak IPC value and stay there for the duration of each kernel. For each of the three kernels shown for the irregular workload, the results are more surprising. Despite having significant control and memory divergence, over time the IPC of BFS does stabilize. Despite the fact that each thread in the system is performing different amounts of work, in the aggregate a collision of irregularity in all the threads results in stability. Figure 5 also illustrates the effect different threshold values have on the stopping point of *Principal Kernel Projection*. We empirically find that 0.25 results in a good compromise between accuracy and speedup across all our apps.

4. MEASUREMENT SETUP

To evaluate *Principal Kernel Analysis* we combine experiments on both silicon and simulation, using three generations of NVIDIA GPUs. *Principal Kernel Selection* requires silicon profiling to infer kernel similarity. For detailed and lightweight profiling, we use Nsight Compute and Nsight Systems respectively [37]. We

evaluate *Principal Kernel Selection* in both silicon and simulation, while we show *Principal Kernel Projection* for simulation only. We obtain the principal kernels by applying *Principal Kernel Selection* to a V100 [14]. Once identified, we use these kernels to evaluate their effectiveness on a Turing RTX 2060 [10] and an Ampere RTX 3070 [13]. To evaluate the both selection and projection, we also conduct simulation experiments using Accel-Sim [1, 7, 30, 57].

We evaluate our technique with the complete benchmark sets of Rodinia [12], Parboil [51], Polybench [21], the machine learning suite DeepBench [3], and the GeMM-based CuTLASS [36] benchmark suite. We also evaluate the subset of the reference implementations of the applications in MLPerf [43] for which we could get realistic datasets and confirm correct functionality in silicon. Specifically those are: ResNet [23] using the ImageNet dataset [45], SSD [32] using the COCO dataset [31], GNMT [59] using the German and English euro database, BERT [15] using the SQUAD dataset [42], the medical imaging 3D-Unet [26] using the BRATS dataset [35] [4] [5]. Every benchmark application was compiled using CUDA 11.1 and cuDNN 8.0.2.

5. EVALUATION

We stipulate that the combination of *Principal Kernel Selection* and *Principal Kernel Projection* drastically reduce the expected simulation time with an acceptable loss of accuracy. In this evaluation section we set out to answer the following research questions:

RQ1: How accurately do *Principal Kernel Selection* and *Projection* individually and together predict performance and reduce simulation time and how do they compare to prior work (Section 5.1)?

RQ2: How do the characteristics of applications effect the efficacy of *Principal Kernel Analysis*, and do our results generalize across architctures (Section 5.2)?

RQ3: Given that architects care most about the relative accuracy of a simulator, how well do *Principal Kernel Selection* and *Projection* predict the relative performance of the different architectures we study (Section 5.3)?

In section 5.1, we analyze and compare the achieved simulation speedup and accuracy of *Principal Kernel Analysis* versus: (1) a commonly-used technique to only simulate the first 1 billion instructions, and (2) state-of-the-art sampled simulation of GPU applications, TBPoint [24] in simulation using Accel-Sim [30]. In Section 5.2, we perform a deep-dive into the data for each application suite and configuration we study in both silicon and in simulation. In section 5.3 we present two case studies of how *Principal Kernel Analysis* predicts the relative performance of different architectures.

5.1 Overall Effectiveness

In this subsection, we compare the efficacy of PKA to prior work in simulation using Accel-Sim modeling an

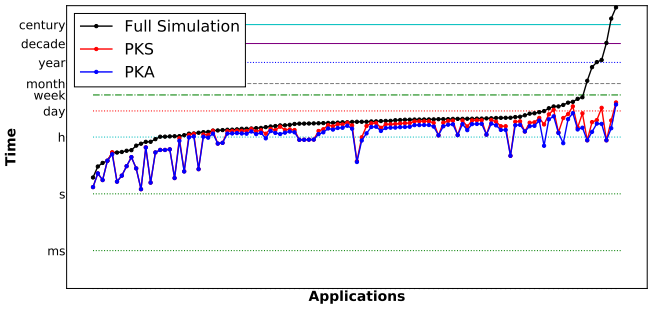


Figure 6: Simulation time using full simulation, PKS and PKA. Y-axis is log scale hours.

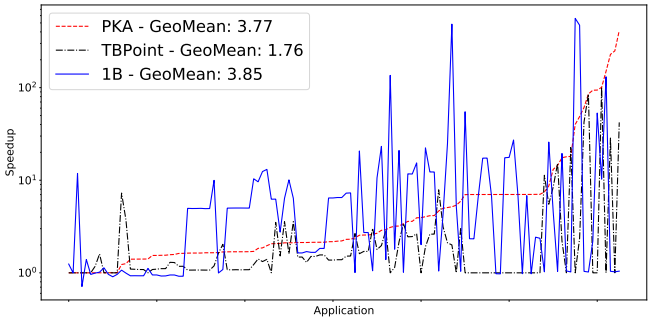


Figure 7: Simulation speed up of PKA, TBPoint and 1B instructions over complete simulation in all the applications that complete in full simulation.

