Swift and Trustworthy Large-Scale GPU Simulation with Fine-Grained Error Modeling and Sampling

***Abstract*—Kernel-level sampling is one of the most effective techniques to run large-scale machine learning GPU workloads on cycle-level simulators by reducing the workload size. Prior works typically use k-means or rule-based clustering to sample kernels, assuming all kernels within a cluster behave similarly. However, we observe that ignoring the heterogeneous runtime characteristics within clusters can lead to significant sampling errors. To address this issue, we introduce STEM, a statistical error modeling approach for fine-grained kernel-level sampling. We first design an error model to determine the optimal sample sizes for each of the kernel clusters. Based on this model, we propose a novel hierarchical clustering methodology, ROOT, using mathematical optimizations to ensure that each kernel cluster exhibits homogeneous runtime behavior. To the best of our knowledge, STEM is the first work to analyze and provide tight bounds on sampling errors in GPU kernel sampling. When evaluated on the latest GPU benchmark suites, STEM demonstrates a sampling error of 0.36%, which is 81.9× and 67.4× smaller than PKA and Sieve, respectively.**

I. Introduction

Cycle-level simulations play a crucial role in computer architecture research, serving as a means to evaluate the performance impact of microarchitectural changes, conduct design space exploration, estimate power and energy consumption, and more. These simulations are extensively used in GPU research, with popular tools such as MacSim [16], AccelSim [15], and mGPUSim [29] widely adopted. Due to the intricate microarchitectural design of GPUs, which includes high degrees of thread-level parallelism (TLP) and a stratified cache and memory hierarchy, cycle-level simulations are particularly beneficial for understanding the performance behaviors of diverse GPU workloads.

GPUs have become one of the most important accelerators for machine learning (ML) workloads [6]. However, due to the large memory footprint and significant computational overhead associated with the latest ML models [22], it is nearly impossible to run GPU simulations with workloads that utilize these models. Without effective optimization techniques, the gap between workloads used on real GPUs and those employed in cycle-level simulations may significantly widen.

Workload sampling is the most popular way to accelerate cycle-level simulations by reducing the workload size while maintaining the unique behaviors of the workloads. Works like SimPoint [10], SMARTS [35], and LoopPoint [26] have been proposed and widely adapted in various cycle-level CPU simulators like Gem5 [20]. The basic idea behind these works is to first divide the workload into multiple intervals and extract signature information from each interval that captures its architectural runtime aspects. Based on these signatures, they then sample a subset of intervals to run simulations using only the selected intervals.

Due to the heterogeneous and parallel nature of GPUs, slightly different approaches have been taken in the GPU domain. Works such as TBPoint [13], PKA [1], and Sieve [21] propose kernel-level workload sampling. By simulating only the sampled kernels that represent the overall behavior of the entire workload, prior works have notably reduced simulation time. Some of these methods incorporate skipping or early termination within kernels to achieve additional speedup. Photon [19] takes a different approach; it performs online analysis during the simulation runtime and skips to the next phase when highly repetitive software behavior is identified.

However, previous works face several limitations. Firstly, prior approaches have a big discrepancy between the sampled simulation and the full simulation. We evaluated current kernel sampling solutions on the CASIO DL Application Suite [5], obtaining average sampling errors of 29.26% for PKA and 24.08% for Sieve. This level of error is critical, rendering kernel-level sampling techniques in cycle-level simulation impractical. Since the error magnitude is comparable to performance gains typically seen in architectural studies, it becomes impossible to determine whether the simulation results are influenced by sampling errors or actual microarchitectural changes.

We identify that the huge errors in prior works are due to their sampling methods that ignore whether they are sampling kernels that are representative of the whole workload. Our observations show that most kernels in GPU workloads exhibit heterogeneous execution time distributions, so different that even the same kernels sometimes have drastically different runtime durations. Prior works overlook this heterogeneity in kernels, treating each kernel's distribution as a black box and sampling too few kernels (often just one), leading to large errors. To challenge the diversity in kernel runtime behaviors, a fine-grained approach is needed that can adapt to each kernel's distribution to sample a representative number of kernels and extrapolate the whole simulation from just the sampled ones.

Secondly, previous works do not quantitatively address how the sampling error impacts the simulation results and overall errors. Prior works like TBPoint, PKA, and Sieve mainly focus on clustering methods to achieve efficient kernel sampling. However, they overlook accuracy during the sampling phase and fail to provide error bounds that quantify the error introduced in performance estimation. Without these error bounds, the sampling error becomes unpredictable, compromising the reliability and integrity of the sampled simulation results.

To overcome these limitations, we first introduce STEM, a statistical error model for GPU kernel sampling that determines optimal sample sizes using the heterogeneous runtime behavior of kernel data as input. We then use this model to propose a novel hierarchical clustering and sampling methodology, ROOT. By performing fine-grained hierarchical clustering on kernel calls, ROOT significantly reduces the number of kernel samples required for simulation, ensuring that the error remains within a specified bound. As a result, STEM and ROOT provides strict and reliable error bound on the discrepancy between the sampled simulation and the full simulation. This error bound works along with a confidence level that ensures that the sampled simulation will give an accurate approximation to the full simulation. Evaluation results on CASIO DL Application Suite demonstrate a significant reduction in error on both ML and non-ML GPU workloads compared to prior methods, while maintaining comparable degree of speedups.

We summarize the contributions of this paper as follows:

- STEM is our statistical error model on GPU workload sampling that leverages the high number of kernel calls in modern GPU workloads. STEM intelligently determines the optimal sample sizes for arbitrary set of kernels.
- ROOT is a novel hierarchical clustering methodology for GPU kernel sampling. ROOT leverages STEM to perform accurate clustering and sampling on kernels that have heterogeneous nature of kernel execution statistics. To our best knowledge, STEM and ROOT is the first work in GPU simulation domain to provide a tight theoretical error bound on the difference between full simulation and sampled simulation.
- STEM and ROOT achieves significantly smaller error compared to prior works, while maintaining comparable speedup on most GPU benckmark suites. STEM and ROOT achieves an error of 0.36% on the latest GPU benchmark suites, which is 81.9× and 67.4× smaller than PKA and Sieve, respectively.
- STEM and ROOT's lightweight profiling overhead enables our method to achieve kernel sampling on large ML workloads, such as the Bloom and GPT-2 models, which prior works cannot handle due to their significant hardware profiling overhead.

II. Background

A. Kernel sampling in GPU workloads

Kernel-level workload sampling is one of the most popular approach for reducing simulation time in the GPU domain [1], [13], [21]. It is a form of representative workload sampling [8], based on the assumption that the set of sampled kernels can fully represent the runtime behavior of the entire workload. Another key assumption underlying this method is that GPUs exhibit a linear sequence of kernel calls. This enables estimating runtime simulation statistics, such as the total number of simulation cycles, for the full workload by simulating only a subset of kernels.

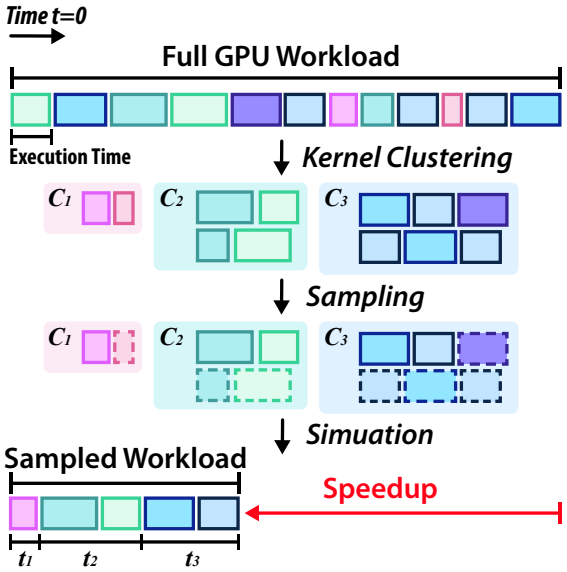

Fig. 1. Kernel-level sampling methodology in GPU simulations. Kernels with similar runtime characteristics are grouped into clusters, and a few number of kernels are sampled from each cluster. The sampled simulation achieves speedup by only running the sampled kernels. The goal of kernel-level sampling is to reduce the sample set size while keeping the sampling error small.

