Accelerating GPU Simulations

About me

- Euijun Chung (Pronounced E-June)
- 2nd-year PhD student from Prof. Kim’s lab
- Research interests: GPU simulation and performance modeling, ML workloads for GPU, etc.

Georgia Tech

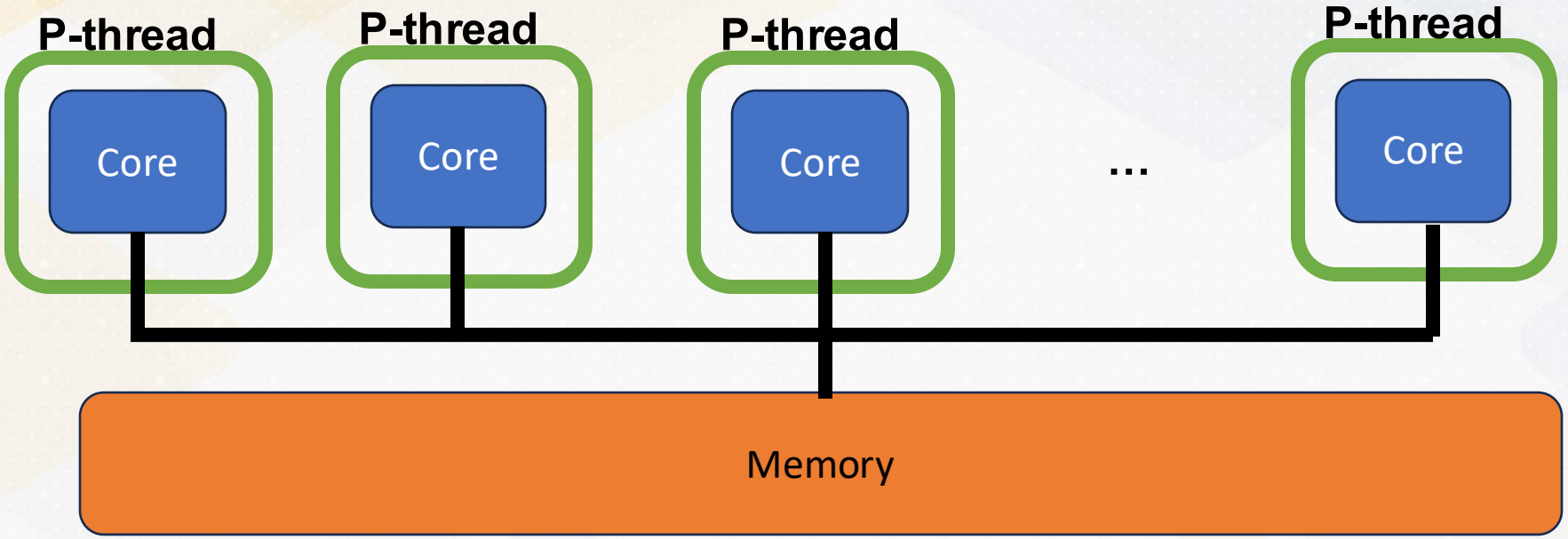
Challenges of Cycle Level Simulation

- Cycle level simulation takes **too long**.
- If a simulation speed is 10KIPS (10 instructions per sec), simulating 1B instructions (1 sec in a real machine) takes ~= 28 hours
- ML workload takes hours: 10 hours of ML workloads → 100 years of simulation

Accelerating Simulations

- Accelerating the simulation itself
  - Parallelizing the simulator
  - Event driven simulation
  - Simplifying the model
  - Sampling
  - Statistical Modeling
  - ML based Modeling
- Reducing the workload size
  - Micro benchmarks
  - Reducing the workload size
  - Create small representative workloads

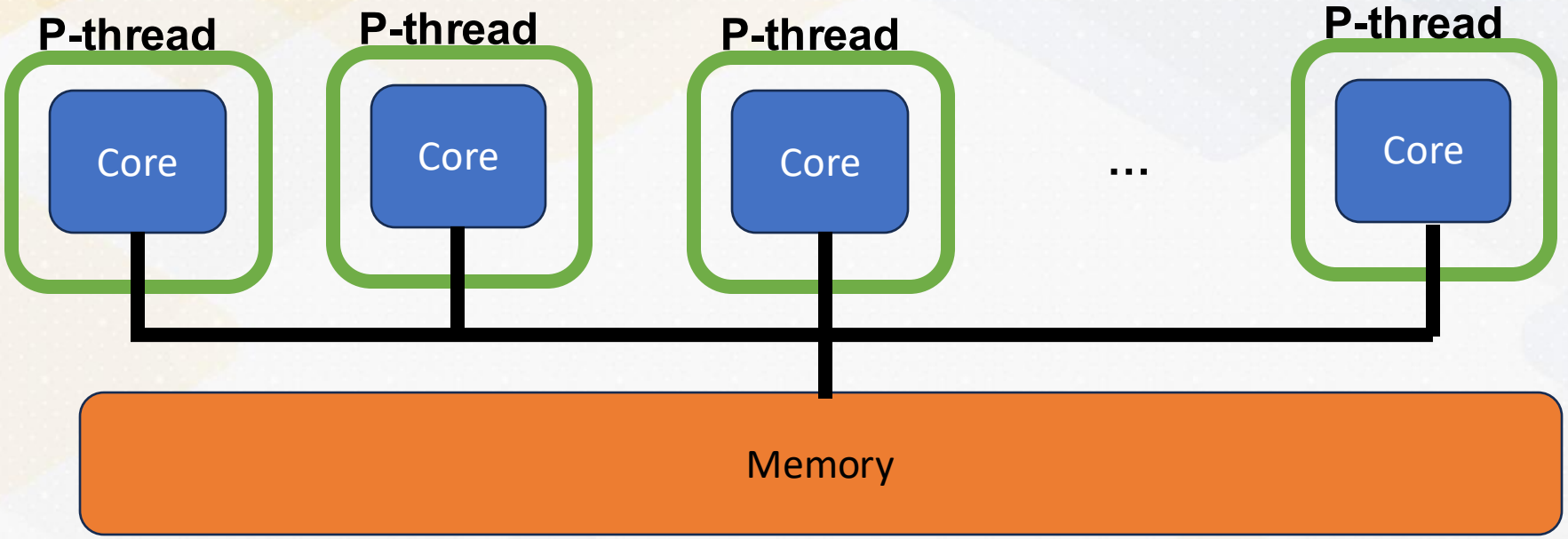
Parallelizing Simulator



- Each P-thread simulates a core. Cores are typically run in parallel so it can be parallelized.
- Memory/Noc (network on chip) needs to be communicated.
- Bottlenecks on the memory

Georgia Tech

Parallelizing Simulator

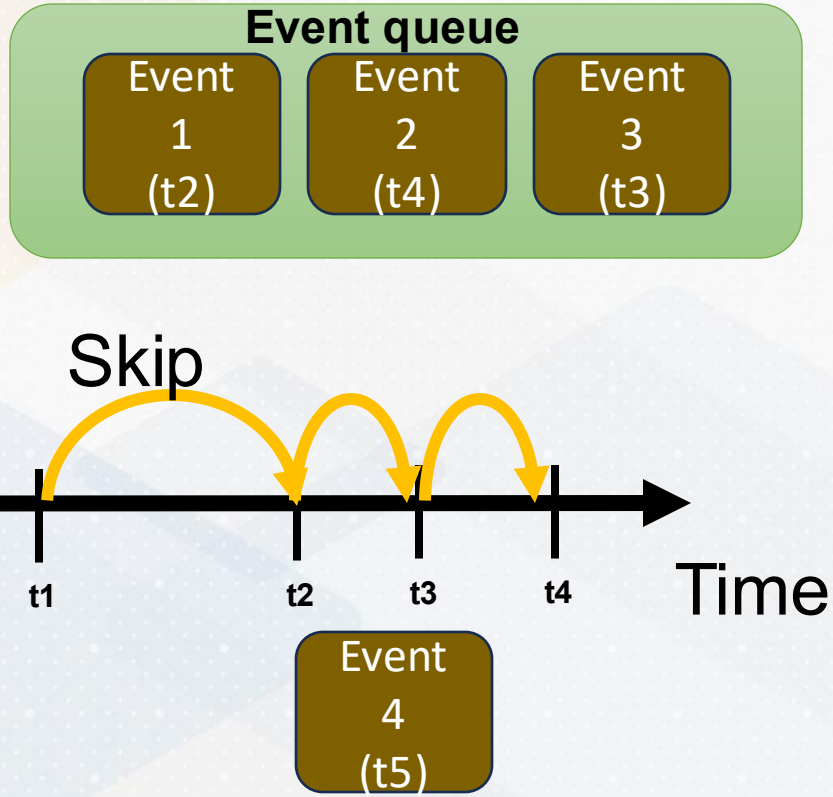


- Parallelizing a modern GPU simulator [Arxiv ‘25]
- Extended GPGPU-sim from 1 thread to 16 threads, average 5.8x speedup

Georgia Tech

Event Driven Simulation

- Instead of operating by cycle, simulator is operated with events.
- Speeding up long latency operation simulations is possible.
- New event is inserted into the event queue.



Georgia Tech

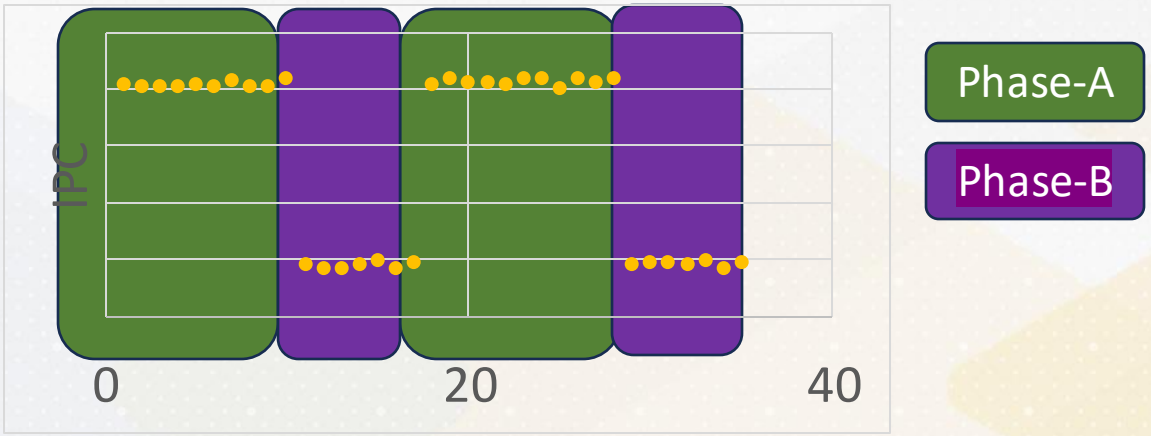
Simplifying Models

- Depending on which one to model, we could simplify the modeling.
- Non-critical components are modeled based on the average throughputs and average latency.
- Detailed modeling is needed for modeling any resource contentions.
- Option #1: simplify the pipeline: instead of modeling instruction fetch/decode/execution, we could assume IPC = issue width. Model only caches and memory. E.g.) z-sim[1] CPU simulator simplifies the core pipeline.
- Option #2: simplify the memory model: assume memory system has a fixed latency

[1] Sánchez, D., & Kozyrakis, C. (2013). ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA 2013* (pp. 475-486). ACM.