NVIDIA Volta V100. Figure 6 plots the simulation times we achieve versus the original simulation times originally shown in Figure 1 across our benchmarks sorted by their simulation time. The figure demonstrates that *Principal Kernel Analysis* is able to reduce the simulation time of every workload we study from centuries to less than one week. The figure demonstrates that many applications see significant reduction from PKS. The effect of the intra-kernel reduction is more skewed. There is a significant constant-factor speedup on most of the longer-running workloads (reducing simulation time from days to hours in some cases). However, the bulk of the reduction in simulation comes from PKS. Most complex applications are divided into multiple kernel launches, hence PKS shows the most benefit when application runtimes are long. The full error results with this speedup are shown in Table 4 and discussed in Section 5.2.

To evaluate the effectiveness of PKA compared to previous work, Figure 7 plots the reduction in cycles of PKA, TBPoint [24], and the commonly used practice of executing the first 1 billion instructions. Only the applications that complete in full simulation (and are hence possible using TBPoint) are plotted here. In lieu of the hand-tuned threshold setting the TBPoint original implementation required, our implementation of TBPoint sweeps across 20 threshold values between 0.01 and 0.2 and follows the same criteria *Principal Kernel Selection* does to decide the best. In these classic workloads, PKA is able to reduce the number of cycles almost as much as the simple, high-error mechanism

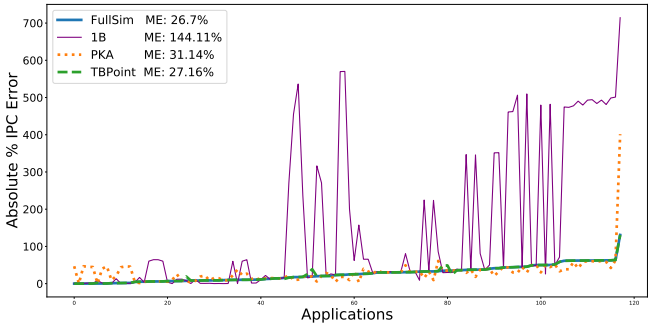


Figure 8: Accel-Sim simulation error using full simulation, 1B instructions, PKA and TBPoint.

of executing the first 1 billion instructions. Although TBPoint is able reduce simulation time significantly, it requires 2.19× more simulation than PKA.

Figure 8 plots the absolute IPC error for the same three mechanisms, sorted by the baseline error of full simulation. Although 1B instructions provides a significant speedup, the error is 5.4× full simulation. TBPoint's relatively conservative reduction mechanism results in an error 0.56 points higher than full simulation, while PKA's error is 4.44 points higher. Comparing TBPoint to PKA, PKA provides a 2.19× reduction in simulation time over TBPoint for only slightly more error. While this tradeoff is appealing, the main advantages of PKA over TBPoint is the ability to evaluate scaled workloads and the automated nature of PKA's selection mechanisms. We discuss the error and speedup of PKA on the scaled MLPerf applications in Section 5.2.

5.2 Results Analysis

To help perform a side-by-side analysis of our various configurations, hardware platforms and applications, Table 4 aggregates the raw results of *Principal Kernel Analysis* in both silicon and in simulation. The first 6 columns of the table present the error and speedup results for the three GPUs we study when applying *Principal Kernel Selection* to silicon-only results. Speedup (SU) represents the reduction in total execution time that is achieved with the adjacent error. The next 5 columns are the simulated Volta error and speedup results for both *Principal Kernel Selection* alone and when combined with *Principal Kernel Projection* (i.e. PKA).

Only the Volta V100 GPU has enough memory to run the MLPerf workloads we study, hence we subdivide this section into first discussing the silicon Volta results (Section 5.2.1) using *Principal Kernel Selection* in isolation, contrasting those results with Turing and Ampere's silicon results in Section 5.2.2 and finish with a discussion of the full volta simulation results in Section 5.2.3.

5.2.1 Volta Silicon Results

In this subsection we discuss the first two columns of Table 4. The Rodinia application suite commonly used in architecture studies has inputs sized such that they finish in simulation, resulting in real execution times of under a millisecond. However, *Principal Kernel Selection* is able to significantly reduce the simulation time of many of the applications that launch a large number of kernels. Overall, the error introduced by *Principal Kernel Selection* in Rodinia is 1.6% with a geomean speedup of 7.2×. Parboil and Polybench are similar (1.3% error/5.8× speedup and 0.8% error/4.2× speedup respectively). Single-kernel applications see no benefit, while apps with many kernels see speedups of up 711.1×, with little accuracy loss.

Cutlass and the various implementations of Deepbench represent highly-tuned machine-learning kernels evaluated in isolation and are used as a reasonable proxy for the matrix-multiplication kernels found in neural networks. They are hand-tuned by NVIDIA engineers, and make use of tensor cores. Although the error rates remain low, the speedup is also muted, in comparison to the other suites (ranging between 1 and 7×). Since these applications launch fewer, targeted kernels, the effectiveness of *Principal Kernel Selection* to reduce the workload is limited.