Although prior works perform sampling differently, their general approach is similar, as depicted in Figure 1. First, the kernel calls of the GPU workload are partitioned into several clusters using various clustering methods, ensuring that kernels within each cluster exhibit similar runtime behaviors. Sampling is then performed on each cluster, with the assumption that sampled kernels will represent the other kernels in the same cluster. Since only the sampled kernels are simulated, this method significantly reduces simulation time. Given that most GPU programs consist of multiple kernel calls, kernel-level workload sampling has proven effective for the majority of GPU workloads in prior studies.

With kernel sampling, the total execution time can be estimated by using a weighted sum. Let C_i denote i-th kernel cluster obtained after the clustering, shown as Figure 1. Also, if we define **sample size** m_i as the **number of kernels that we sample from a cluster**, the estimated total execution time t_{total} can be obtained as follows:

$$t_{total} = \sum_i \frac{\text{num of kernels in } C_i}{\text{sample size for } C_i} \cdot t_i = \sum_i \frac{|C_i|}{m_i} \cdot t_i, \quad (1)$$

where t_i is the duration sum of kernels that are sampled from cluster C_i.

Moreover, let t^* represent the ground-truth total execution time that we want t_{total} to approximate. Since every t_i can be measured during the sampled simulation, we can obtain t_{total} which is a good approximation of t^* only from the sampled simulation. We define the **sampling error** e between the estimated and true total execution times as follows:

$$e \equiv \left| \frac{t^* - t_{total}}{t^*} \right| \times 100(\%). \quad (2)$$

The goal of kernel-level sampling is to reduce the sample size to reduce the simulation time while keeping the sampling error small.

B. Prior works on kernel sampling

TBPoint uses architecture-independent metrics obtained from profiling to apply hierarchical clustering, grouping similar kernels together and then sampling the kernel closest to the center of each group. **PKA** extends the idea of TBPoint by performing PCA for dimensionality reduction on feature vectors from profiled data and then uses k-means clustering, sweeping through k=1 to 20 to find the optimal k. PKA then samples the first-chronological kernel from each clusters. **Sieve**, on the other hand, only uses the number of instructions as the feature vector to reduce profiling overhead. It clusters the kernels into three groups based on the Coefficient of Variation (CoV) of instruction counts, defined as the standard deviation divided by the mean. It then samples the first-chronological kernel with the most dominant CTA size.

The limitation of these works is that both the clustering and sampling procedures are based on empirical insights, lacking a solid theoretical foundation. First, there is no analysis of the relationship between the chosen clustering methodology–typically using hardware metrics–and the resulting kernel clusters. Consequently, it is unclear whether the clustering effectively groups similar kernels into the same clusters and minimizes sampling error, as there is no formal evaluation of its impact. Additionally, the sampling procedure itself lacks a rigorous analysis on the distribution of kernels within clusters. This turns the sampling process into a "black box," as it remains uncertain whether the sampled kernels truly represent the overall behavior of the kernels within the same cluster. Additionally, there is no qualitative discussion on sampling error or corresponding error bounds, which undermines the trustworthiness of kernel sampling methods.

To address these challenges, our approach leverages kernel execution time statistics as the core metric for effective clustering and accurate sampling. By applying statistical techniques, we optimize both the kernel clustering and sampling process while explicitly quantifying the sampling error. Our proposed fine-grained clustering method, integrated with statistical error modeling, enhances the robustness and theoretical foundation of the sampling procedure. The importance of statistical error modeling is discussed in detail in Section III, and the proposed methods are outlined in Section IV.

C. Other sampling methods in GPU workloads

Other popular GPU workload sampling methods, aside from kernel sampling, focus on sampling or skipping simulation within a single kernel. Both TBPoint and PKA incorporate intra-kernel sampling to achieve additional speedup at the cost of introducing a small level of error. They monitor whether the runtime behavior of kernels stabilizes, and if so, they skip the remaining simulation for that kernel or proceed to the next one. Photon [19] employs online analysis to dynamically check during runtime if the basic block (BB), warp, or kernel has stabilized, then skips the simulation to the next phase.

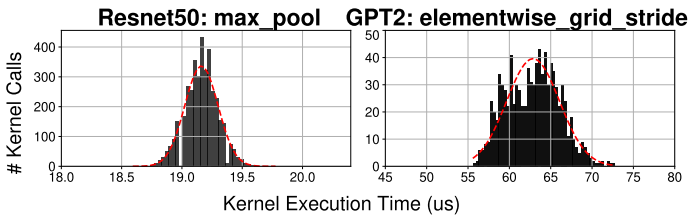

Fig. 2. Kernel execution time histograms of GPU kernels in ResNet50 and GPT-2 workload. The workload and kernel's name are shown on the top. The red dotted lines show the ideal normal distribution. Wide kernel distributions suggest that an accurate error modeling like STEM is needed on kernel-level sampling. Moreover, the distributions motivate STEM to apply statistical methods for error modeling.

In this paper, we focus exclusively on kernel-level workload sampling, as modern workloads exhibit a massive number of kernel calls, with each kernel being relatively short. This makes it feasible to simulate a subset of kernels at the cycle-level within a feasible amount of time. Additionally, since kernel sampling is orthogonal to methods like warp- or BB-level sampling proposed in prior work, these techniques can be combined with other inter-kernel level sampling methods; which is expected to yield even greater speedup.

D. GPU hardware profilers

GPU vendors provide various hardware profiling tools that are valuable for architectural research. NVIDIA offers Nsight Systems (NSYS) [24], a performance analysis tool designed for scaling optimizations, and Nsight Compute (NCU) [23], which provides detailed hardware-level runtime metrics such as the number of instructions, memory accesses, warp occupancy, and more. These insights help CUDA programmers optimize their code. NVIDIA also provides the NVidia Binary Instrumentation Tool (NVBit) [33], a library for NVIDIA GPUs that enables instruction injection into precompiled CUDA programs, allowing researchers to gather hardware-level information or modify program behavior by altering instructions, register values, and other elements. Similarly, AMD provides the Radeon GPU Profiler (RGP) for its GPUs, offering functionality comparable to NCU for NVIDIA GPUs.

GPU hardware profilers play a crucial role in analyzing kernels, and the profiled data can be leveraged for efficient kernel sampling. STEM utilizes NSYS to capture the execution time of every kernel call. Other approaches, such as TBPoint, use GPUOcelot [7], PKA relies on NCU to gather 12 metrics for kernel clustering, and Sieve employs NVBit to obtain the number of instructions at runtime.

III. Motivations on designing STEM

We propose two key points in designing our kernel sampling methodology that address the limitations of prior works: determining the optimal sample size to balance the trade-off between speedup and error, and employing fine-grained kernel clustering to achieve accurate representative sampling in GPU workloads.