Sampling Techniques

- Random Sampling: simple; randomly choose where to simulate.
  - Execution driven: fast forward the part that won't be simulated or use a check-point based method
  - Trace driven: generate traces only for simulating sessions
- Handling state information: e.g.) cache, branch predictors → warm up time is needed; or running time is sufficiently long enough to overcome
- Program phase-based sampling (e.g., Simpoint[1])



[1] Sherwood, T., Perelman, E., Hamerly, G., & Calder, B. (2002). Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*.

Georgia Tech

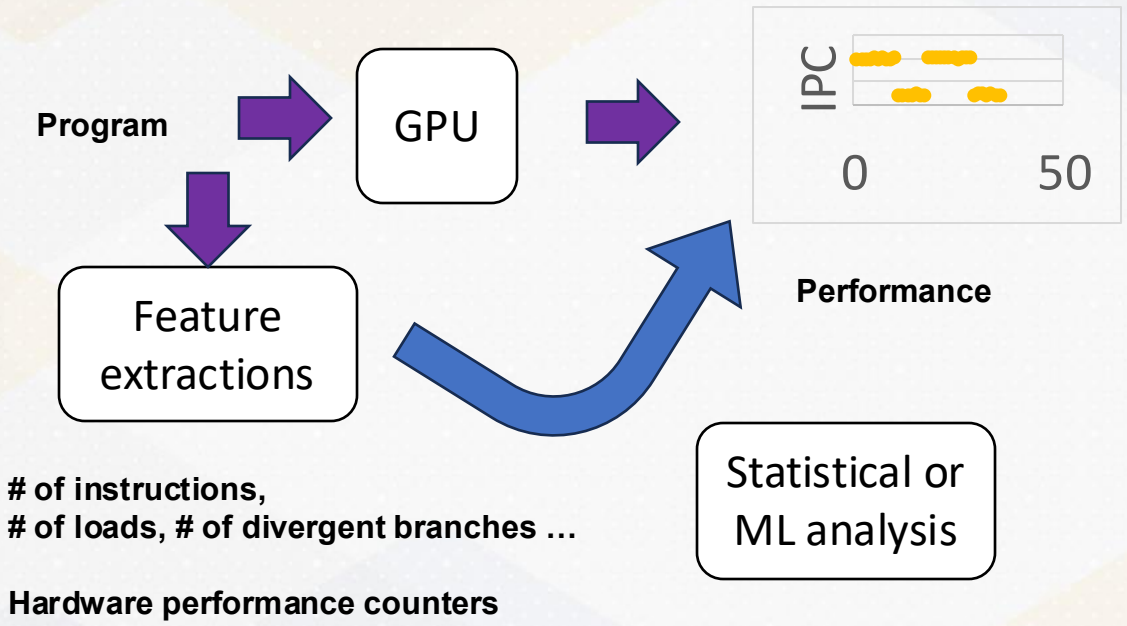
GPU Sampling Techniques

- Phase-based sampling within the program is non-trivial
  - Execution length for one single thread is too short.
- Solutions:
  - CUDA block level sampling: simulate 100 CUDA blocks instead of 1000s of CUDA blocks
  - Kernel level sampling: simulate 1-2 kernels instead of 10s of kernels
  - Warp level sampling: reduce the number of warps to simulate

Reducing Workloads

- Method #1: reducing iteration counts
  - E.g.) Machine learning workloads run 1000 iterations. Each iteration shows very similar characteristics; instead of 1000 iteration, iterates only once.
- Method #2: reducing input sizes
  - E.g.) Graph algorithm traverses for 1B nodes → graph algorithm traverses for 1M nodes
- Method #3: identify dominant kernels
  - E.g.) A program has 100s of functions. One or two functions dominate 90% of application time; change the application to run only those two functions.

Data-Driven Modeling



- E.g.) Cycle count = C1 x # instructions ^(exp1) + C2 x # FP instructions^(exp2) + C3 x # memory instructions^(exp3) etc.

Georgia Tech

Related works

Workload sampling

- TBPoint [IPDPS ‘14]
- PKA [MICRO ‘20]
- Sieve [ISPASS ‘23]
- Photon [MICRO ‘23]
- STEM [MICRO ’25] (my work!)

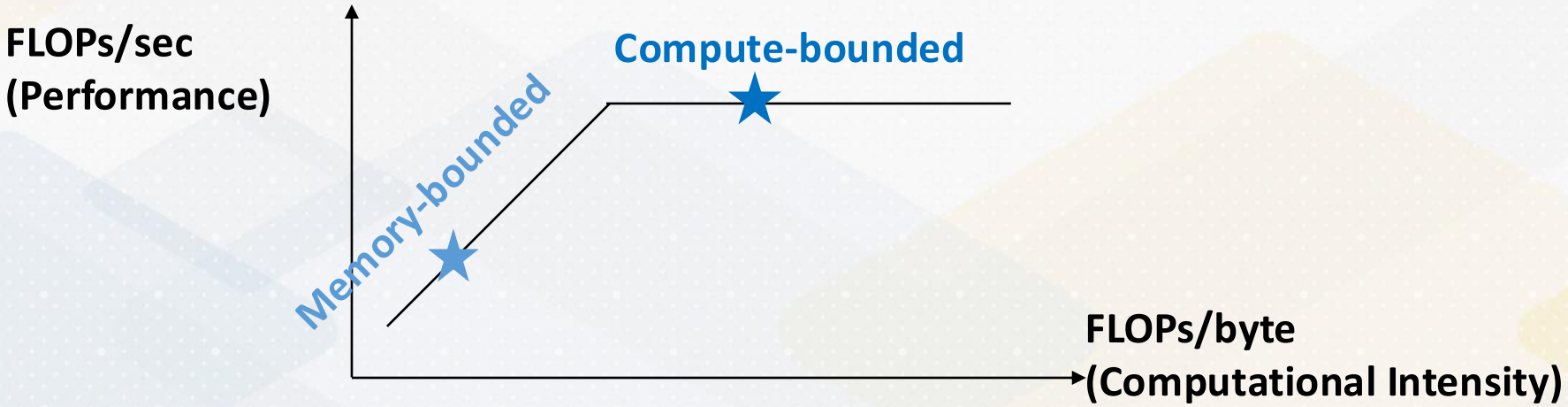
Other techniques

- Path Forward Beyond Simulators [MICRO ‘23]
- GPU Scale-Model Simulation [HPCA ‘24]

ML-based performance modeling

- Path Forward Beyond Simulators [MICRO ‘23]
- Performance model for GPU kernels in DNN workloads

- Takeaway: **Simple regression models** with **data size as an input** can predict the performance of ML workloads pretty well!



Georgia Tech

Fast simulators for ML workloads

- ML workloads are relatively easier to model the performance
- E.g. roofline models for GEMM (General Matrix Multiply) kernels

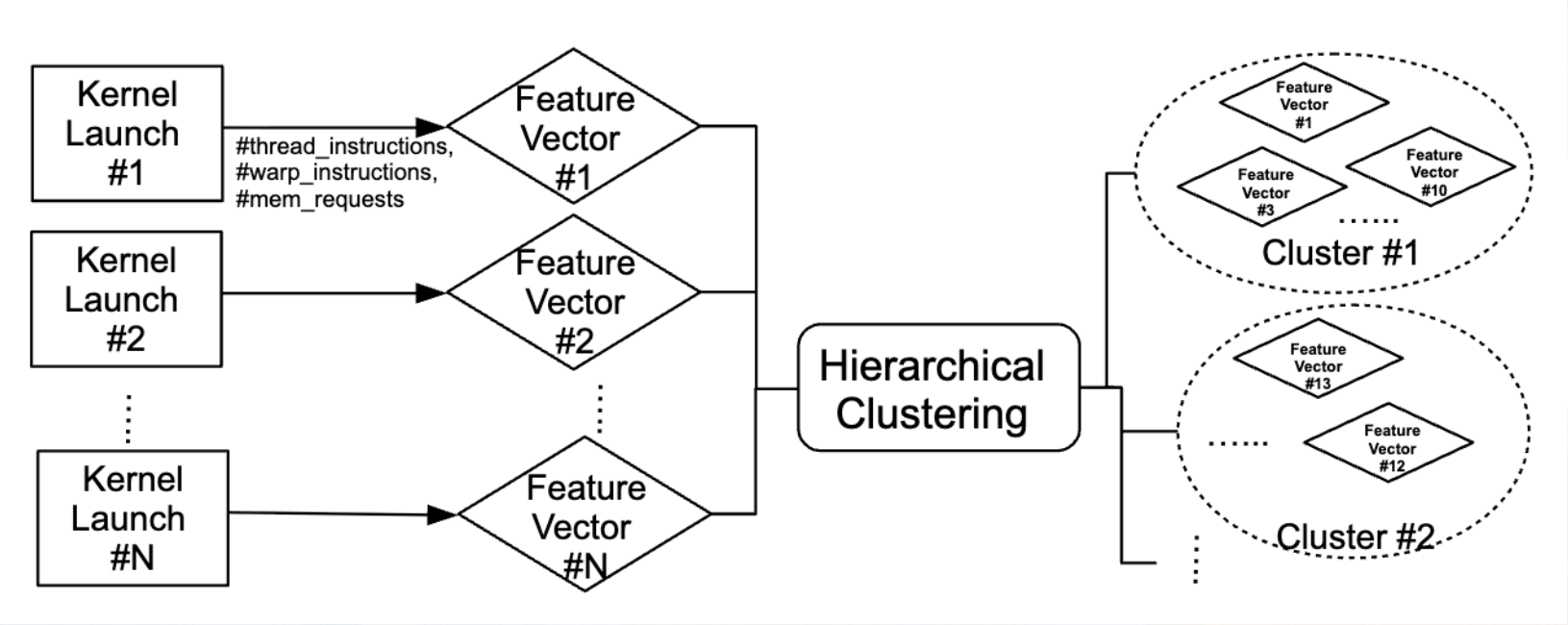
- ASTRA-Sim [ISPASS ‘23] – ML on Multi-GPU systems
- LLMSimulator [MICRO ‘24] – LLM inference on Multi-GPU systems
- TrioSim [ISCA ’25] – ML on Multi-GPU systems

- Event-drive, cycle-accurate simulators with **simple** performance models

Georgia Tech

TBPoint [IPDPS '14]