The final suite of applications are from the MLPerf suite. Among all the considered suites, these are the longest workloads. Running the BERT inference using the Offline scenario takes roughly 10 minutes in silicon. Profiling these workloads was challenging, as the number of kernels is orders of magnitude larger than the other suites. For the vision and classification inference workloads, ResNet and 3D-Unet, a complete profiling was achievable using Nsight Compute. For the larger workloads, the hierarchichal clustering technique was used. The effects of which is clear because of the penalty said technique incurs in the mean error. The largest workload is SSD training, with 5.3 million kernels, of which 20 thousand kernels were profiled in detailed. The average error across the runs of MLPerf benchmarks is 10.0%, and the geomean speedup is 1987×.

5.2.2 Turing and Ampere Silicon Results

In this subsection, we discuss columns 3-6 in Table 4. To validate our hypothesis that the principal kernels identified using the volta are representative, regardless of architecture, we use them to evaluate *Principal Kernel Analysis*'s accuracy in other generations.

We begin by examining the results from the Rodinia,Parboil and Polybench suites. The overall error and performance trends are maintained inter-generation; if applying *Principal Kernel Selection* to a Rodinia workload yields a speedup above 400 × in Volta, we see the same in Turing and Ampere.

Turing and Ampere's performance in the Cutlass Perf Suite SGEMM presents a negligible mean error and a geomean speedup of 6×. The Tensor Core version keeps the mean error under 1% for both Turing and Ampere and yields the same speedup of 7× as the Volta card did. The reduced kernel projection of Turing and Ampere performs roughly the same for Parboil and Polybench suites as the Volta GPU did. If the selected kernels work in Volta, they work in Turing and Ampere, and the complement is also true.

The next suite of workloads is Deepbench. Start-

Table 4: Cycle error and Speedup for for Principal Kernel Selection (PKS) in Silicon and using Accel-Sim. Full Principal Kerenel Analysis (PKA) results also shown for simulation. Entries with "*" do not have data for reasons explained in Section 5. SU=Speedup (in ×). Errors are in %. H=Hours.