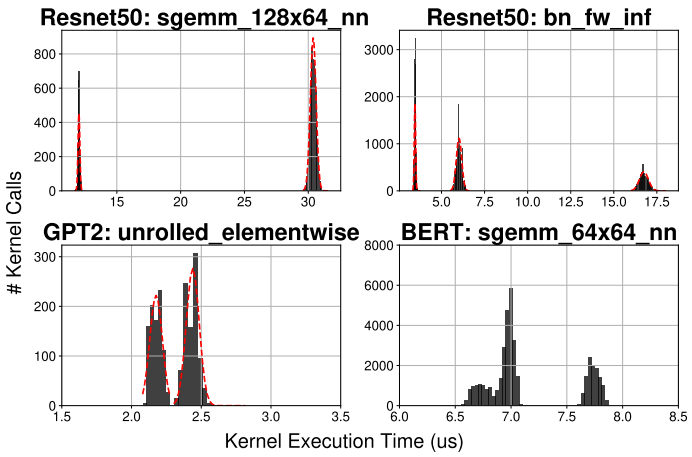

Fig. 3. Kernel execution time histograms illustrating distinct runtime behaviors of the same kernels across ResNet50, GPT-2, and BERT workloads. The multiple peaks in each histogram suggest the need for a more refined clustering method to accurately capture and sample from these varying behaviors, rather than treating all kernels in a cluster uniformly.

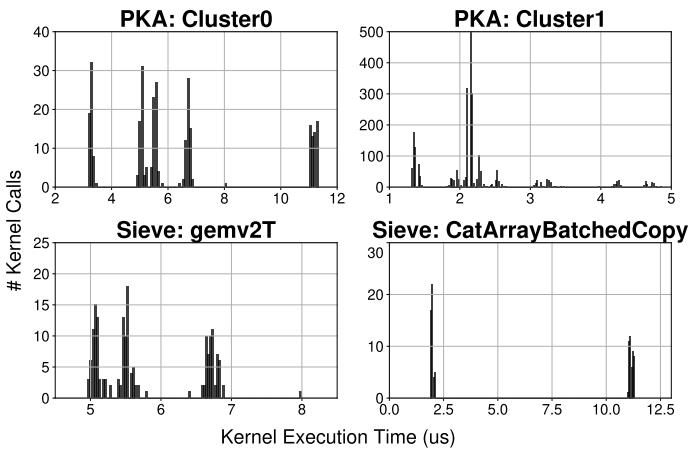

Fig. 4. Execution time distributions of kernel clusters in PKA and Sieve methods. Each subplot depicts the distribution of a single cluster, where cluster name and method are shown at the top. Using the DLRM workload, we observe that even a single cluster in both PKA and Sieve exhibits highly diverse execution behaviors. Since both PKA and Sieve methods sample only one kernel per cluster, this can lead to significant sampling error.

A. Kernel sampling with statistical error modeling

Figure 2 shows two execution time histograms of two kernels, the name of each kernel and workload are shown at the top. The diagram illustrates that even identical kernels can exhibit varying behaviors at runtime, as indicated by the wide distributions. This suggests that kernel sampling should wisely determine the sample size, primarily considering statistical measures such as the mean and variance of the distribution, ensuring that the sampled mean can closely approximate the population mean. Previous methods like PKA and Sieve often introduce large sampling errors by sampling only one kernel per cluster, despite the fact that these approaches can achieve significant speedup. This may harm the accuracy and reliability of the sampled simulation, rendering the estimation meaningless. This issue is particularly pronounced in ML workloads with massive number of kernel calls in clusters.

To mitigate this issue, we employ STEM during the sampling phase. Instead of sampling only the first-chronological kernel from each cluster, our proposed STEM method determines an optimal sample size based on the distribution within the cluster. The details of this approach are discussed in the next section.

B. Fine-grained kernel clustering

Figure 3 illustrates how the same kernel can exhibit distinct runtime behaviors, with multiple peaks in execution time distributions. These peaks should be separated prior to sampling for accurate analysis. However, not all kernels display clearly separated peaks; for example, BERT's `sgemm` kernel shows overlapping peaks, making clustering difficult. This highlights the importance of fine-grained clustering methods that can effectively separate such execution time peaks.

Figure 4 presents the execution time distributions of kernels within clusters produced by the PKA and Sieve methods. Surprisingly, each cluster contains highly diverse kernel behaviors, which suggests the clustering phase in both PKA and Sieve is ineffective. Since PKA and Sieve only sample a single kernel from each cluster, this can lead to significant error in sampling, particularly when execution times within a cluster are heterogeneous.

TABLE I
TOP 5 TIME-CONSUMING GPU KERNELS IN BERT WORKLOAD. THE WORKLOAD INVOLVES A LARGE NUMBER OF KERNEL CALLS, SUFFICIENT ENOUGH TO APPLY STATISTICAL APPROACHES.

Kernel Name	# Calls	Total Time (ns)
volta_sgemm_64x32_sliced1x4_tn	278400	2910365652
volta_sgemm_32x32_sliced1x4_tn	69600	1878500991
splitKreduce_kernel	348000	803385686
volta_sgemm_64x64_tn	69600	715703692
gemmSN_TN_kernel	41600	482964944

ROOT is our proposed kernel sampling methodology that performs a fine-grained kernel clustering by leveraging the STEM. ROOT employs a recursive greedy approach along with kernel execution data to perform efficient kernel clustering while ensuring that the error of the sampled simulation remains within a small bound ϵ. We describe ROOT and STEM in detail in the following section.

C. Applying the Central Limit Theorem

To design an accurate error modeling for kernel clustering and sampling, we first employ the Central Limit Theorem (CLT) to build a confidence bound on the performance estimate [18]. CLT ensures that the sample mean converges to a normal distribution if each sample is independent and identically distributed (i.i.d.). The i.i.d. condition required by the CLT is easily satisfied when random sampling with replacement is used. Independence is ensured since each sample is drawn independently, without influencing the others. The identically distributed condition is met since every sample follows the same underlying probability distribution.

Since the standard error of a normal distribution is well-known, we can leverage the CLT to theoretically model the sampling errors in GPU kernels. We also leverage the high number of repetitive kernel calls in modern GPU workloads in applying CLT, as the sample mean converges to a normal

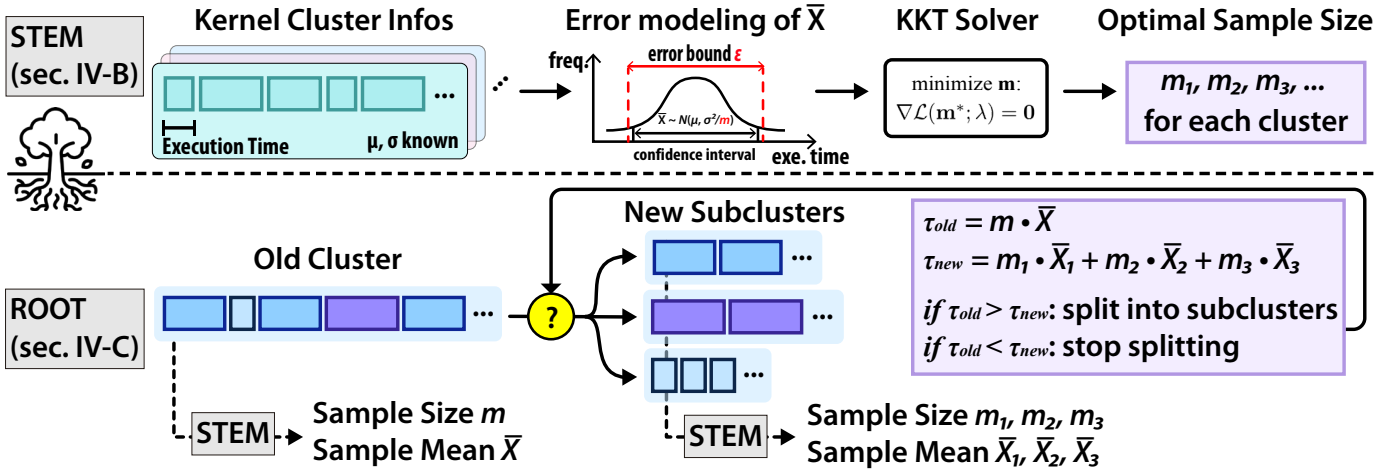

Fig. 5. Overview of STEM (top) and ROOT (bottom). STEM first takes kernel cluster and execution time of each kernel as input. An error model then estimates the sampling error of sample mean $\bar{X}$. A KKT solver is then used to minimize the sample size for each cluster while ensuring the sampling error is bounded. ROOT determines whether the original cluster or newly split subclusters provide greater simulation speedup by leveraging STEM. If $\tau_{old} > \tau_{new}$, the new simulation time with subclusters is shorter, ROOT proceeds to split the original cluster. Otherwise, the recursion is halted, preserving the original cluster.

distribution when the sample size goes to infinity. As shown in Table I, we observe a massive amount of repeated kernel calls through most of the GPU workloads.