- Inter-kernel (kernel-level) sampling



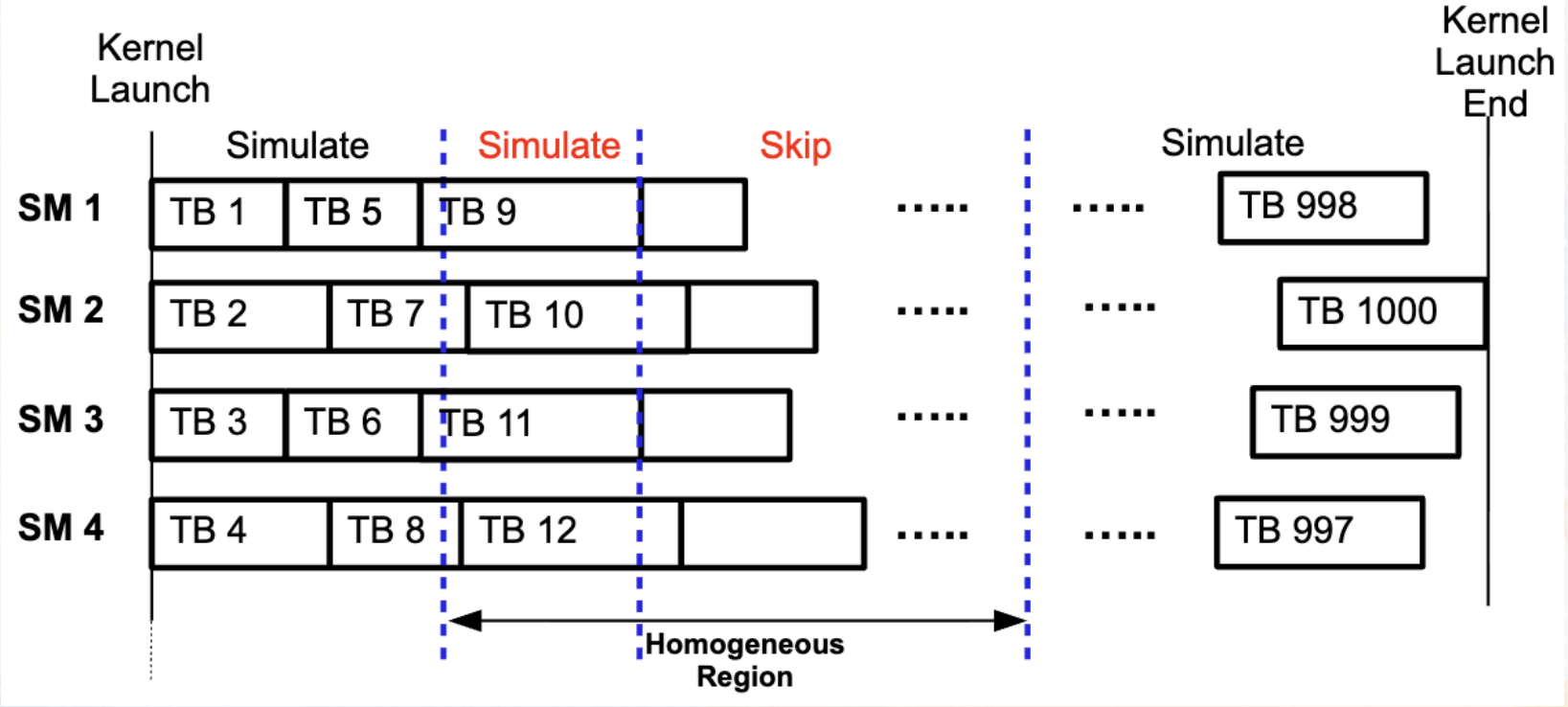
$$\text{inter\_feature\_vector}_i =< \text{Kernel\_Launch\_Size}, \text{Control\_Flow\_Divergence}, \text{Memory\_Divergence}, \text{Thread\_Block\_Variations} >$$
$$=< \frac{\#\text{thread\_insts}_i}{\text{avg\_thread\_insts}}, \frac{\#\text{warp\_insts}_i}{\text{avg\_warp\_insts}}, \frac{\#\text{mem\_reqs}_i}{\text{avg\_mem\_reqs}}, \text{CV\_TB\_size} >$$

1. Extract feature vector per each kernel
2. Use clustering to cluster similar kernels into groups

Georgia Tech

TBPoint [IPDPS '14]

- Intra-kernel sampling

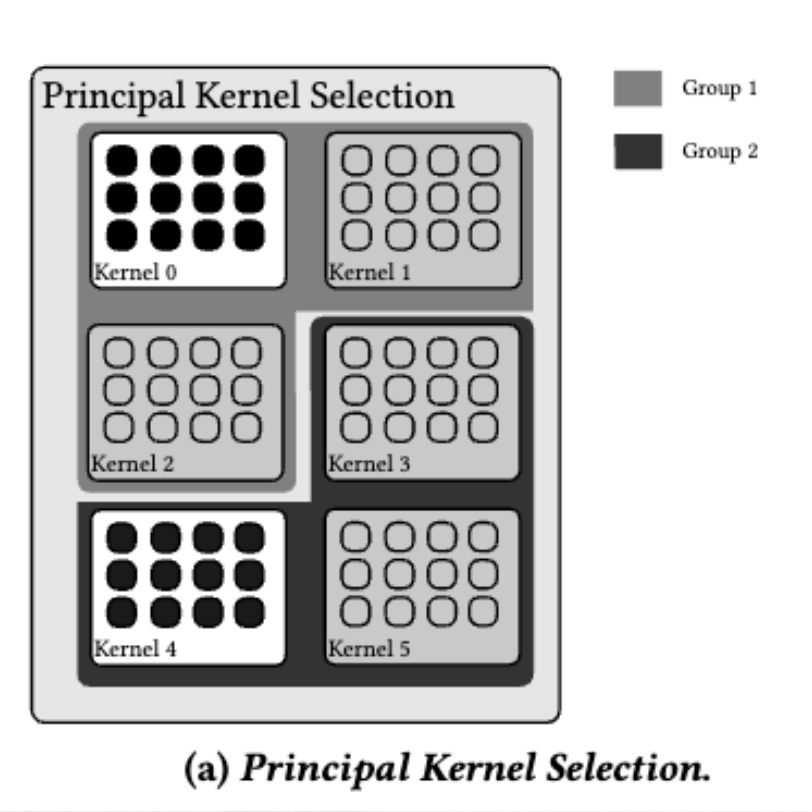


- Skip the homogeneous region
- Requires **full functional simulation** → Does not scale with ML workloads

Georgia Tech

PKA [MICRO '20]

- Inter-kernel (kernel-level) sampling: more metrics than TBPoint



(a) *Principal Kernel Selection.*

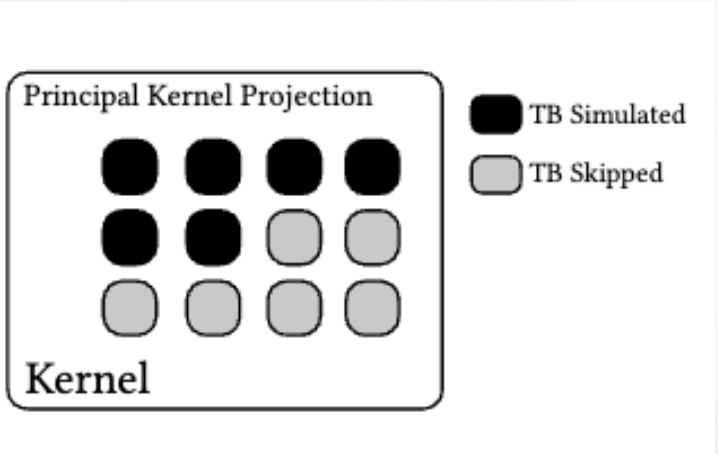
Metric
Coalesced global loads
Coalesced global stores
Coalesced local loads
Thread global loads
Thread global stores
Thread local loads
Thread shared loads
Thread shared stores
Thread global atomics
#Instructions
Divergence efficiency
#thread blocks

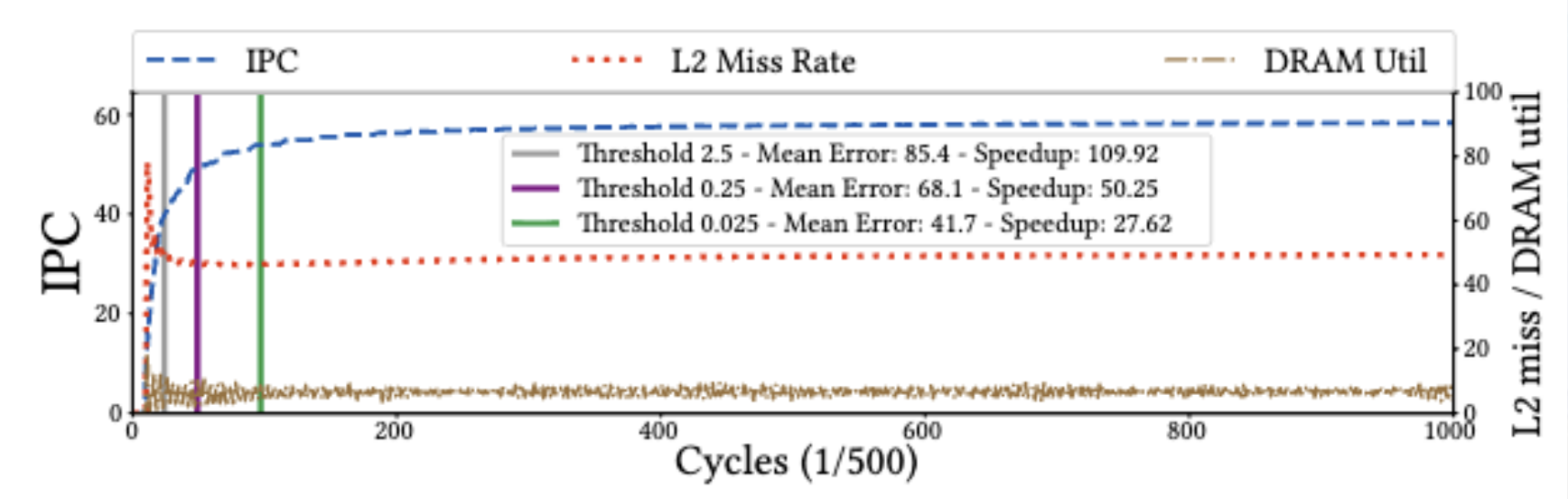
→ **K-mreans clustering**

Georgia Tech

PKA [MICRO '20]

- Intra-kernel sampling: early-stopping (more scalable than TBPoint)
- Detect a stable IPC, end simulation early and estimate the final statistics





Georgia Tech

Sieve [ISPASS ‘23]

- PKA’s limitation: too many metrics to profile!
- Nsight Compute incurs 100-1000x slowdown on most workloads
- Idea: **only use instruction count** for hand-tuned kernel clustering

- Categorize kernels into tiers:
- **Tier-1**: There is **no** variation in the number of instructions across invocations of the same kernels.
- **Tier-2**: There is **little** variation in the number of instructions across invocations of the same kernels.
- **Tier-3**: There is **large** variation in the number of instructions across invocations of the same kernels.