Application	Silicon						Simulation				
	Volta		Turing		Ampere		Volta				
	Error [%]	SU	Error [%]	SU	Error [%]	SU	SimError	PKS Error	EK SimTime [H] (SU)	PKA Error	PKA SimTime [H] (SU)
Rodinia Suite											
b+tree	0	1	0	1	0	1	5.8	5.8	0.4 H (1.0)	3.5	0.2 H (1.7)
backprop	0	1	0	1	0	1	4.3	4.3	0.1 H (1.0)	4.3	0.1 H (1.0)
bfs1MW	5	1.5	0.4	1.2	0.7	1.3	36.7	34.5	1.4 H (1.5)	12.1	1.0 H (1.7)
bfs4096	1.6	1.2	2	1.2	1.8	1.2	15.5	23.0	0.1 H (1.2)	23.0	0.1 H (1.2)
bfs65536	1.9	19.6	35.6	31.1	2.8	19.4	14.2	12.1	0.0 H (21.4)	12.5	0.0 H (22.2)
dwt2d_192	1.2	3.5	1.4	3.2	6.3	3.3	45.2	48.3	0.0 H (3.5)	48.3	0.0 H (3.5)
dwt2d_rgb	0.3	2.3	1.2	2	0.1	2	1.6	0.1	0.1 H (2.4)	0.1	0.1 H (2.4)
gauss_208	5	435.6	7.8	449	7.2	446.1	56.7	63	0.0 H (429.6)	51	0.0 H (431.1)
gauss_mat4	1.8	5.9	0.9	5.9	1.1	6.1	77.8	86.8	0.0 H (6.0)	86.8	0.0 H (6.0)
gauss_s16	2.5	14.9	2.9	14.8	0.1	14.5	73.5	84.5	0.0 H (15.0)	73.5	0.0 H (20.1)
gauss_s64	0.7	60.1	1.6	61.3	2.4	62	69.8	79.0	0.0 H (63.7)	67.9	0.0 H (74.0)
gauss_s256	0.4	226.3	8.5	167.9	3.8	232.4	53.4	65.8	0.0 H (248.0)	50.8	0.0 H (258.4)
hots_1024	0	1	0	1	0	1	3.9	3.1	0.2 H (1.0)	9.1	0.1 H (1.3)
hots_512	0	1	0	1	0	1	16.1	16.1	0.0 H (1.0)	16.1	0.0 H (1.0)
hstort_500k	4.8	4.4	6	4.6	3.9	4.4	45.1	46.5	0.3 H (4.3)	46.5	0.3 H (4.3)
hstort_r	4.6	5.6	7.8	6.6	5.9	6	49.5	47.8	2.3 H (5.6)	45.4	2.2 H (5.7)
kmeans_28k	1.4	1.6	0	1.3	0	1.6	15.8	16.6	17 M (1.6)	16.6	17 M (1.6)
kmeans_819k	0	1.2	0	1.3	0.1	1.4	60.8	38.9	5.1 H (1.1)	3	1.5 H (3)
kmeans_oi	0.1	1.2	0	1.3	0.1	1.4	57.6	32.8	3.8 H (1.1)	0.2	1.8 H (2.0)
lavaMD	0	1	0	1	0	1	13.2	13.2	8.0 H (1.0)	0.1	6.7 H (1.2)
lud_i	2	19.5	6.7	13.2	4	16	10.6	15.8	0.0 H (18.2)	11.6	0.0 H (18.7)
lud_256	0.4	8.5	0.5	7.8	0.6	8	11.8	15.7	0.0 H (7.6)	11.8	0.0 H (7.2)
nn	0	1	0	1	0	1	38	38	0.0 H (1.0)	38	0.0 H (1.0)
nw	3.6	88.2	7.7	92.1	2.9	87.5	0.1	1.3	0.0 H (87.1)	2.5	0.0 H (87.6)
scluster	0.9	128.9	1.9	127.5	1.2	128.5	25.9	30.4	0.0 H (125.5)	30.4	0.0 H (119.5)
srad_v1	2	98.2	0.9	99.2	0.6	99.5	2	2.3	0.1 H (101.8)	2.3	0.1 H (101.8)
Parboil Suite											
bfs	4.2	1.1	3.9	1.1	4	1.1	37.8	40.4	0.9 H (1.1)	40.4	0.9 H (1.1)
cutcp	3.3	4.1	2.9	4	3	4	17.5	19.5	0.9 H (4.0)	19.5	0.9 H (4.0)
histo	0.4	20.1	0.2	20	0.3	19.9	60.9	57.4	0.2 H (18.4)	57.4	0.2 H (18.4)
mri	0.4	3	0.2	3	0.3	3	8.2	8.2	0.2 H (2.9)	8.2	0.2 H (2.9)
sad	0	1	0	1	0	1	7.8	7.8	0.3 H (1.0)	7.8	0.3 H (1.0)
sgemm	0	1	0	1	0	1	153.9	153.9	2.9 H (1.0)	153.9	2.9 H (1.0)
spmv	2.2	48.9	0.8	50.4	0.5	50.3	14.2	12.4	0.1 H (50.9)	12.4	0.1 H (50.9)
stencil	0	100	1.3	101.3	0.3	99.7	30.1	30.1	0.0 H (1)	30.1	0.0 H (1)
Polybench Suite											
2Dcnn	0	1	0	1	0	1	12	17	1.3 H (1.0)	42	0.2 H (4.6)
2mm	0	2	0.1	2	0	2	6.8	1.7	99.7 H (2.0)	15	3.8 H (1.3)
3dconvolution	4.6	242.9	2.2	259.8	0.4	253	50.3	56.6	0.0 H (243.7)	56.6	0.0 H (249.7)
3mm	0.4	3	0.1	3	0.5	3	11.4	11.6	1.7 H (3.0)	7.9	1.3 H (4.0)
atax	0	1	0	1	0	1	22.4	22.4	2.3 H (1.0)	22.4	2.3 H (1.0)
bicg	0	1	0	1	0	1	23	23	2.2 H (1.0)	23	2.2 H (1.0)
correlation	0	1	0	1	0	1	42.8	42.8	494.4 H (1.0)	42.8	494.4 H (1.0)
covariance	0	1	0	1	0	1	43.4	43.4	502.6 H (1.0)	43.4	502.6 H (1.0)
fdtd2d	1.6	711.1	1.3	722.5	1.6	706.9	6.5	2.6	0.3 H (725.6)	2.6	0.1 H (2725.5)
gemm	0	1	0	1	0	1	12.8	12.8	1.9 H (1.0)	7.5	1.5 H (1.3)
gsummv	0	1	0	1	0	1	0.1	0.1	2.5 H (1.0)	0.1	2.5 H (1.0)
gramschmidt	4.9	498.2	6.8	507.1	4.3	494.5	27.8	26.3	1.1 H (500)	26.3	1.1 H (500)
mvt	0	1	0	1	0	1	22.9	22.9	2.3 H (1.0)	22.9	2.3 H (1.0)
syr2k	0	1	0	1	0	1	119	188	50 D (1.0)	11.0	24 H (50)
syrk	0	1	0	1	0	1	1.7	1.7	45.2 H (1.0)	17.6	8.2 H (5.5)
Cutlass Perf Suite SGEMM (10 inputs)											
Mean	0.3	6.0	0.0	6.0	0.0	6.0	1.9	1.9	4.9 H (6.1)	3.7	2.4 H (7.6)
Cutlass Perf Suite WGEMM (TensorCore) (10 inputs)											
Mean	0.3	7.0	0.7	7.0	0.1	7.0	44.9	45.0	1.8 H (7.0)	42.7	0.4 H (12.3)
Deepbench Suite - Convolution - Inference (5 inputs)											
Mean	0.8	1.5	0.9	1.5	0.6	1.6	13.4	13.5	2.3 H (1.4)	13.6	2.1 H (1.5)
Deepbench Suite - Convolution - Training (5 inputs)											
Mean	1.3	2.8	51.3	5.0	0.5	3.6	*	*	*	*	*
Deepbench Suite - Convolution - Inference (TensorCore) (5 inputs)											
Mean	0.9	1.5	0.2	1.5	0.2	1.5	11.1	11.9	2.9 H (1.4)	13.0	2.5 H (1.6)
Deepbench Suite - Convolution - Training (TensorCore) (5 inputs)											
Mean	2.1	1.9	*	*	*	*	21.6	25.8	14.8 H (1.7)	28.3	12.5 H (2.9)
Deepbench Suite - GEMM bench - Inference (5 inputs)											
Mean	2.4	1.1	4.1	1.2	4.2	1.2	10.3	12.4	2.2 H (1.2)	12.4	2.2 H (1.3)
Deepbench Suite - GEMM bench - Training (5 inputs)											
Mean	0.9	1.3	0.2	1.6	0.6	1.5	12.6	11.6	3.5 H (1.3)	11.6	3.4 H (1.4)
Deepbench Suite - GEMM bench - Inference (TensorCore) (5 inputs)											
Mean	2.4	1.1	4.0	1.2	4.0	1.2	10.4	12.5	3.1 H (1.2)	12.5	3.1 H (1.2)