Consider a case where we sample m kernels from a cluster C. Our primary goal is to estimate the total execution time of C by only knowing the execution time of the sampled kernels. By using CLT, no matter what the distribution of kernels in C looks like, the sample mean $\bar{X}$ always follow a normal distribution if the sample size is bigger than 30 [30]. Therefore, with 95% confidence, the error bound of the sample mean is given as

$$[\mu - 1.96\frac{\sigma}{\sqrt{m}}, \mu + 1.96\frac{\sigma}{\sqrt{m}}],$$

where μ and σ are the mean and standard deviation of every kernel execution time in C [8]. We can leverage this result to design a rigorous error model; more details can be found in the following sections.

IV. STEM and ROOT Methodology

STEM is a statistical error model that leverages the CLT to determine the optimal sample sizes for the given kernel clusters. The summary of STEM is shown on the upper part of Figure 5, where we use the CLT and KKT-solver to obtain the optimal solution on arbitrary set of kernel clusters.

ROOT is our fine-grained hierarchical GPU kernel sampling methodology that leverages STEM in both clustering and sampling phases. The lower part of Figure 5 visualizes the branching condition of ROOT, using STEM during the decision making process. Figure 6 is an example of using ROOT to perform fine-grained kernel clustering. The following subsections explain STEM and ROOT in details.

A. STEM: Statistical Error Modeling for GPU simulation

We first consider the case where a single kernel cluster C is given, and we want to determine the minimum sample size m to ensure the sampling error of sample mean $\bar{X}$ falls inside the given error bound ϵ. We use random sampling with replacement to ensure that i.i.d. conditions are satisfied. From the CLT, we obtain a sample mean $\bar{X}$ with follows a normal distribution, i.e., $\bar{X} \sim \mathcal{N}(\mu, \sigma^2/m)$ [17] where μ and σ^2 is the mean and variance of kernel execution times in C.

To obtain the sampling error of $\bar{X}$, we use (1) to get $t^* = |C| \cdot \mu$ and $t_{total} = |C| \cdot \bar{X}$. The sampling error definition (2) gives the following equation when the given confidence level is $1 - \alpha$:

$$e = \left|\frac{|C|\bar{X} - |C|\mu}{|C|\mu}\right| = \left|\frac{\mu \pm \frac{z_{1-\alpha/2}\sigma}{\sqrt{m}} - \mu}{\mu}\right| = \frac{z_{1-\alpha/2}\sigma}{\mu\sqrt{m}} \leq \epsilon.$$

Therefore, the sample size m ensuring $e \leq \epsilon$ can be obtained as follows:

$$m = \min\left\{\left\lceil\left(\frac{z_{1-\alpha/2}}{\epsilon}\frac{\sigma}{\mu}\right)^2\right\rceil, 30\right\}, \tag{3}$$

where ceiling function is added to ensure m is an integer, and the $\min\{\cdot, 30\}$ function ensures the sample size to be always bigger than 30 to satisfy the CLT condition. We name this Equation (3) as STEM.

The beauty of STEM is that it can be used on any clusters with arbitrary kernel distributions. However, if we assume kernels in C follows normal distribution as kernel histograms show in Figure 2, STEM can ignore the $\min\{\cdot, 30\}$ term. We call this the **Gaussian assumption**, and we can use this assumption on cases where we need bigger degree of speedup. We will talk about the effect of this assumption in the Evaluation section by analyzing the impact on error when the $\min\{\cdot, 30\}$ term is omitted.

B. STEM for multiple kernel clusters

We next consider the case where a set of clusters is given and we want to determine the optimal number of samples for each cluster that ensure the error bound $e \leq \epsilon$.

Let a set of kernel clusters as $\{C_0, C_1, ..., C_{k-1}\}$ and denote $N_i = |C_i|$ for convenience. Assume we sample

$m_0, m_1, ..., m_{k-1}$ number of kernels from each of the cluster respectively. Then, for any i in the range, C_i's estimated execution time, t_i, can be obtained as $t_i = N_i \bar{X}_i$ where $\bar{X}_i \sim \mathcal{N}(\mu_i, \sigma_i^2/m_i)$. Using the linear combination rule of normal random variables [28], the estimated execution time of every sub-cluster t is given as

$$t = \sum_{i=0}^{k-1} t_i = \sum_i N_i \bar{X}_i \sim \mathcal{N}(\sum_i N_i \mu_i, \sum_i N_i^2 \frac{\sigma_i^2}{m_i}) = \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2),$$

where $\tilde{\mu}$ and $\tilde{\sigma}$ are shown for brevity.

This t should also satisfy the error bound under $1-\alpha$ confidence interval, thus the following inequality

$$\left| \frac{(\tilde{\mu} + z_{1-\alpha/2}\tilde{\sigma}) - \tilde{\mu}}{\tilde{\mu}} \right| \leq \epsilon,$$

which becomes

$$\sum_i N_i^2 \frac{\sigma_i^2}{m_i} \leq \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i \mu_i \right)^2,$$

should also hold.

The workload size to simulate is proportional to the sum of sampled kernel's execution times. Therefore, we define τ which is the **execution time sum of sampled kernels**. Since the goal of STEM is to minimize the sample size to reduce the simulation time, we find the minimum τ within the error bound ϵ by solving a non-linear minimization problem as follows.

Problem 1.

$$\begin{aligned} \underset{m_i}{\text{minimize}} \quad & \tau = \sum_i m_i \mu_i \\ \text{subject to} \quad & \sum_i N_i^2 \frac{\sigma_i^2}{m_i} \leq \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i \mu_i \right)^2 \\ \text{and} \quad & m_i > 0 \quad \text{for} \quad \forall i \in \{0, ..., k-1\}. \end{aligned}$$

Solution. **KKT Solver.** Let $a_i \equiv \mu_i$, $b_i \equiv N_i^2 \sigma_i^2$, and $c \equiv (\epsilon \sum_i N_i \mu_i / z_{1-\alpha/2})^2$ for brevity. We apply the Karush–Kuhn–Tucker (KKT) conditions to obtain the following solution:

$$m_i = \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \quad \text{for } \forall i \in \{0, ..., k-1\},$$

where the ceiling function ensures integer m_i values, with minor sub-optimality. Details found in Appendix VIII-A. □

Using the KKT Solver, we obtain the optimal sample sizes when a set of clusters is given. Next step is to design a hierarchical clustering method with STEM.

C. ROOT: Fine-grained hierarchical GPU kernel clustering

Figure 6 illustrates ROOT's recursive methodology. First, kernels are clustered by their names, which can be obtained through hardware-level profilers like NSYS. However, as demonstrated in Figures 3 and 4, kernels with the same name often exhibit distinct execution peaks in their histograms. Because the number of peaks within each cluster is unknown, traditional methods like k-means clustering, which require a fixed number of clusters, are not applicable. Instead, a fine-grained approach that leverages STEM is necessary to intelligently divide these execution peaks into subclusters.