Georgia Tech

Sieve [ISPASS ‘23]

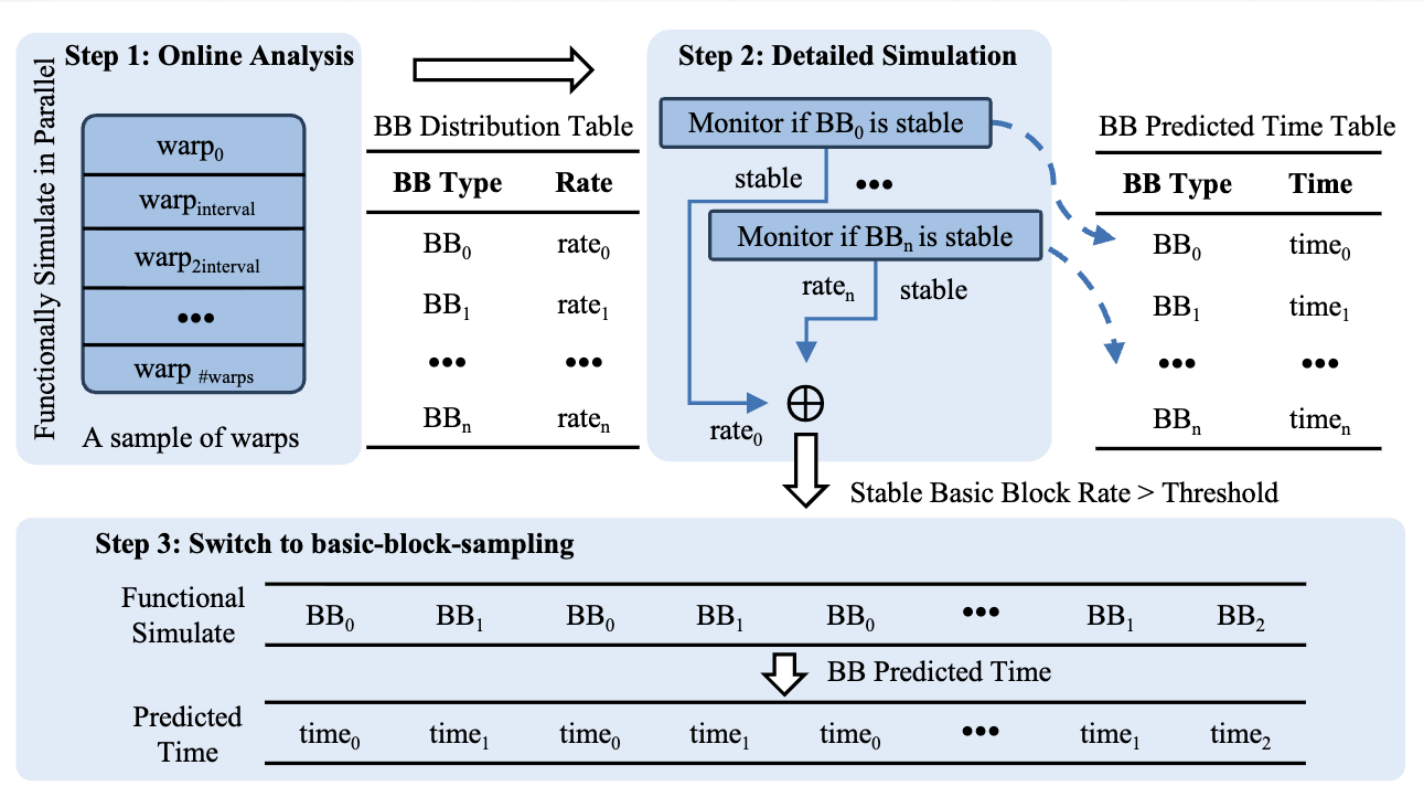
- **Tier-1**: There is **no** variation → sample the first kernel
- **Tier-2**: There is **little** variation → sample the first kernel
- **Tier-3**: There is **large** variation → split the group further with KDE (kernel density estimation), and sample the first kernel from each group

- Limitation:
- Sieve still needs instr-level counters
- Low accuracy on divergent workloads

Georgia Tech

Photon [MICRO '23]

- Strategy: fine-grained, online simulation skipping
- Similar to PKA's "if IPC is stable then skip simulation" method, but in three levels: BB (Basic block), Warp, and Kernel level.



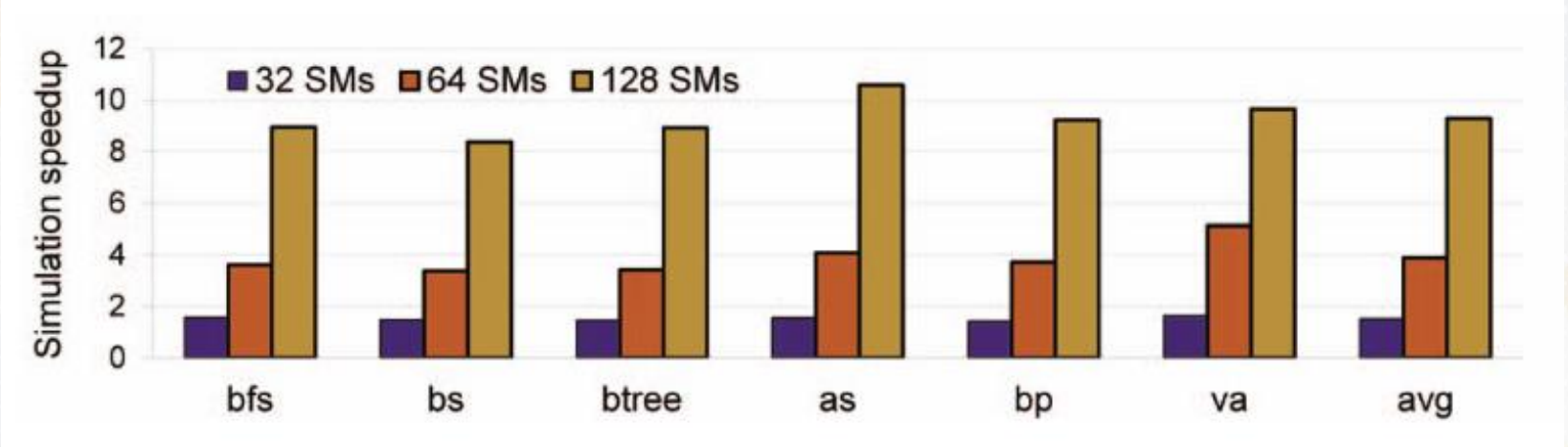
Georgia Tech

Photon [MICRO ’23]

- Uses online analysis during simulation → No offline step for profiling
- What about for the trace-based simulations? → Still needs full trace

GPU Scale-Model Simulation [HPCA '24]

- Estimating performance on large GPUs with small-GPU simulators
- **Strong-scaling** workloads: workload is fixed; independent of system size
- **Weak-scaling** workloads: the workload scales with system size



- The work provides simulation speedup (only) for weak-scaling workloads

Georgia Tech

Open research questions

- **Where should we simulate** if we have multiple concurrent GPU kernels? (e.g. Multi-GPU systems with computation & communication overlap)

- How can we **warm up the cache** in between the simulation points?

  e.g. L2$ in between kernels of full simulation and sampled simulation can affect the accuracy of workload sampling

- In CPUs, multiple techniques exist; fast-forwarding, checkpointing, etc.

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

Euijun Chung, Seonjin Na, Sung Ha Kang, Hyesoon Kim
Georgia Institute of Technology
2025 IEEE/ACM International Symposium on Microarchitecture

Georgia Tech

Cycle-level GPU microarchitecture simulation

Cycle-level simulations enable **fast validation** of new (micro)architecture designs.

Architectural design changes

Cycle-level GPU Simulator

IPC
Cache Hit Rate
Num of Instrs
Memory Access
TLB Hit Rate
Power usage
...

Cycle-level GPU Simulations are slow

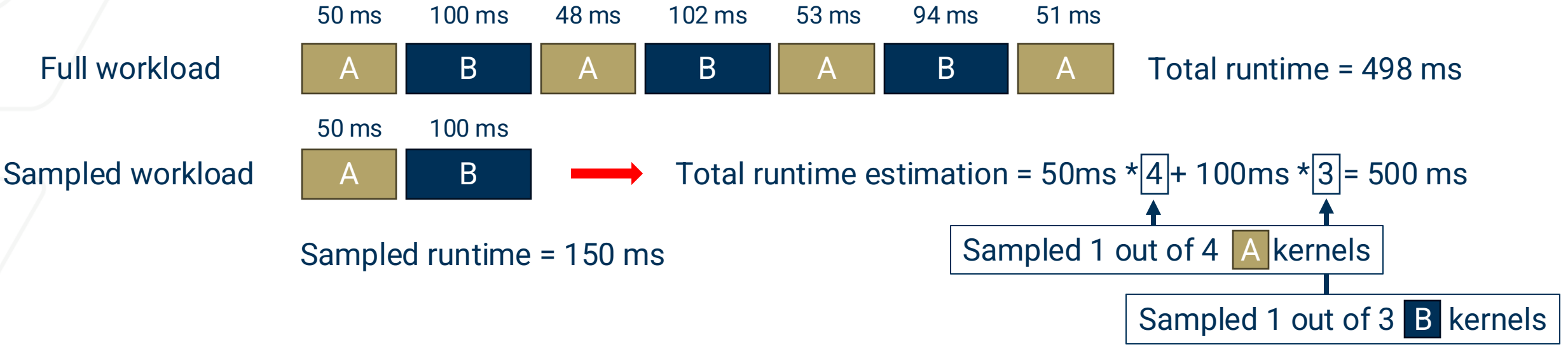
Problem: Cycle-level simulators track the state of GPU at every cycle.

✓ A 1-second workload on a real GPU can take several days on a simulator.

	Real GPU*	Macsim [17]
Simulation Rate (KIPS)	4103750	50.5
GPT-2: Generate 100 tokens	0.925 sec	**20.88 hrs**

Can we reduce the simulation time by leveraging the characteristics of large-scale GPU workloads?

Kernel-level sampling for GPU workloads

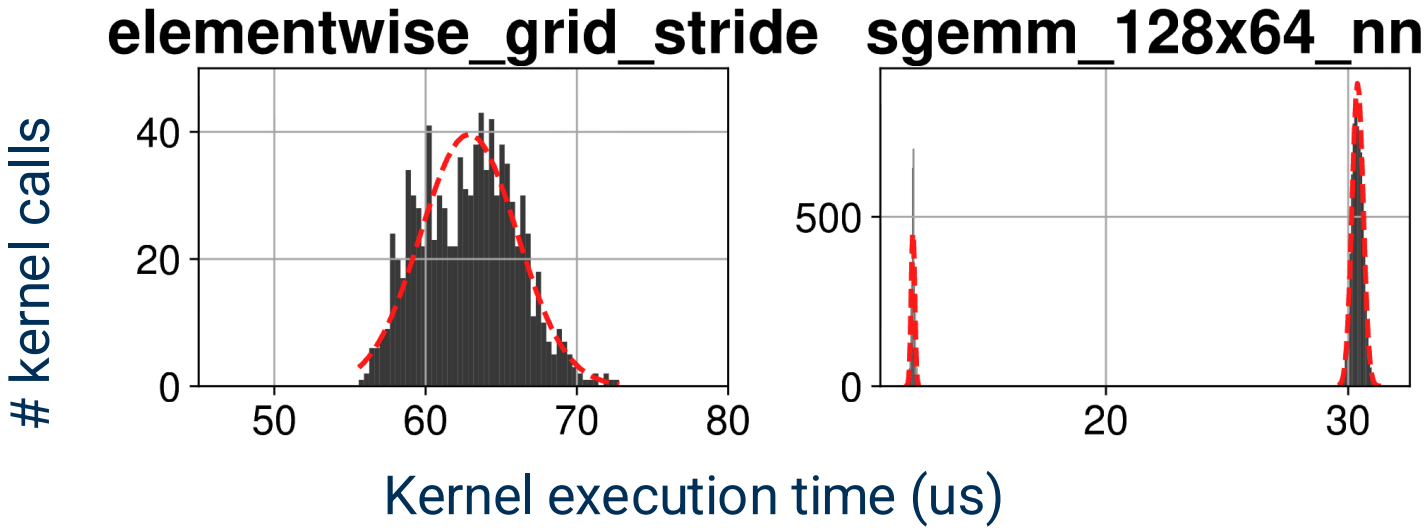


- **Speedup** over full simulation $\approx \frac{498}{150} = 3.32$
- **Sampling error** $= \frac{|500-498|}{498} \times 100(\%) = 0.4\%$

Tradeoff on **speedup** and **accuracy**:
More kernel samples make the sampled simulation **longer** but **accurate.**

Diverse behaviors of GPU kernels in ML workloads

Challenge: Identical GPU kernels show huge variation across invocations at runtime.