Application	Principal Kernel Selection using Silicon						Simulation				
	Volta		Turing		Ampere		Volta				
	Error [%]	SU	Error [%]	SU	Error [%]	SU	Baseline SimError	PKSErr	PKS SimTime [H] (SU)	PKAError	PKA SimTime [H] (SU)
Deepbench Suite - GEMM bench - Train (TensorCore) (5 inputs)											
Mean	0.8	1.3	0.1	1.5	0.8	1.5	12.7	11.8	4.2 H (1.3)	11.8	4.1 H (1.3)
Deepbench Suite - RNN bench - Inference (9 inputs)											
Mean	3.3	3.0	5.6	5.3	3.2	4.5	18.7	13.0	6.1 H (1.9)	13.0	6.1 H (1.9)
Deepbench Suite - RNN bench - Train (5 inputs)											
Mean	0.5	1.1	1.5	1.2	1.1	1.1	19.4	18.8	6.3 H (1.2)	18.8	6.3 H (1.2)
Deepbench Suite - RNN bench - Inference (TensorCore) (10 inputs)											
Mean	3.4	3.2	6.6	5.0	3.6	4.3	18.8	13.3	5.7 H (2.1)	13.3	5.7 H (2.1)
Deepbench Suite - RNN bench - Train (TensorCore) (5 inputs)											
Mean	0.6	1.1	1.6	1.2	0.7	1.1	19.6	19.0	6.0 H (1.2)	19.0	6.0 H (1.2)
MLPerf Suite											
BERT Offline Inference	12.5	21564	*	*	*	*	*	29.51	0.4 H	29.51	0.4 H (1)
SSD Training	32.5	13000	*	*	*	*	*	35.9	4.5 H	28	0.5 M (500)
ResNet 50 64b Inference	3.2	1144	*	*	*	*	*	6.4	10 H	18	1.3 H (17)
ResNet 50 128b Inference	3.8	851	*	*	*	*	*	3.5	8 H	12	1.5 H (5)
ResNet 50 256b Inference	0.7	330	*	*	*	*	*	2.2	18 H	24	1.6 H (11)
GNMT Training	16.2	9630	*	*	*	*	*	17.0	36 H	39	25 H (1.4)
3D-Unet Inference	2.8	141	*	*	*	*	*	49.3	0.1 H	49.3	0.1 H (1)

ing with the convolution inference workloads, both the CUDA and Tensor Core variants. We see the same sensible errors and modest speedups across all GPU generations, a mean error of 0.8% and a speedup of 1.5×. In training, things get interesting due to a quirk with the cuDNN libraries. Out of the five workloads, only one workload per card had the same number of kernels as the Volta card. One of the cuDNN functions selects the best performing backward and forward propagation algorithms based on some metrics at runtime. Introducing the profiler in the mix resulted in several different combinations of algorithms being used, therefore the work being done was no longer the same across muliple runs. The Turing card had an error of 51.3% and a speedup of 5×, while the Ampere card had an error of 0.5% and a speedup of 3.6 ×. The GEMM bench and RNN bench results are similar for both cards and both CUDA and Tensor Core variants.

5.2.3 Simulation Results

Finally, in this subsection, we discuss columns 7-11 in Table 4 which evaluate PKS and PKS+PKP (i.e. PKA) in simulation using Accel-Sim and the V100 model [30]. In Table 4 we report the simulator error with respect to silicon ("SimError") such that PKS and PKA error can be put into context.

We start with the Rodinia, Parboil and Polybench suites. The average error between the baseline simulator and PKS is consistently very close, with the speedups tracking what we saw in silicon (Section 5.2.1). Applying full-PKA to these applications is a mixed bag, where many of the applications see the bulk of their speedup from PKS. There are a few exceptions to the rule, inparticular fdtd2d, syr2k, syrk, 2mm, 2Dcnn and others show large reductions in simulation time when PKP is applied. We experienced some issues with with myocyte, where the profiling and tracing runs (necessary for Accel-Sim) ran a mismatched number of kernels.

Cutlass and Deepbench show little additional accuracy loss between full-simulation and PKS. The average simulation time of these workloads is 17 hours, while enabling PKS drops the average simulation time to 3 hours. The mean error across all Deepbench workloads is 15.0% in Accel-Sim, and 14.5% with PKS enabled, cutting simulation time in half. Here, PKA is generally more effective, since the kernels tend to be longer and there are fewer of them. In the Deepbench convolution training application, we experienced the same kernel-id mismatch we did with myocyte.