ROOT is a hierarchical method designed to tackle this problem. Its primary goal is to minimize simulation time using a recursive greedy approach, while ensuring that the error remains within the specified error bound. ROOT determines whether to split a cluster into smaller subclusters by comparing the estimated simulation runtime and error for each potential split. Assume we have a kernel cluster C, and applying a clustering method on C produces subclusters $\{C_i : i = 0, \ldots, k-1\}$. The clustering method can be any approach, including k-means. ROOT utilizes STEM to determine whether the sampled simulation time will be shortened by sampling from subclusters rather than treating C as a single cluster. By comparing the **simulation time of the original cluster** (τ_{old}) with the **total simulation time of the subclusters** (τ_{new}), ROOT decides whether to split the cluster or not.

$$\tau_{old} = m\bar{X} = \lceil (z_{1-\alpha/2}\sigma/\mu\epsilon)^2 \rceil \cdot \bar{X}$$
$$\tau_{new} = \sum_i m_i \bar{X}_i$$

If $\tau_{old} > \tau_{new}$, partitioning the kernel cluster C into multiple subclusters $\{C_0, \ldots, C_{k-1}\}$ will reduce the overall simulation time. In this paper, we use $k = 2$, but various values of k can be swept to find the optimal one–i.e., the k that minimizes τ_{new}–and then compare it to τ_{old} to determine if splitting is beneficial.

We recursively apply the decision process to achieve fine-grained kernel clustering with bounded error. We conclude the section by providing a proof that any union of sampling error-bounded cluster sets also maintains bounded error, provided the same sample sizes are used across the clusters. By applying Theorem 1 to each cluster set, we ensure that the total sampling error across all clusters remains bounded by ϵ.

Theorem 1. *Let* $S^{(0)} = \{C_0^{(0)}, C_1^{(0)}, ...\}$, $S^{(1)} = \{C_1^{(1)}, C_1^{(1)}, ...\}$, ..., $S^{(N-1)} = \{C_0^{(N-1)}, C_1^{(N-1)}, ...\}$ *be* N *sets of kernel clusters where the corresponding sampling error of each cluster set is bounded by* ϵ *when sample sizes* $\{m_i^{(j)}\}$ *are used for cluster set* $S^{(j)}$*. Then, the union of every cluster set* $\bigcup_{j=0}^{N-1} S^{(j)}$ *also gives a bounded sampling error when the same set of sample sizes* $\bigcup_{j=0}^{N-1} \{m_i^{(j)}\}$ *are used.*

Proof. The proof is found in Appendix VIII-B □

D. Sampling GPU Kernels from clusters

We initiate kernel sampling when a cluster cannot be further split to achieve more simulation speedup. Clusters with red outlines in Figure 6 indicate where recursion has halted due to the condition $\tau_{old} < \tau_{new}$. For each cluster, random sampling with the sample sizes m_i from STEM is performed.

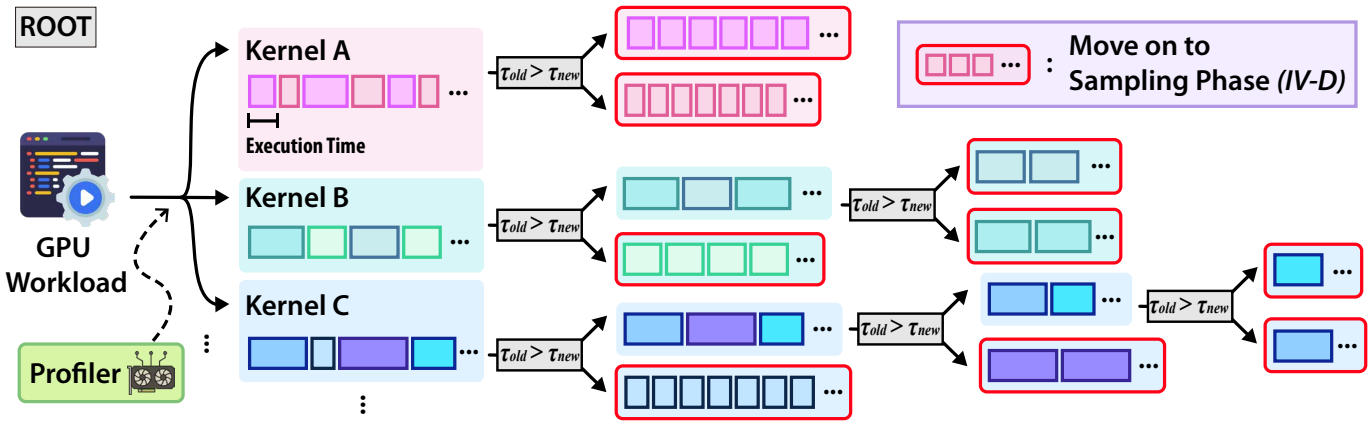

Fig. 6. ROOT's recursive greedy methodology is depicted. First, the GPU workload is initially split into clusters by kernel name. Next, k-means clustering is applied to divide these clusters into subclusters. ROOT then evaluates whether to proceed with splitting by comparing the estimated simulation time; splitting only occurs if it yields a greater speedup. If not (red outlines), the recursion stops and ROOT moves to the sampling phase, where the sample size by STEM is applied.

V. Evaluation

A. Experiment Setups

Experiment Environment. We used an NVIDIA RTX 2080 GPU to evaluate our sampling methodology, along with NSYS [24] as the GPU runtime profiler.

Benchmark Suites. We employed the Rodinia GPU Benchmark Suite 3.1 [3] and the CASIO DL Application Suite [5] for workloads comparing STEM with prior works. While Rodinia represents traditional GPU workloads, CASIO focuses on state-of-the-art ML applications. We also added 6 ML workloads (Bert, Bloom, Deit, Gemma, GPT-2, Resnet50) from Huggingface [14]; details shown in the Appendix VIII-C.

Baseline Methods. For baseline kernel sampling methods, we compared PKA and Sieve with the results of our work. We used the NCU profiler to gather the hardware information required by these methods. Only the kernel sampling part of PKA was implemented, as intra-kernel sampling is outside the scope of this paper. The Hugging Face workloads are too large to run NCU, which is used in PKA and Sieve, so we only compare our method with random sampling, where each kernel is randomly sampled with a probability of 0.1%.

Replication & Hyperparameters. To minimize randomness in the clustering methods, we repeated each experiment 10 times and averaged the results. The harmonic mean was used for speedup [9], while the arithmetic mean was used for sampling error. We set the error bound ϵ to 0.05 and used $k = 2$ for k-means clustering in ROOT.

STEM and ROOT Methodologies. We considered two methods: one using only STEM (labeled **STEM-only** in the figures), and the other using both STEM and ROOT (labeled **STEM+ROOT**). The STEM-only method does not employ ROOT in making decisions about splitting clusters into subclusters; instead, it splits a cluster if the sample size m exceeds 50. Since STEM-only applies a single general criterion for clustering, the combination of STEM and ROOT produces much finer-grained clustering results. Additionally, STEM+ROOT leverages the Gaussian assumption (Sec. IV-A), omitting the $\min\{\cdot, 30\}$ function in Equation (3) to achieve greater speedup. Our evaluation results indicate that this assumption is valid, as the error difference between STEM and STEM+ROOT is negligible.

B. Speedup and Error validation

Figure 7 and 8 summarize the evaluations of STEM and ROOT in comparison to prior kernel sampling methods. The exact numbers for the average speedup and error are shown in Table II. We discuss the findings separately for each benchmark suite.