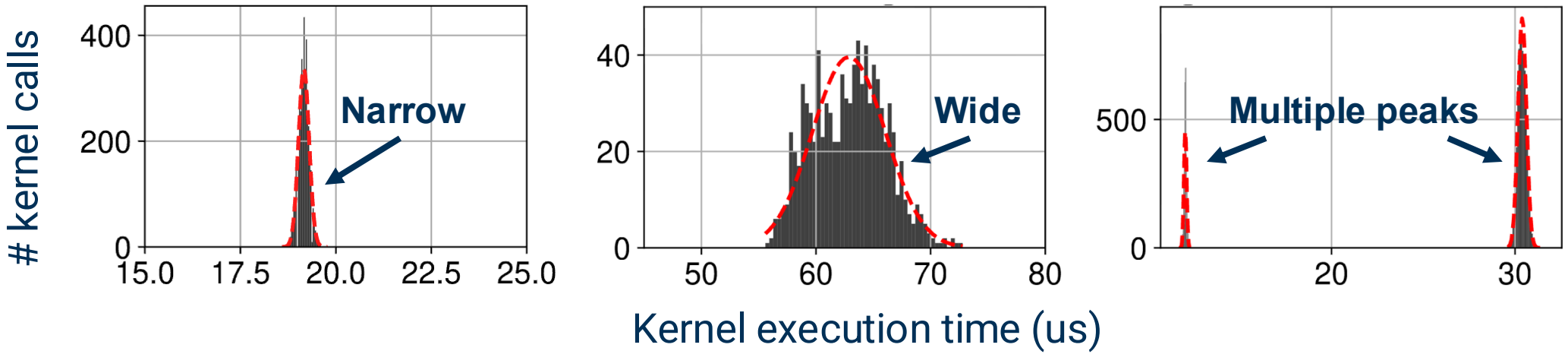
Strategy: **Sample sizes** should be chosen **differently** for each kernel.

- Kernels with **broad distributions** → **More samples** for high accuracy
- Kernels with **distinct peaks** → **Sample less from each peak** for high speedup & accuracy

Leveraging kernel execution time for sampling

Idea: leverage **kernel execution time** as a key signature to sample kernels.

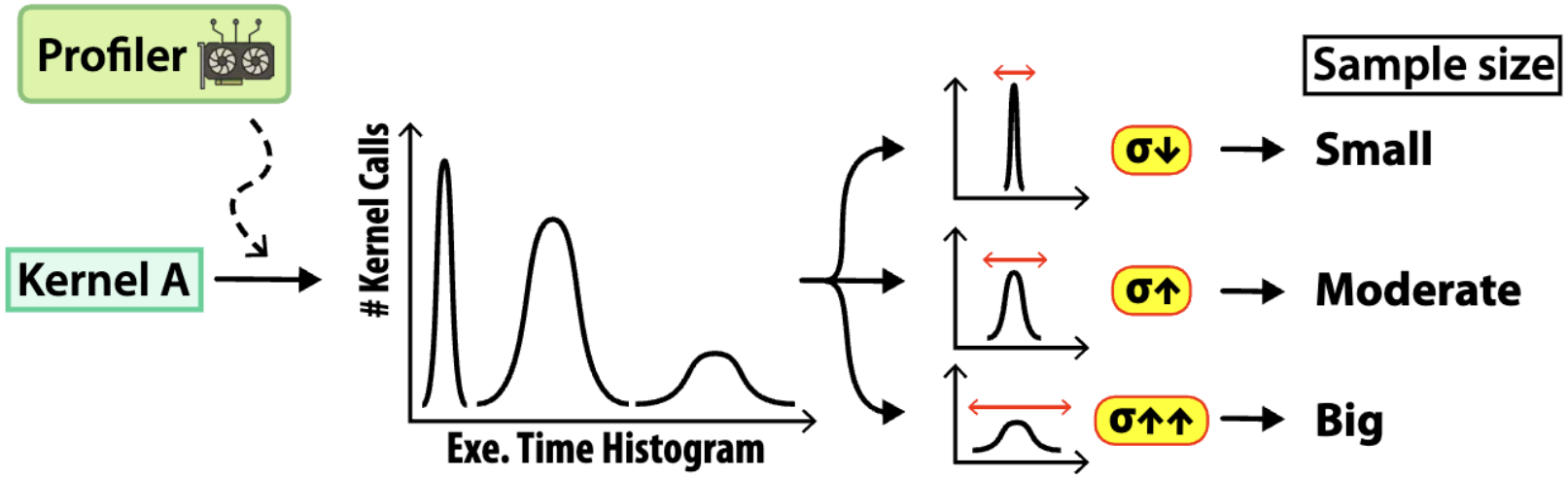
✓ Kernel's execution time reveals much information about runtime characteristics.



- **Narrow:** constant execution time → **less** samples
- **Wide:** variable performance due to uarchitecture-sensitive code → **more** samples
- **Multiple:** kernel used in multiple contexts → **separate** samples from each peak

Statistical approach for kernel-level sampling

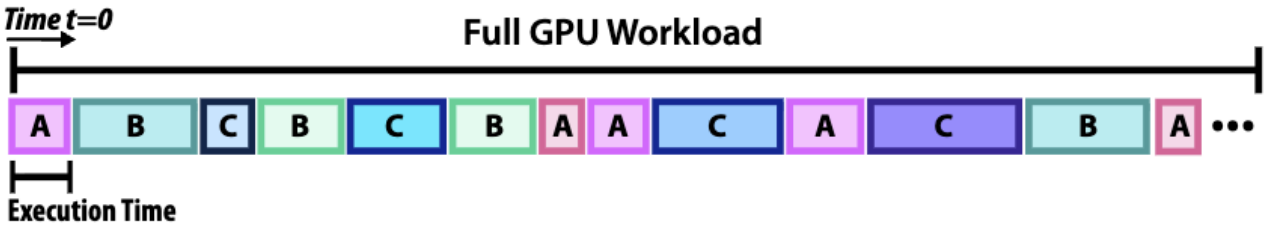
Solution: **Statistical approach** based on kernel profiles.



We can adaptively determine the **sample sizes of kernels** based on their **execution time distributions**.

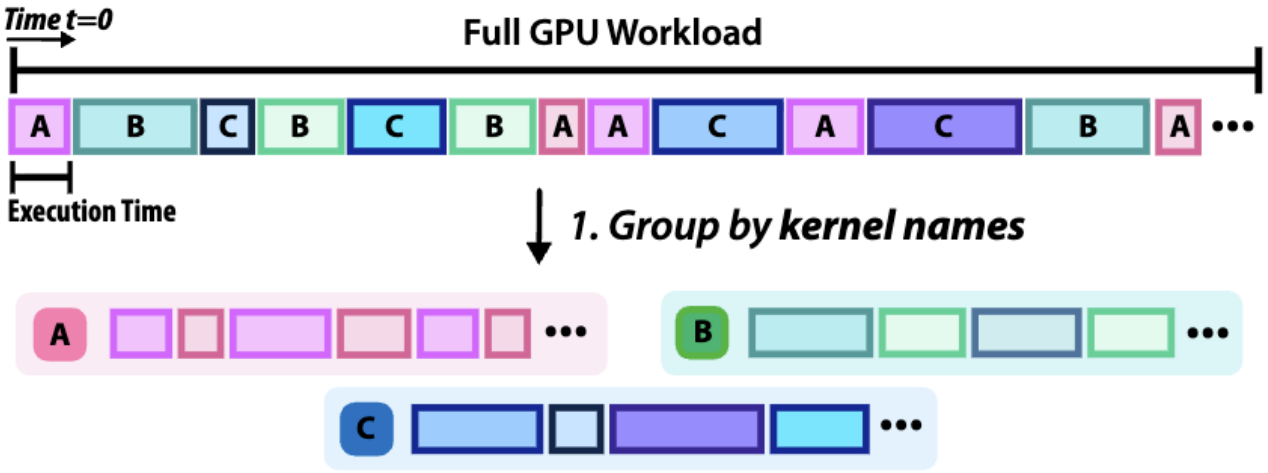
- Sample size based on CoV (Coefficient of variation, σ/μ)

Kernel-level sampling of GPU workloads

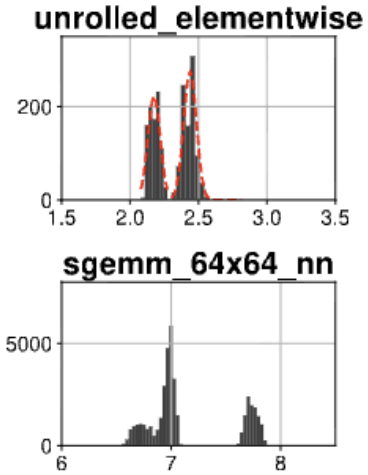


Given a workload with a kernel sequence,
How to select the important kernels?