For the MLPerf workloads we can identify two situations. The first one is when the workload can be completely profiled with detail using Nsight Compute. This is the case with ResNets Inference and the 3D-Unet Inference, which in conjunction present a low average *Principal Kernel Selection* silicon error of 2% and a geomean speedup of 460X. The simulations show an average error of 15%, no speedup is reported, because these four workloads have not been simulated to completion. Therefore we just present the time to simulate in hours for *Principal Kernel Selection*, while the speedup of *Principal Kernel Analysis* is presented relative to *Principal Kernel Selection*. The second situation is when we cannot profile with detail, and have to use our hierarchichal technique. These are Single Stage Detector training, GNMT training (RNN translation), and BERT inference. These workloads have millions of kernels. We profiled 1,000,000 kernels of BERT inference with the detail profiler in approximately a month. The error is higher, an average of 20%, and the speedup is considerably larger.

5.3 Case Studies on Relative Accuracy

In this subsection, we explore the particular use-case architectures care about when using simulators: If I make a change to the architecture, does the trend of the simulator match the trend of final hardware? To evaluate the effectiveness of *Principal Kernel Analysis* in this scenario, we perform two case studies where we measure the speedup of an architectural change and calculate the corresponding speedup measured by *Principal Kernel Analysis*.

In Figure 9, we evaluate the relative speedup of a

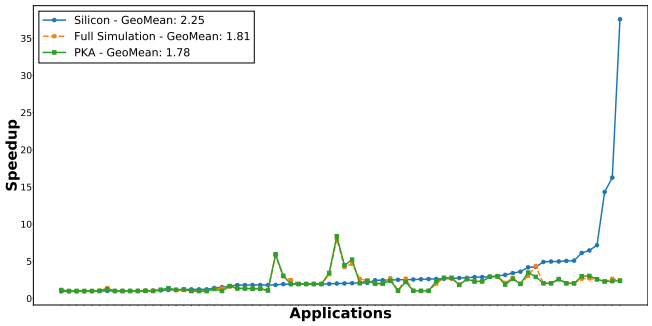


Figure 9: Volta V100 speedup over Turing RTX 2060 in silicon, when using full simulation and when using *Principal Kernel Analysis*.

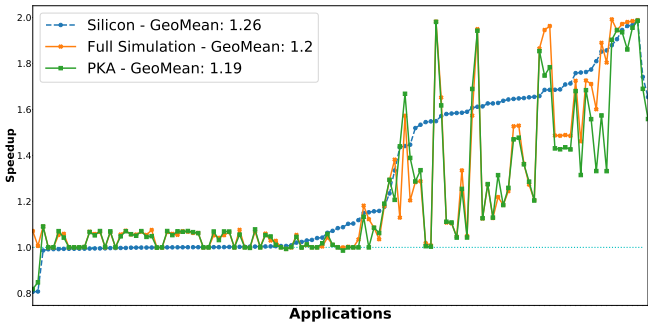


Figure 10: Speedup of using 80 cores over 40 cores in a Volta V100 evaluated in silicon, full simulation and using *Principal Kernel Analysis*.

.

high-end Volta V100 over a lower-end (but newer architecture) Turing RTX 2060. Note that we are unable to execute all the workloads on the Turing, in particular MLPerf, due to its limited memory capacity. Figure 9 demonstrates that *Principal Kernel Analysis* very closely matches the predicted speedup of full simulation (although the baseline simulator has some inaccuracy). However, the simulator's inaccuracy is independent from *Principal Kernel Analysis*'s effectiveness.

To cover all the workloads, while still evaluating a silicon-validated architectural change, we half the number of SMs on the V100 using NVIDIA's Multi-Process Service. Figure 10 plots the resulting speedup of using 100% of the SMs over using 50% for silicon, full simulation and *Principal Kernel Analysis*. Again, despite perturbations in the baseline simulator's accuracy, *Principal Kernel Analysis* tracks very closely to full simulation for the workloads that can be fully simulated. For the MLPerf workloads (on the far right), PKA's speedup error is less than 10%.

6. RELATED WORK

Prior work on CPUs showed that single-threaded applications could have varying performance characteristics over time [48] and that Basic Block Vectors can be used as an architecture-independent way for capturing these features [17, 41, 47]. In this work, we showed that massively multi-threaded accelerators have spatial patterns and small, but deliberate modifications to Basic Block Vectors can be sufficient to capture them. Harmony [29] introduced parallel block vectors for summarizing the degrees of parallelism at the granularity of basic blocks. Eeckhout et al. [17] propose PCA plus clustering based workload characterization for CPU SPEC workloads. Phansalkar et al. [41] pointed out the importance of selecting microarchitecture-independent characteristics.

Later work on CPUs showed that sampling based on basic block vectors could be used to automatically characterize large-scale behaviors of programs with a direct application for improving simulation time [49]. Furthermore, this understanding of program behavior could be implemented in hardware at a minimal cost, and be used to detect phase-based program behavior [50]. In this work, we show that spatial features can be used to a similar end for characterizing the behavior of GPU programs and improving simulation time, and opens the possibility for hardware-based phase prediction.