Rodinia Suite: Since most Rodinia workloads do not consist of a large number of kernel calls, kernel-level sampling methods typically show lower speedups. The average speedups for PKA, Sieve, STEM-only, and STEM+ROOT were 6.20×, 4.77×, 1.16×, and 2.48×, respectively. The speedup numbers suggest applying kernel sampling alone appears less beneficial for performance gains. In terms of error, the difference between prior works and STEM+ROOT is significant, reducing sampling error from 47.73% and 27.46% to just 6.72%.

It is indeed challenging to achieve significant speedups with Rodinia due to its relatively small workload footprint, highlighting that the suite has already been widely employed in most cycle-level simulators [4], [15]. However, as shown in Figure 7 and 8 in specific benchmarks like `gaussian`, `lud_cuda`, `sc_gpu`, and others, STEM+ROOT achieves a comparable degree of speedup to prior works while maintaining near-zero sampling error. For example, STEM and ROOT reduced the sampling error by 21.88×, 108.42×, and 18.72× compared to PKA across the three benchmarks, along with comparable speedups.

Casio Suite: We proceed to evaluate our method on the latest ML benchmarks, which involve a high number of kernel calls, allowing STEM to fully utilize its statistical modeling capabilities. PKA and Sieve, although capable of achieving significant speedups by sampling only one kernel per cluster, suffer from large sampling errors. Initially, using the STEM-only strategy, we achieved an average speedup of 8.07× with

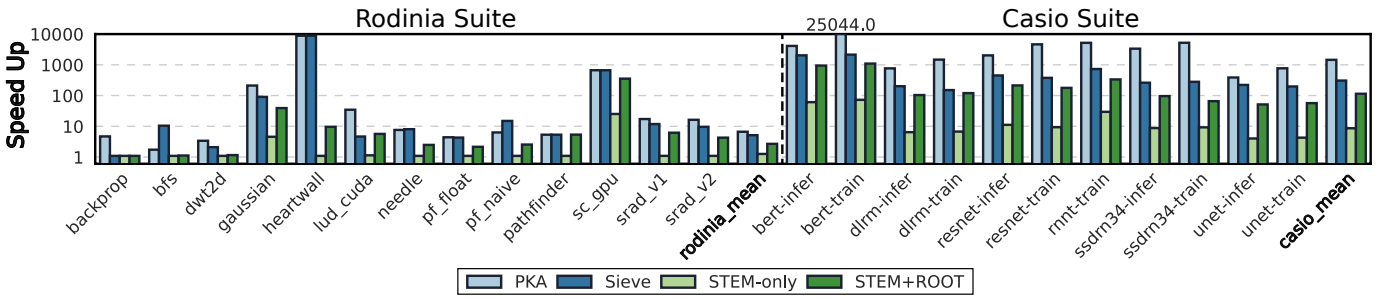

Fig. 7. Speedup comparison of four kernel sampling methods on the Rodinia and Casio GPU benchmark suites. The speedup is presented in log-scale, with each bar representing the speedup of a specific sampling method and the average speedup across all benchmarks shown on the far right.

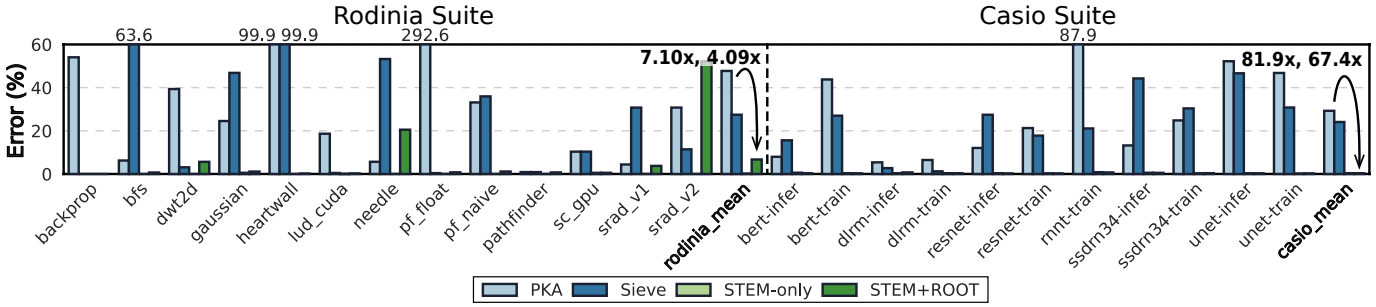

Fig. 8. Sampling error comparison of four sampling methods on Rodinia and Casio suites. Since STEM is based on rigorous error modeling, it achieves near-zero error but has a smaller speedup than prior works. After the ROOT is added to STEM, it achieves comparable speedup to prior works while reducing the sampling error to near-zero.

a sampling error of 0.37%. Although the error is small enough to make the sampled simulation trustworthy, the speedup was insufficient compared to previous works. Since STEM-only always samples more than 30 kernels per cluster to ensure the Gaussian sample mean, additional overhead is added compared to prior works.

By applying the STEM+ROOT method, we achieved a 109.60× speedup with an even smaller error of 0.36%. Notably, the fine-grained approach reduced the error further than STEM-only, despite accepting the Gaussian assumption in Equation (3). Additionally, STEM+ROOT achieved comparable speedup while reducing the error by 81.89× and 67.40× compared to PKA and Sieve, respectively. STEM and ROOT greatly improve the reliability of efficiency kernel sampling in cycle-level GPU simulations.

VI. Discussion

A. Tradeoff between Speedup and Error

Speedup and error are always in a tradeoff relationship in kernel sampling; as more kernels are sampled, the speedup decreases but the sampling error becomes smaller. The challenge lies in sampling as few kernels as possible while keeping the error small enough to ensure the reliability of the sampled simulation. Prior works, like PKA and Sieve, fail in this regard because their sampling errors are too large to trust their results. In many cases, these methods result in errors greater than 10%, which is too significant, rendering kernel sampling for cycle-level simulations practically meaningless.

Our method, on the other hand, excels by only slightly increasing the sample size while achieving near-zero sampling error, significantly outperforming prior works. Figure 9 shows the scatter plot where the x-axis represents the speedup in log scale and the y-axis indicates the sampling error in percentage. The mean values for each sampling method are marked with black $\times$'s. The plot demonstrates that by using STEM, we are able to achieve near-zero sampling error owing to its rigorous error modeling. However, STEM on its own involves larger sample sizes, resulting in lower speedups. To overcome this limitation, the fine-grained clustering from ROOT significantly reduces the sample size, achieving a speedup comparable to prior works while keeping the sampling error low.

Figure 10 also evaluates our method on larger ML workloads, which are so extensive that hardware-level profilers like NCU would take months to complete. This makes approaches such as PKA and Sieve impractical, so we compared our method only with random sampling, where each kernel is sampled with a 0.1% chance. The results show that STEM alone achieves a similar degree of speedup as random sampling but with much smaller error. When ROOT is combined with STEM, despite a slight sacrifice of 0.35% in error, we achieve a significant 22.84× speedup. This demonstrates that our method, STEM with ROOT, performs exceptionally well in these larger workloads.

B. Using kernel execution time as a metric

Using microarchitecture-dependent metrics for workload sampling is not an ideal in workload sampling, as samples may not accurately represent the entire workload when microarchitecture changes. However, if state-of-the-art methods employing architecture-independent metrics already exhibit substantial errors across many workloads, this concern becomes less significant. While numerous prior sampling techniques for

TABLE II
AVERAGE SPEEDUP AND ERROR OF FOUR KERNEL SAMPLING METHODS ON THREE GPU BENCHMARK SUITES. THE HUGGINGFACE SUITE WAS NOT TESTED WITH BASELINE METHODS DUE TO EXCESSIVE PROFILING OVERHEAD.