Kernel-level sampling of GPU workloads

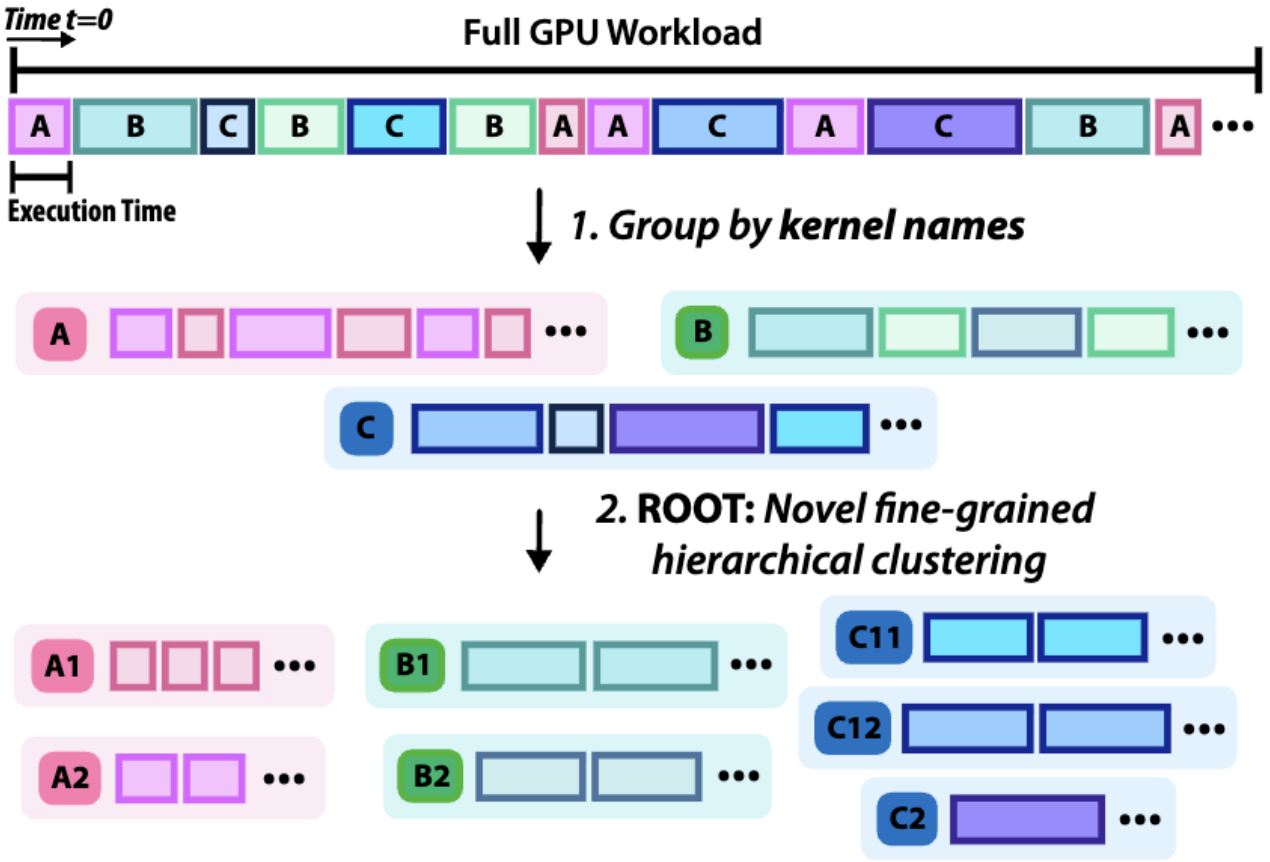


1. Group kernels by kernel names.



Question: How to separate the peaks into separate groups?

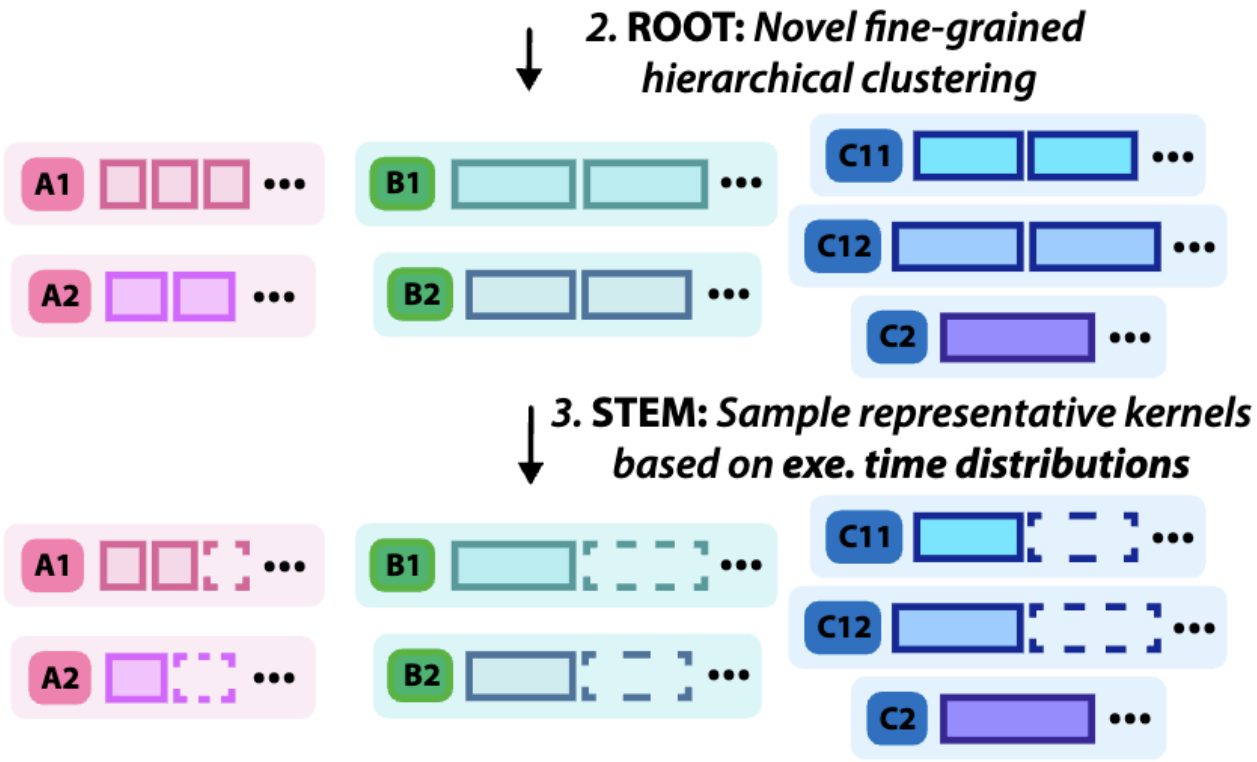
Kernel-level sampling of GPU workloads

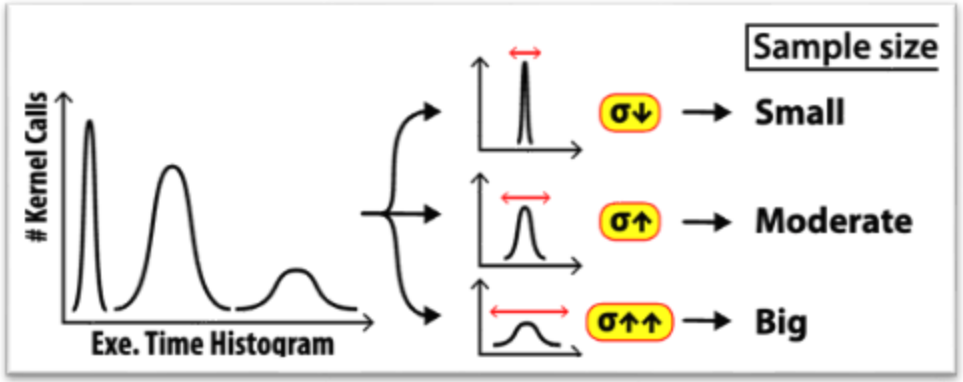


2. ROOT **additionally separates** runtime-heterogenous kernels into different groups.

Georgia Tech

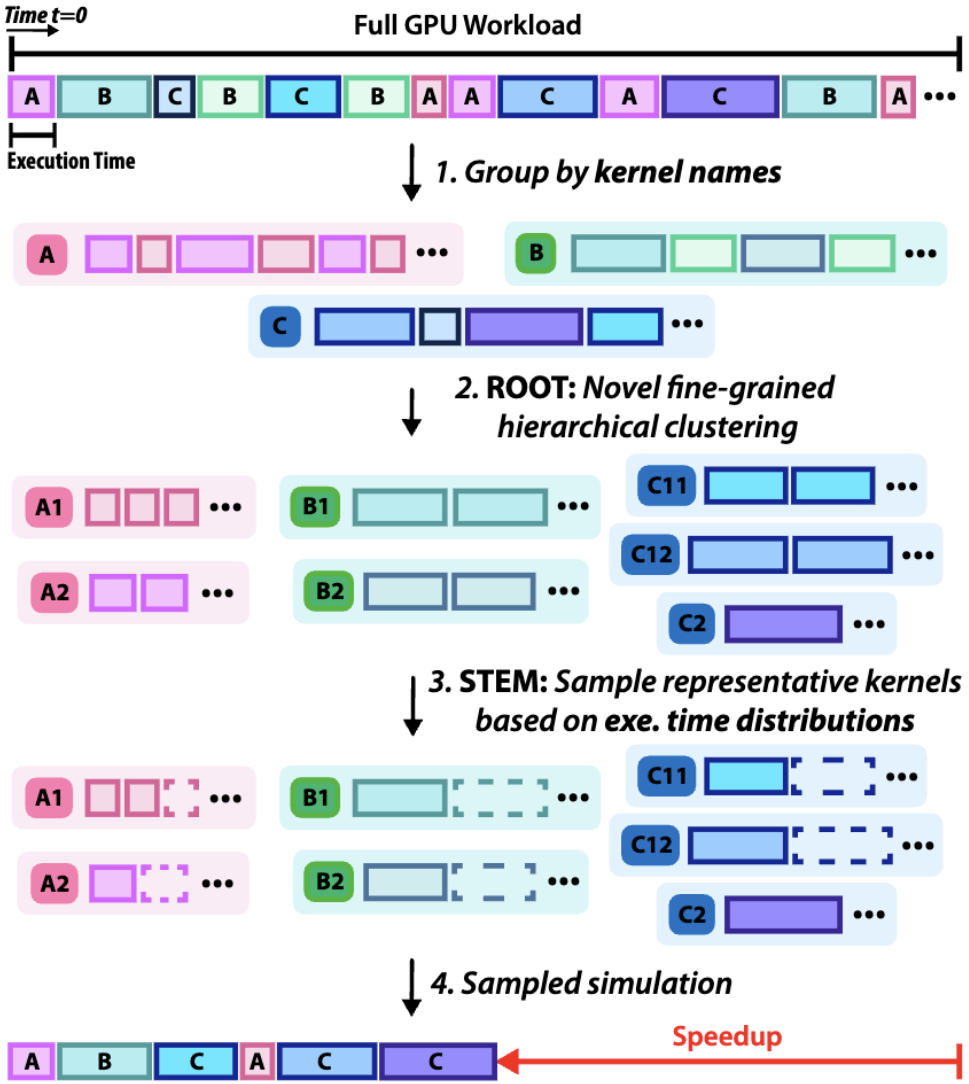
Kernel-level sampling of GPU workloads





3. STEM selects the **optimal sample size** of each group for the best speedup and accuracy.

Kernel-level sampling of GPU workloads



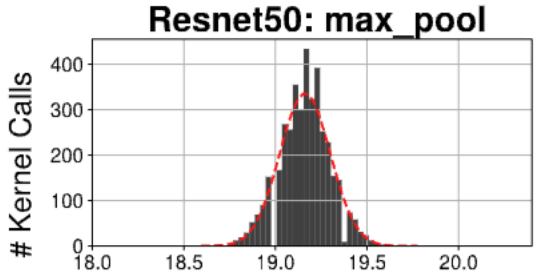
✓ Speedup = A B C B C B A A C A C B A / A B C A C C

✓ Sampling error is minimized.

Georgia Tech

Applying the Central Limit Theorem

Central Limit Theorem (CLT): **The mean of samples** will *always* follow a **Gaussian distribution** as the sample size $m \to \infty$.



m kernel samples

Average kernel execution time follows a Gaussian distribution $\bar{X} \sim N(\mu, \sigma^2/m)$.

We can **analytically calculate** the relationship between the sample size and the error.