Furthermore, Sherwood et al. [25,49] showed that CPU phases are often stable over millions of cycles. This idea has been used by later works such as the Application Slowdown Model [52] for sampling performance metrics over time to represent large-scale behaviors. Our work provides similar insights, but over space.

Tallam et al. [54] showed that traces of control flow could be used to capture the data dependencies in applications faster, for use in code optimization and dynamic program slicing techniques. Recent work on accelerating GPU simulation showed that parallelization of the simulator itself could improve simulation time between 2.5x and 3.5x compared to serial simulation for multiple GPUs [53]. Parallelization has also been used CPUs [46].

Pai et al. [39] predict kernel execution to improve performance and fairness at runtime for preemptive multi-kernel execution. They show that kernel execution time can be obtained by sampling, possibly as soon as a single thread block finishes execution (i.e., waiting the first wave of thread blocks to be executed).

7. CONCLUSION

This paper presents *Principal Kernel Analysis*, a silicon-verified mechanism to concisely represent GPU applications such that simulation times for scaled, real-world workloads are tractable. *Principal Kernel Analysis* is a hierarchical mechanism that reduces both the number of kernels executed through *Principal Kernel Selection*, and decreases the number of threadblocks executed in each kernel with *Principal Kernel Projection*. We demonstrate the effectiveness and generality of *Principal Kernel Selection* over 147 workloads across the three most recent NVIDIA GPU generations, resulting in average speedup and error rates that range from an average of 7.2× @12.6% error across Rodinia to 1987× @28.5% error across 7 MLPerf applications. We then perform a case study applying *Principal Kernel Analysis* to Accel-Sim, demonstrating that the centuries-long simulation times for MLPerf can be reduced to a matter of hours with error rates that are in-line the baseline simulator.

REFERENCES

[1] A. Ariel, W. W. Fung, A. E. Turner, and T. M. Aamodt, "Visualizing complex dynamics in many-core accelerator architectures," in *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 2010, pp. 164–174.

[2] 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," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[3] Baidu, "Deepbench: Benchmarking deep learning operations on different hardware," 2017. [Online]. Available: https://github.com/baidu-research/DeepBench

[4] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. Kirby, J. Freymann, K. Farahani, and C. Davatzikos, "Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features," *Scientific data*, vol. 4, Sep. 2017.

[5] S. Bakas, M. Reyes, A. Jakab, S. Bauer, M. Rempfler, A. Crimi, R. T. Shinohara, C. Berger, S. M. Ha, M. Rozycki, M. Prastawa, E. Alberts, J. Lipková, J. B. Freymann, J. S. Kirby, M. Bilello, H. M. Fathallah-Shaykh, R. Wiest, J. Kirschke, B. Wiestler, R. R. Colen, A. Kotrotsou, P. LaMontagne, D. S. Marcus, M. Milchenko, A. Nazeri, M. Weber, A. Mahajan, U. Baid, D. Kwon, M. Agarwal, M. Alam, A. Albiol, A. Albiol, A. Varghese, T. A. Tuan, T. Arbel, A. Avery, P. B., S. Banerjee, T. Batchelder, K. N. Batmanghelich, E. Battistella, M. Bendszus, E. Benson, J. Bernal, G. Biros, M. Cabezas, S. Chandra, Y. Chang, and et al., "Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge," *CoRR*, vol. abs/1811.02629, 2018. [Online]. Available: http://arxiv.org/abs/1811.02629

[6] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 163–174.

[7] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.

[8] B. Beckmann and A. Gutierrez, "The AMD gem5 APU Simulator: Modeling Heterogeneous Systems in gem5," in *Tutorial at the International Symposium on Microarchitecture (MICRO)*, 2015.

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

[10] J. Burgess, "Rtx onâĂŤthe nvidia turing gpu," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.

[11] T. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout, "Barrierpoint: sampled simulation of multi-threaded applications," in *IEEE International Symposium on Performance Analysis of Systems and Software-ISPASS*. IEEE, 2014, pp. 2–12. [Online]. Available: http://dx.doi.org/10.1109/ISPASS.2014.6844456

[12] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[13] J. Choquette, E. Lee, R. Krashinsky, V. Balan, and B. Khailany, "The a100 datacenter gpu and ampere architecture," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, 2021.

[14] J. Choquette, "Volta: Programmability and performance," in *Hot Chips: A Symposium on High Performance Chips*, 2017.

[15] J. Devlin and M. Chang, "M, k. lee and k. toutanova,âĂIJbert: Pretraining of deep bidirectional transformers for language understanding,âĂİ, 2018," *arXiv preprint arXiv:1810.04805*.

[16] G. Diamos, A. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2010, pp. 353–364.

[17] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload design: Selecting representative program-input pairs," in *Proceedings. International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 2002, pp. 83–94.

[18] I. V. Evstigneev, T. Hens, and K. R. Schenk-Hoppé, "Evolutionary stable stock markets," *Economic Theory*, vol. 27, no. 2, pp. 449–468, 2006.