Benchmark Suites	Rodinia		Casio		Huggingface	
	Speedup (×)	Error (%)	Speedup (×)	Error (%)	Speedup (×)	Error (%)
PKA (baseline)	6.195	47.732	1425.006	29.260	–	–
Sieve (baseline)	4.77	27.46	296.823	24.085	–	–
STEM-only	1.157	0.085	8.071	0.370	1388.638	0.213
STEM+ROOT	2.481	6.720	109.595	0.357	31719.057	0.565

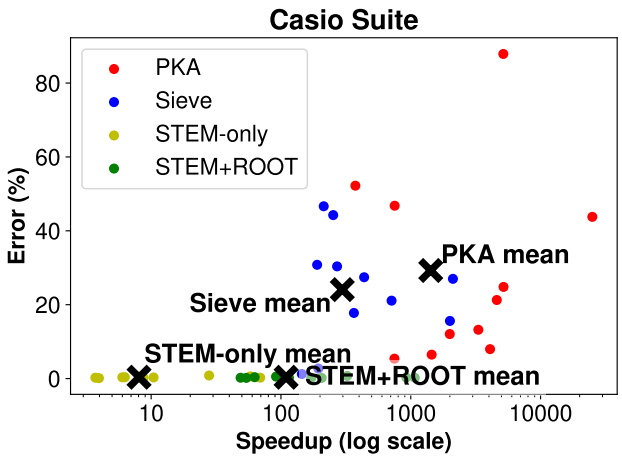

Fig. 9. Scatter plot showing the speedup (log scale) and error (%) of four kernel sampling methods using the Casio benchmark suite. The black X's indicate the mean performance of each method.

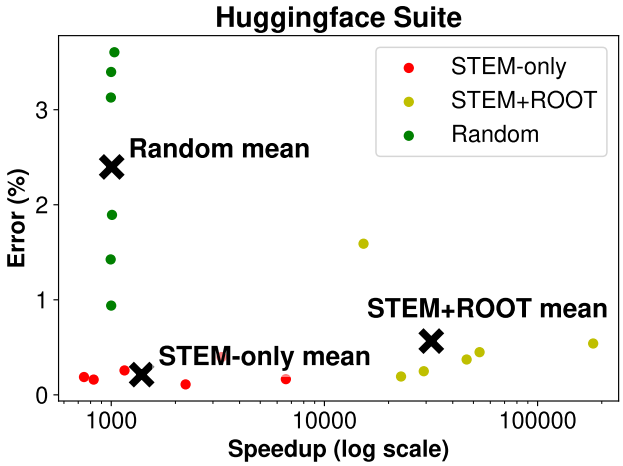

Fig. 10. Scatter plot of four kernel sampling methods using the HuggingFace benchmark suite. Due to the high hardware profiling overhead of PKA and Sieve, we compared STEM only with random sampling.

both CPU and GPU rely on architecture-independent metrics, we argue that these metrics alone cannot lead to accurate methodologies due to the heterogeneous runtime behavior of GPU workloads. For instance, Figure 4 shows that a single cluster in Sieve contains kernels with the same name and a low coefficient of variation (CoV) in runtime instruction counts, yet their execution distributions are highly diverse.

In response, STEM utilizes kernel execution time as the metric for both kernel clustering and sampling. We showed that this approach significantly reduces sampling error, as demonstrated in Figure 8. By designing a rigorous models based on execution time, we developed STEM and ROOT to establish a robust kernel sampling methodology. Additionally, using kernel execution time offers other advantages, such as reducing the profiling overhead. We successfully applied this approach to the Huggingface suite, which was previously unmanageable with prior works due to its large workload size.

C. Assuming Gaussian kernel distributions

The reason STEM+ROOT exceeds the 5% error bound in some Rodinia benchmarks is as it ignores the $\min\{\cdot, 30\}$ term by assuming the sample mean will follow a Gaussian distribution. Evidence for this can be seen in the STEM-only results, since the error remained within the bound due to the $\min\{\cdot, 30\}$ term. However, in most cases, the error difference between STEM-only and STEM+ROOT was negligible: in fact, the mean error for STEM+ROOT in the CASIO suite was smaller. These evaluation results suggest that our Gaussian assumption holds in most cases, allowing us to achieve greater speedup with no significant sacrifice in sampling error.

D. Limitations and Future works

STEM and ROOT's kernel sampling methods, as well as other prior kernel sampling approaches, rely on the assumption that the GPU workload operates using a single stream of kernels. Under this assumption, simulating a subset of kernels provides accurate estimations. However, if the workloads utilize multiple streams with asynchronous kernel calls or employ techniques like CUDA Graphs–which launch multiple kernels while managing dependencies–the performance estimation in Equation (1) and the sampling error calculation in Equation (2) are no longer applicable. Since dependencies and overlaps between kernels can result in vastly different total execution times for workloads, a new approach is necessary for sampling such workloads and accurately estimating their total execution time. A similar challenge arises in the multi-GPU domain; when multiple streams or devices are involved, kernel dependency information, as well as synchronization and communication overhead, must be factored in to achieve accurate performance modeling.

Moreover, no kernel-level sampling work, including this one, has addressed the challenges posed by ML compilers, architecture-dependent kernels, and hardware-level optimizations in ML workloads. This presents a particularly difficult problem as modern ML compilers in Python-based libraries optimize workloads through a complex stack of intermediate representations, generating different kernels based on architecture and workload size. As a result, any kernel sampling methodology, including ours, must rely on runtime profiling for each specific GPU architecture to obtain accurate kernel sampling data that reflects the hardware configurations.

Fortunately, our work leverages lightweight runtime profilers like NSYS, minimizing profiling overhead compared to prior works that utilize more intensive profilers like NCU.

VII. Conclusion

This paper presents STEM, a statistical error modeling technique for accurate kernel sampling in cycle-level GPU simulations. By rigorously modeling execution time, STEM significantly reduces sampling error compared to prior methods. Additionally, we propose ROOT, a novel fine-grained hierarchical kernel clustering methodology that leverages STEM to achieve accurate sampling while maintaining error bounds. Our evaluation shows that STEM and ROOT achieve speedups of 2.48×, 109.60×, and 31719.1× with errors of 6.72%, 0.36%, and 0.57%, respectively across three GPU benchmark suites. These errors are up to 81.9× smaller than those of prior works in the CASIO suite, and we successfully applied our approach to the Huggingface suite, where previous methods failed due to excessive profiling overhead. Our results demonstrate that with STEM and ROOT, kernel sampling can significantly reduce the time required for cycle-level GPU simulations while maintaining high accuracy.