- The **minimum number of samples** to ensure the error bound ϵ:

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

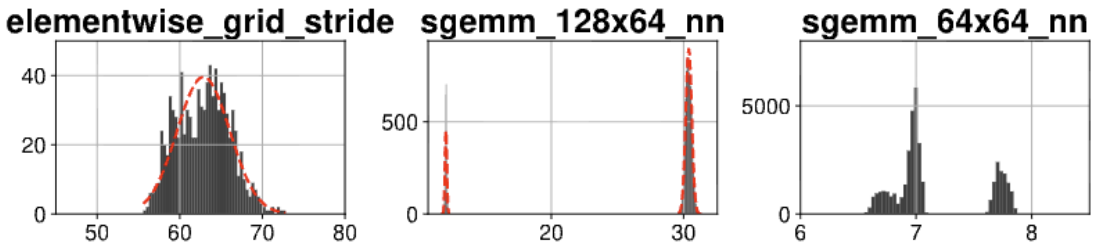
Error bound (e.g. 5%)

Assuming Gaussian distribution

Georgia Tech

STEM: Statistical Error Model for kernel sampling

Minimization problem: minimize **simulation time** while the **error is bounded**.



Sample m1, m2, m3 kernels from each cluster

The **goal**:

- Minimize the total simulation time (= Maximize the speedup)

The **constraints**:

- **Sampling error** should stay within the given error bound ϵ.
- We should sample **at least one sample** from each cluster

✓ The **solution** can be obtained with **KKT** (Karush–Kuhn–Tucker) **conditions**

- More details in the paper!

Georgia Tech

ROOT: Fine-grained hierarchical kernel clustering

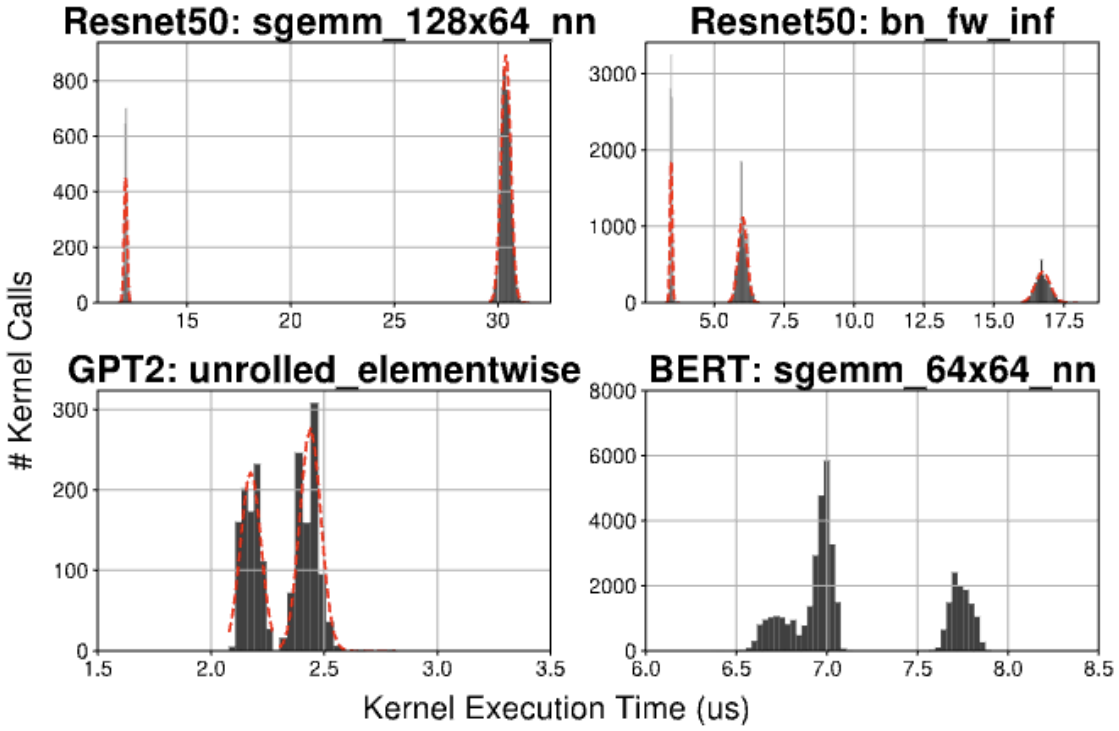
Goal: Distinguish each peak into separate clusters to accurately & effectively sample

Challenges:

- Peaks are often distinct or overlapping
- The number of clusters is unknown, cannot apply traditional clustering algorithms (k-means)

Solution:

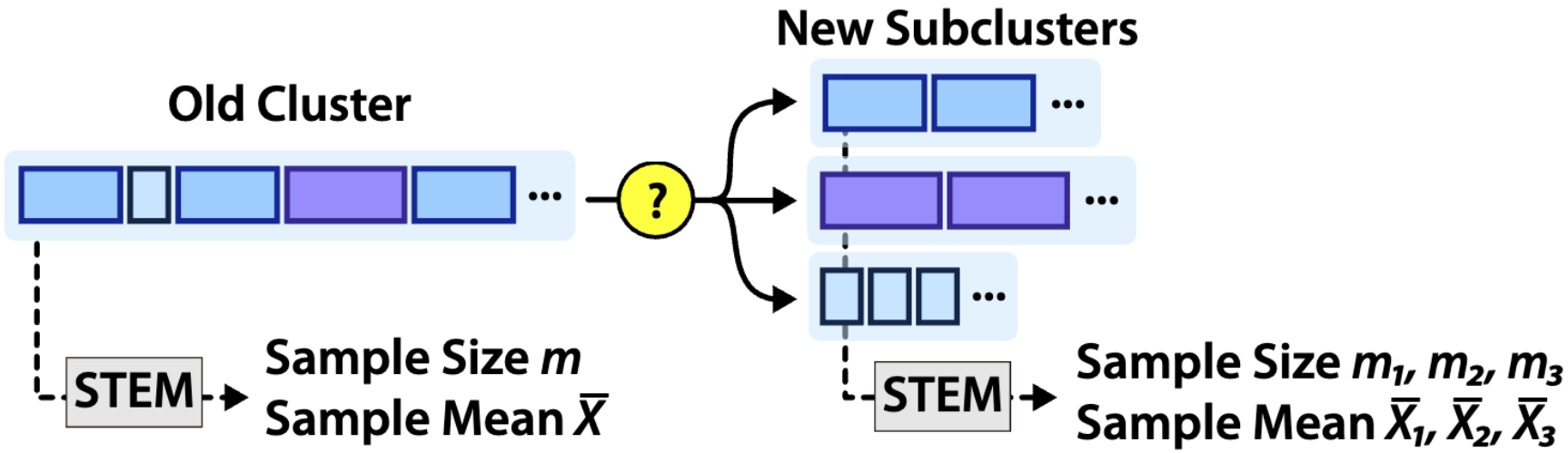
- We propose ROOT, **hierarchical** clustering solution for GPU kernels



Georgia Tech

Deriving the ROOT

ROOT is a recursive algorithm that uses STEM to estimate whether splitting will lead to a better speedup.

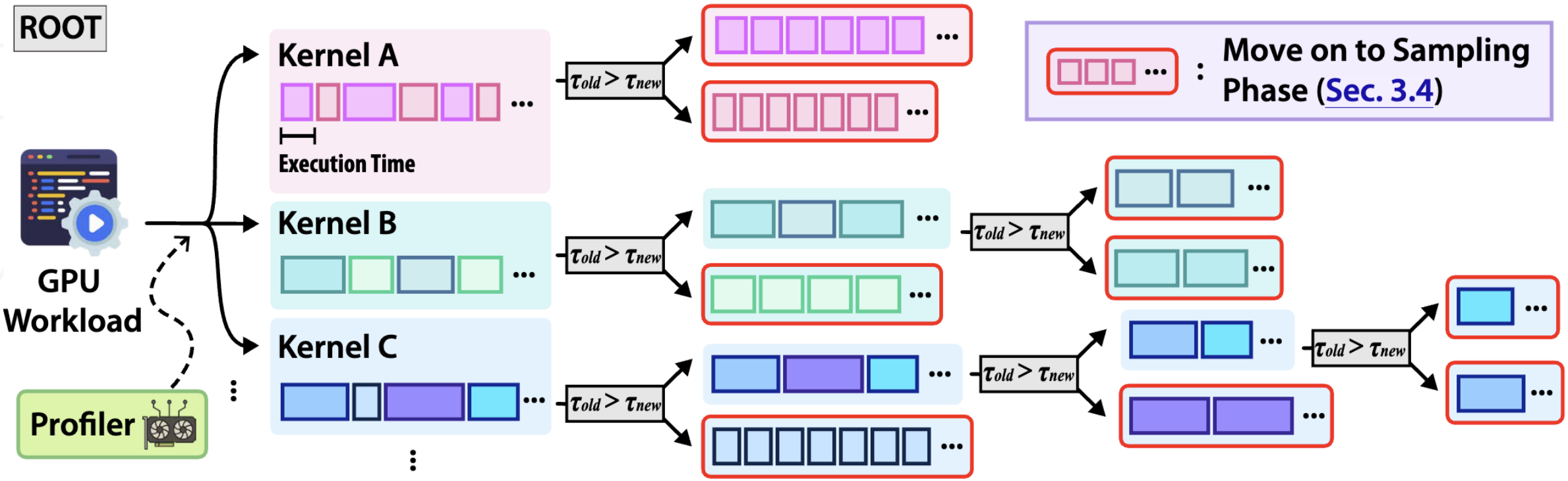


Compare the simulation time: $\tau_{old} = m\bar{X}$

$$\tau_{new} = \sum_i m_i \bar{X}_i$$

→ We can **estimate** how much simulation time we can save from splitting kernels into smaller clusters

Hierarchical clustering of ROOT



Georgia Tech

Evaluation of STEM+ROOT

Evaluated GPU workloads:

- Rodinia* (GPGPU workloads)
- Casio** (ML workloads)
- Huggingface (Large-scale LLM/ML workloads)

Baseline methods:

- Random sampling
- PKA [MICRO ‘20]
- Sieve [ISPASS ‘23]
- Photon [MICRO ‘23]

* S. Che et al., Rodinia: A benchmark suite for heterogeneous computing, 2009
** M. Davies et al., A Journey of a 1,000 Kernels Begins with a Single Step: A Retrospective of Deep Learning on GPUs, ASPLOS 2024

Speedup & Error validation

Methods	Rodinia (GPGPU)		CASIO (ML)		Huggingface (LLM & ML)	
	Speedup (×)	Error (%)	Speedup (×)	Error (%)	Speedup (×)	Error (%)
Random*	7.09	26.67	984.87	28.39	1004.97	2.40
PKA	8.35	34.85	1425.01	29.26	N/A (*Profiling overhead*)**	
Sieve	2.62	6.63	391.09	23.75	N/A (*Profiling overhead*)**	
Photon	2.84	2.71	168.61	9.85	N/A (*BBV process overhead*)**	
STEM	**3.00**	**0.93**	**109.595**	**0.36**	**31719.057**	**0.57**

*Uniform random; We sample 10% and 0.1% of kernels for Rodinia and CASIO, respectively.

**Profiling and BBV processing overhead is estimated at up to 78.68 days.

- STEM & ROOT achieves **comparable speedup** with **significantly lower sampling error** compared to baseline methods.