[19] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring gpgpu workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *IEEE International Symposium on Workload Characterization (IISWC'10)*. IEEE, 2010, pp. 1–10.

[20] T. Grass, A. Rico, M. Casas, M. Moreto, and E. AyguadÃĺ, "Taskpoint: Sampled simulation of task-based programs," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 296–306.

[21] 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*, 2012.

[22] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program analysis," in *Journal of Instruction Level Parallelism*, 2005.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

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

[25] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 93–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956567

[26] F. Isensee, J. Petersen, S. A. A. Kohl, P. F. Jäger, and K. Maier-Hein, "nnu-net: Breaking the spell on successful medical image segmentation," *ArXiv*, vol. abs/1904.08128, 2019.

[27] A. Jaleel, E. Ebrahimi, and S. Duncan, "Ducati: High-performance address translation by extending tlb reach of gpu-accelerated systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 1, pp. 1–24, 2019.

[28] M. Kambadur, S. Hong, J. Cabral, H. Patil, C.-K. Luk, S. Sajid, and M. A. Kim, "Fast computational gpu design with gt-pin," in *2015 IEEE International Symposium on Workload Characterization*. IEEE, 2015, pp. 76–86.

[29] M. Kambadur, K. Tang, and M. A. Kim, "Harmony: Collection and analysis of parallel block vectors," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 452–463.

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

[31] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and

P. DollÃąr, "Microsoft coco: Common objects in context," 2014, cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list. [Online]. Available: http://arxiv.org/abs/1405.0312

[32] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2016, to appear. [Online]. Available: https://doi.org/10.1007/978-3-319-46448-0_2

[33] 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 *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. New York, NY, USA: Association for Computing Machinery, 2005, p. 190âĂŞ200.

[34] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf *et al.*, "Mlperf training benchmark," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 336–349, 2020.

[35] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ã. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H. C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput, "The multimodal brain tumor image segmentation benchmark (brats)," *IEEE Transactions on Medical Imaging*, vol. 34, no. 10, pp. 1993–2024, 2015.

[36] NVIDIA, "CUTLASS: CUDA template library for dense linear algebra at all levels and scales," https://github.com/NVIDIA/cutlass, 2018.

[37] NVIDIA, "NVIDIA Nsight CLI," https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html, 2019.

[38] M. OâĂŹConnor, 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 *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 41–54.

[39] S. Pai, R. Govindarajan, and M. J. Thazhuthaveetil, "Preemptive thread block scheduling with online structural runtime prediction for concurrent gpgpu kernels," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014, pp. 483–484.

[40] H. Patil, R. Cohn, M. Charney, R. Kapoor, and A. Sun, "Pinpointing representative portions of large intel itanium programs with dynamic instrumentation," in *In International Symposium on Microarchitecture*. IEEE Computer Society, 2004, pp. 81–92.

[41] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring program similarity: Experiments with spec cpu benchmark suites," in *IEEE International Symposium on Performance Analysis of Systems and Software, 2005. ISPASS 2005.* IEEE, 2005, pp. 10–20.

[42] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *EMNLP*, 2016.

[43] 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 *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[44] T. G. Rogers, D. R. Johnson, M. O'Connor, and S. W. Keckler, "A variable warp size architecture," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 489–501, 2015.

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, p. 211âĂŞ252, Dec. 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[46] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 475–486, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485963

[47] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2001, pp. 3–14.

[48] T. Sherwood and B. Calder, "Time varying behavior of programs," 1999.

[49] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[50] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," *SIGARCH Comput. Archit. News*, vol. 31, no. 2, pp. 336–349, May 2003. [Online]. Available: http://doi.acm.org/10.1145/871656.859657

[51] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, 2012.

[52] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 62–75.

[53] 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 *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: ACM, 2019, pp. 197–209. [Online]. Available: http://doi.acm.org/10.1145/3307650.3322230

[54] S. Tallam and R. Gupta, "Unified control flow and data dependence traces," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 3, Sep. 2007. [Online]. Available: http://doi.acm.org/10.1145/1275937.1275943

[55] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 335–344.

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

[57] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: ACM, 2019, pp. 372–383. [Online]. Available: http://doi.acm.org/10.1145/3352460.3358307

[58] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "Turbosmarts: Accurate microarchitecture simulation sampling in minutes," SIGMETRICS Performance

Evaluation Review, Tech. Rep., 2005.

[59] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[60] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *in Proceedings of the 30th annual international symposium on Computer architecture*, 2003, pp. 84–97.

[61] Z. Yu, L. Eeckhout, N. Goswami, T. Li, L. John, H. Jin, and C. Xu, "Accelerating gpgpu architecture simulation," in *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, 2013, pp. 331–332.

[62] Z. Yu, L. Eeckhout, N. Goswami, T. Li, L. K. John, H. Jin, C.-Z. Xu, and J. Wu, "Gpgpu-minibench: accelerating gpgpu micro-architecture simulation," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 64, no. 11, pp. 3153–3166, 2015. [Online]. Available: http://dx.doi.org/10.1109/TC.2015.2395427