VIII. Appendix

A. Solution for ***Problem 1.***

Let $a_i \equiv \mu_i$, $b_i \equiv N_i^2\sigma_i^2$, and $c \equiv (\epsilon \sum_i N_i\mu_i/z_{1-\alpha/2})^2$ for simplicity. Then, the **Problem 1** becomes as below:

$$\begin{aligned} \underset{m_i}{\text{minimize}} \quad & \sum_i a_i m_i \\ \text{subject to} \quad & \sum_i \frac{b_i}{m_i} - c \le 0 \\ \text{and} \quad & m_i > 0 \ \text{ for } \ \forall i \in \{0...k-1\}. \end{aligned}$$

The corresponding Lagrangian function $\mathcal{L}$ can be written as follows:

$$\mathcal{L}(\mathbf{m}, \lambda) = \sum_i m_i a_i + \lambda_k \cdot (\sum_i \frac{b_i}{m_i} - c) + \sum_i \lambda_i \cdot (-m_i).$$

The solution $\mathbf{m}^*$ must satisfy the following four Karush–Kuhn–Tucker (KKT) conditions:

- Stationary Condition: $\nabla\mathcal{L}(\mathbf{m}^*; \lambda) = \mathbf{0}$ (a)
- Primal Feasibility:
  $\sum_i b_i/m_i^* - c \le 0$ (b)
  and $(-m_i^*) \le 0$ for $\forall i \in \{0...k-1\}$ (c)
- Dual Feasibility: $\lambda_i \ge 0$ for $\forall i \in \{0...k\}$ (d)
- Complementary Slackness:
  $\lambda_k \cdot (\sum_i b_i/m_i^* - c) + \sum_i \lambda_i \cdot (-m_i^*) = 0$ (e)

From (b), (c), and (d), we can see that in each term either one of λ_i or the multiplied term should be zero. Since we are assuming $m_i > 0$, $\lambda_i = 0$ for $\forall i \in \{0...k-1\}$.

Also, from (a), $a_i - \lambda_k b_i/(m_i^*)^2 - \lambda_i m_i^* = 0$.

Since $a_i \neq 0$, $\lambda_k \neq 0$ and thus the equality of (b) holds and thus $m_i^* = \sqrt{\lambda_k b_i/a_i}$ for $\forall i \in \{0...k-1\}$.

By putting this into (b), we obtain $\sum_i \sqrt{a_i b_i/\lambda_k} = c$ and thus $\lambda_k = (\sum_i \sqrt{a_i b_i}/c)^2$. Therefore, the solution to the non-linear optimization problem is:

$$m_i = \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \text{ for } \forall i \in \{0...k-1\}.$$

B. Proof of ***Theorem 1***

Proof. By the definition of sampling errors,

$$\sum_i (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}} \le \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i^{(j)} \mu_i^{(j)} \right)^2 \quad (4)$$

satisfies for arbitrary $\forall j \in \{0, ..., N-1\}$.

Since $\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i^{(j)} \mu_i^{(j)}$ is positive for every j, we can apply the following inequality when summing up the inequality (4) for all j's:

$$\sum_j x_j^2 \le \left(\sum_j x_j \right)^2 \text{ when } x_j \ge 0 \text{ for } \forall j.$$

Therefore, summing up (4) by j gives

$$\begin{aligned} \sum_{ij} (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}} &\le \left(\frac{\epsilon}{z_{1-\alpha/2}} \right)^2 \sum_j \left(\sum_i N_i^{(j)} \mu_i^{(j)} \right)^2 \\ &\le \left(\frac{\epsilon}{z_{1-\alpha/2}} \right)^2 \left(\sum_{ij} N_i^{(j)} \mu_i^{(j)} \right)^2 . \quad (5) \end{aligned}$$

The sum $\sum_{ij}$ in (5) is the same as summing through every cluster in the union set $\bigcup_{j=0}^{N-1} \{C_{i_j}^{(j)}\}$. By substituting

$$\tilde{\mu} = \sum_{ij} N_i^{(j)} \mu_i^{(j)} \text{ and } \tilde{\sigma}^2 = \sum_{ij} (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}},$$

we can transform (5) into the following inequality

$$\left| \frac{(\tilde{\mu} + z_{1-\alpha/2}\tilde{\sigma}) - \tilde{\mu}}{\tilde{\mu}} \right| \le \epsilon,$$

which infers that the union of cluster sets also gives bounded sampling error under $1-\alpha$ confidence interval.

□

C. Huggingface workload configurations

Below are the URLs to Huggingface repositories of the models and datasets that we used for the Huggingface benchmark suite.

- Image Dataset [12]: A subset of 10 classes from Imagenet [6]. Link
- Bert [2]: Pytorch model converted from the official Google's implementation to classify 10,000 premise-hypothesis pairs for sequence classification. Link
- Bloom [27]: Use 8-bit BLOOM model to generate 100 sentences with a max token length of 100. Link
- Deit [32], [34]: Data-efficient Image Transformer for classifying 3,925 images from imagenette dataset. Link
- Gemma [31]: 2B base version of the Gemma model by Google, generates 1,000 sentences. Link
- GPT-2 [25]: GPT-2 model by OpenAI, generates 1,000 sentences. Link
- ResNet50 [11]: Classifes 13,400 images from imagenette dataset. Link

REFERENCES

[1] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled gpu workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 724–737. [Online]. Available: https://doi.org/10.1145/3466752.3480100

[2] P. Bhargava, A. Drozd, and A. Rogers, "Generalization in nli: Ways (not) to go beyond simple heuristics," 2021.

[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing." Piscataway, NJ, USA: IEEE, 2009, pp. 44–54.

[4] E. Chung, "Macsim user guide and available traces," https://github.com/gthparch/MacSim-User-Guide, 2024.

[5] M. Davies, I. McDougall, S. Anandaraj, D. Machchhar, R. Jain, and K. Sankaralingam, "A journey of a 1,000 kernels begins with a single step: A retrospective of deep learning on gpus," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 20–36. [Online]. Available: https://doi.org/10.1145/3620665.3640367

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[7] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, "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*, ser. PACT '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 353–364. [Online]. Available: https://doi.org/10.1145/1854273.1854318

[8] L. Eeckhout, *Computer Architecture Performance Evaluation Methods*, 1st ed. Springer Cham, 2022.

[9] L. Eeckhout, "Rip geomean speedup use equal-work (or equal-time) harmonic mean speedup instead," *IEEE Computer Architecture Letters*, 2024.

[10] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program phase analysis," *J. Instr. Level Parallelism*, vol. 7, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:11937761

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] J. Howard, "Imagenette: A smaller subset of 10 easily classified classes from imagenet," March 2019. [Online]. Available: https://github.com/fastai/imagenette

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

[14] Huggingface, "Huggingface," https://huggingface.co/.

[15] 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.

[16] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, "Macsim: A cpu-gpu heterogeneous simulation framework user guide."

[17] D. M. Lane, "Onlinestatbook: Sampling distribution of the mean," https://onlinestatbook.com/2/sampling_distributions/samp_dist_mean.html.

[18] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[19] C. Liu, Y. Sun, and T. E. Carlson, "Photon: A fine-grained sampled simulation methodology for gpu workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1227–1241. [Online]. Available: https://doi.org/10.1145/3613424.3623773

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

[21] M. Naderan-Tahan, H. SeyyedAghaei, and L. Eeckhout, "Sieve: Stratified gpu-compute workload sampling," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 224–234.

[22] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2024.

[23] NVIDIA, "Nvidia nsight compute," https://developer.nvidia.com/nsight-compute.

[24] NVIDIA, "Nvidia nsight systems," https://developer.nvidia.com/nsight-systems.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[26] A. Sabu, H. Patil, W. Heirman, and T. E. Carlson, "Looppoint: Checkpoint-driven sampled simulation for multi-threaded applications," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 604–618.

[27] T. L. Scao and A. F. et al., "Bloom: A 176b-parameter open-access multilingual language model," 2023.

[28] J. Soch, "The book of statistical proofs," https://statproofbook.github.io/P/norm-lincomb.

[29] 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: Association for Computing Machinery, 2019, p. 197–209. [Online]. Available: https://doi.org/10.1145/3307650.3322230

[30] E. Tanis and R. V. Hogg, *Probability and Statistical Inference*, 1977.

[31] G. Team, T. Mesnard, and C. H. et al., "Gemma: Open models based on gemini research and technology," 2024.

[32] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers and distillation through attention," 2021.

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

[34] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," 2020.

[35] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "Smarts: accelerating microarchitecture simulation via rigorous statistical sampling," in *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, 2003, pp. 84–95.