Speedup & Error validation on cycle-level simulators

μarch Changes	*PKA error (%)	*Sieve error (%)	Photon error (%)	**STEM (ours) error (%)**
Baseline	20.06	24.40	5.96	**2.03**
Cache size $\times 2$	22.66	25.67	5.44	**1.93**
Cache size $\times \frac{1}{2}$	16.65	22.61	5.33	**1.96**
#SM $\times 2$	17.90	28.18	6.49	**2.28**
#SM $\times \frac{1}{2}$	23.68	23.08	5.14	**2.30**

*We observe high error on Rodinia workloads, as we use smaller configurations to run full cycle-level simulation for error measurement.

- STEM & ROOT consistantly achieves **lower sampling error** compared to baseline methods on Macsim.
- **Adaptive sample size** helps preserving the accuracy **even if we change the microarchitecture**

Georgia Tech

More details & evaluation results in our paper!

- Mathematical **modeling and proofs** on statistical sampling
- **Sensitivity analysis** on changing the error bound
- Evaluating STEM on a GPU with **kernel profiles from a different GPU**
- Evaluation on **microarchitecture metrics** (Cache hit rate, # instrs, etc.)
- Workload **profiling overhead** comparison for sampling
- and more.

Conclusion

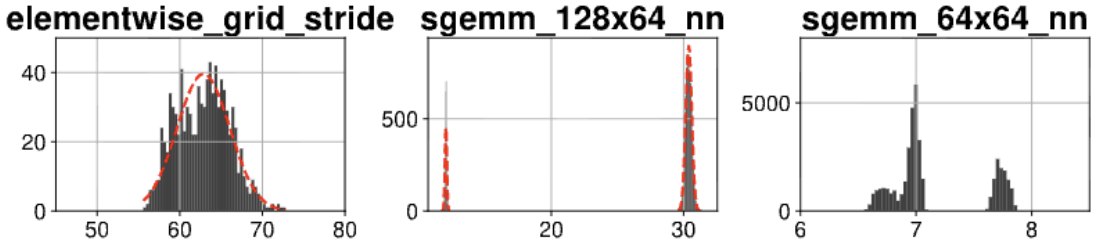
- Our work leverages execution time of kernels to distinguish runtime-heterogeneous kernels, thereby achieving **fast** and **accurate** kernel sampling.
- **STEM**: Statistical Error Model for optimal sample size selection with bounded error
- **ROOT**: Fine-grained hierarchical clustering for kernel-level sampling

- Accurate and scalable kernel sampling solution for large-scale GPU workloads
  - **Small** (<1%) sampling **error** & **high speedup** on GPU workloads.
  - **Scalability on large-scale workloads** with minimal profiling overhead

Backup slides

STEM: Statistical Error Model for kernel sampling

Question: What if we are sampling kernels from multiple clusters at the same time?



Sample m1, m2, m3 kernels from each cluster

Optimization problem:

$$\underset{m_i}{\text{minimize}} \quad \tau = \sum_i m_i \mu_i$$

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

$$\text{and} \quad m_i > 0 \ \text{ for } \ \forall i \in \{0, \ldots, k-1\}.$$

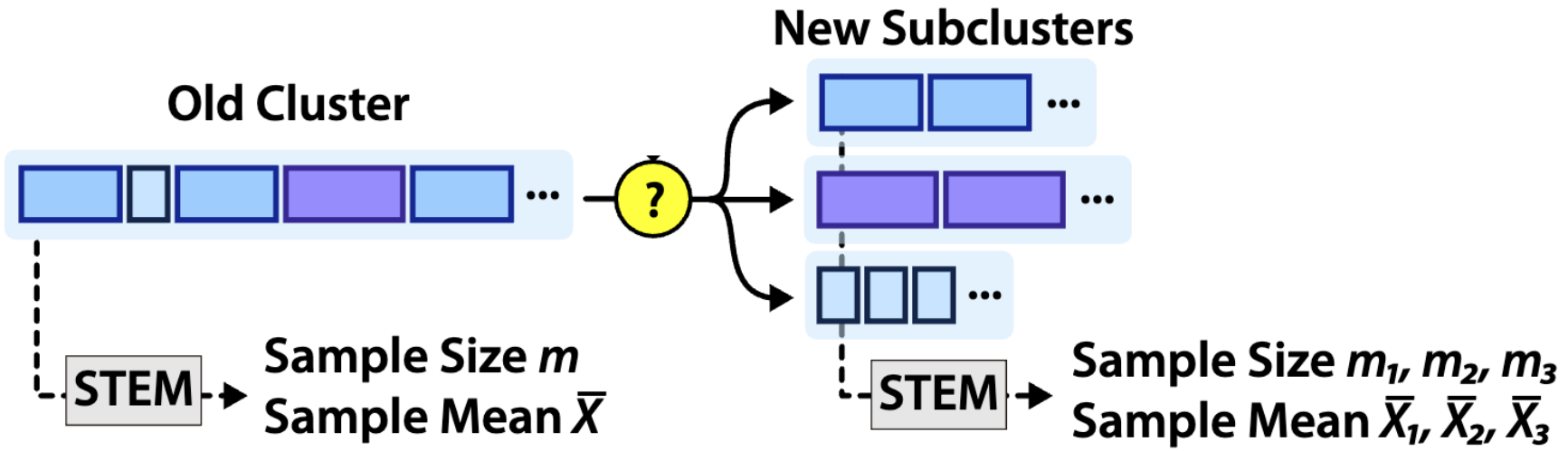
Solution (Using KKT Conditions):

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

$$a_i \equiv \mu_i,\ b_i \equiv N_i^2 \sigma_i^2, \text{ and } c \equiv (\epsilon \textstyle\sum_i N_i \mu_i / z_{1-\alpha/2})^2$$

Georgia Tech

Deriving the ROOT



Compare the speedup:

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

$$\tau_{new} = \sum_i m_i \bar{X}_i = \sum_i \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \cdot \bar{X}_i$$

Baseline kernel sampling methods for GPU workloads

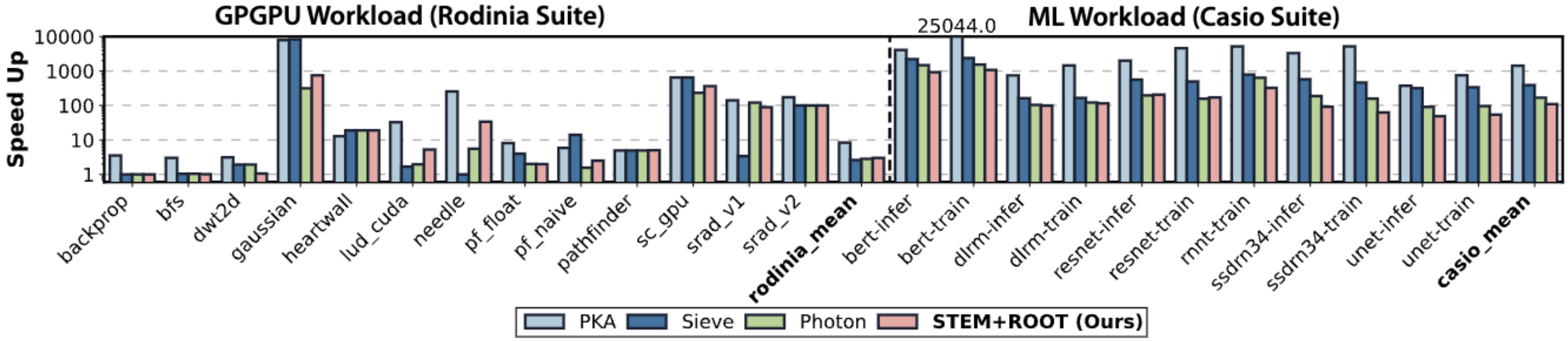
Sampling Methods	PKA [2]	Sieve [24]	Photon [21]	STEM+ROOT (ours)
Kernel signature	12 instr. level metrics	Kernel name & Num. of instrs	GPU Basic Block Vector (BBV)	Kernel name & Exe. time distribution
Clustering	k-means	Hand-tuned, based on CoV (σ/μ)	Find a kernel with similar BBV and #warps (95% threshold)	Fine-grained hierarchical (ROOT)
Kernel sample size	Single per cluster, first chronological	Single per cluster, first chronological		Adaptive sampling with statistically determined sample size (STEM)
Profiling granularity	Instr. count and statistics *per warp*	Instr. count *per warp*	Basic block count *per warp*	Execution time *per kernel*
Scalability for large-scale workloads	Very low	Low	Low	High

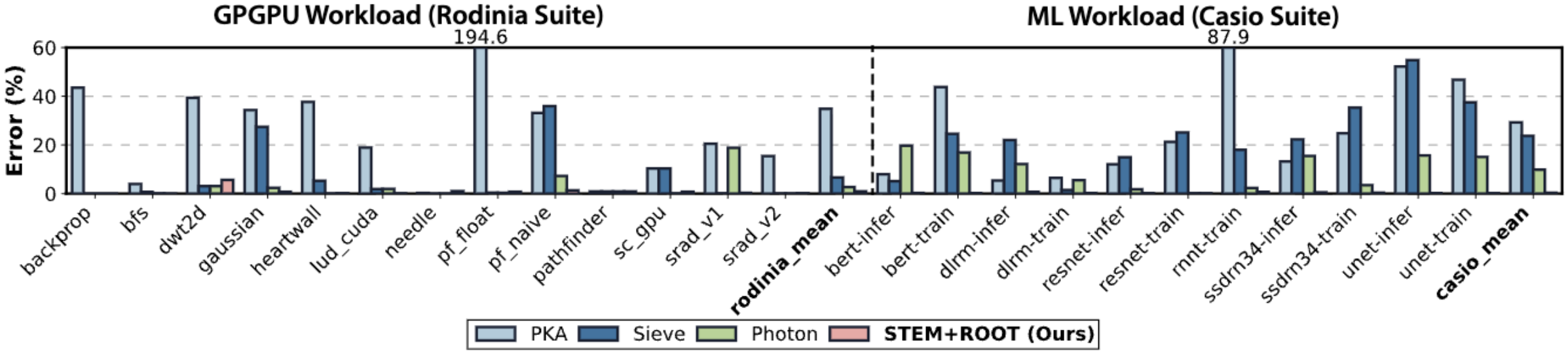
Limitations on previous works:

- PKA, Sieve, and Photon all rely on **static code-level analysis**, which fail to capture runtime heterogeneity of GPU kernels
- PKA and Sieve rely on **heavy profiling** of instr-level metrics
- Photon's BBV comparions between kernels involve $\boldsymbol{O(N^2 d)}$ **computations**.
  - $\mathrm{N} = \text{Number of kernels}, \mathrm{d} = \text{BBV dimension}$

Georgia Tech

Speedup & Error validation





Baseline methods: PKA [Micro ‘20], Sieve [ISPASS ‘23], Photon [MICRO ‘23]

Profiling overhead

Sampling methods	Profiler used, metrics collected	Rodinia (GPGPU)	CASIO (ML)	Huggingface (LLM & ML)
PKA [2]	NCU, collecting 12 metrics	35.57×	3704.23×	N/A
Sieve [24]	NVBit, collecting num. of instrs	94.14×	293.58×	N/A
Photon [21]	NVBit, collecting & processing BBVs	12.81×	38.58×	N/A
STEM (ours)	NSYS, collecting kernel exe. time	**1.54×**	**5.53×**	**1.33×**

Using **execution time** as a key parameter gives a huge improvement in **scalability**

Georgia Tech