2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

GPU Scale-Model Simulation

Hossein SeyyedAghaei, Mahmood Naderan-Tahan, Lieven Eeckhout
Ghent University, Belgium

***Abstract*—The continuously increasing GPU system scale and compute capabilities, i.e., increasing number of streaming multiprocessors (SMs), caches, on-chip and off-chip memory bandwidth, pose a major challenge for performance evaluation methodologies. Architectural simulation is time-consuming and resource-intensive, and because of simulator and/or simulation host infrastructure limitations, it might not even be possible to simulate large-scale systems. Scale-model simulation is a recently proposed performance prediction methodology to predict large-scale system performance based on (much smaller) scale models. Prior work in scale-model simulation for general-purpose multi-core CPUs and specialized graph analytics accelerators, unfortunately, cannot be readily applied to GPUs because different GPU applications exhibit vastly different scaling behavior with system size, thereby breaking the one-size-fits-all regression models deployed in prior work.**

This paper proposes a GPU scale-model simulation methodology that leverages performance measurements of two scale models alongside a miss rate curve to predict GPU target system performance. A key asset of GPU scale-model simulation is that it does not require access to a simulation model of the target system, unlike prior work in simulation acceleration. Our experimental evaluation demonstrates the accuracy of GPU scale-model simulation for both strong-scaling and weak-scaling workload scenarios. Under strong scaling, the performance of a 128-SM target system is predicted within 4% error on average, and at most 17%, using 8-SM and 16-SM scale models. Under weak scaling, the performance of a 128-SM target system is estimated with an average error of 1.7%, and at most 4.5%, while yielding a 9.3× simulation time speedup. We furthermore demonstrate how scale-model simulation predicts multi-chiplet GPU performance with an average error of 2.5% (and at most 4.3%). Alternate solutions are substantially less accurate.

I. Introduction

Graphics Processing Units (GPUs) are widely deployed hardware accelerators. Not only are GPUs used for traditional graphics workloads, general programming interfaces such as CUDA and OpenCL have paved the way for using GPUs to accelerate general-purpose computing. So-called GPU-compute workloads have consequently emerged from a wide variety of application domains, ranging from high-performance computing (HPC) [4, 17, 43], graph analytics [5, 61], Artificial Intelligence (AI) and Machine Learning (ML) [41, 50, 51]. Today, many application domains critically depend on GPU performance and its scaling.

GPU compute capabilities have exponentially increased over the past few years. We have seen a dramatic increase in the number of compute cores, or Streaming Multiprocessors (SMs) in Nvidia terminology. To feed the increasing number of SMs with data at a fast enough pace, on-chip caches have increased as well as off-chip memory bandwidth. For example, whereas Nvidia's Fermi GPU first released in 2010 featured 16 SMs, a 768 KB last-level cache (LLC) and 192 GB/s memory bandwidth, Nvidia's latest Hopper H100 GPU released in 2022 features 144 SMs, a 60 MB LLC and 3 TB/s memory bandwidth [1]. To scale GPU performance in the wake of Moore's Law slowing down, multiple GPU chips are being integrated into multi-GPU systems with a high-bandwidth interconnection network, e.g., Nvidia's NVLink and NVSwitch [24, 26, 33], or even using interposer-based interconnects [7].

Increased GPU system scale and compute capabilities pose a major challenge to performance analysis and evaluation methodologies. Although significant progress has been made in analytical performance modeling for GPUs, see for example [29, 30, 31, 42, 60], architectural simulation is and remains the most widely used performance evaluation methodology at early stages of the design cycle [9, 39, 56, 58]. Simulating increasingly large system sizes with realistic workloads in reasonable time budgets is challenging, up to the point that it becomes impractical or even infeasible to simulate some of the high-end next-generation system configurations at cycle-level detail [58]. Not only is architecture simulation extremely time-consuming, it also takes up considerable resources and hence comes at a considerable cost, i.e., server farms need to procured, maintained, and powered on for long periods of time to support the many simulations that need to be conducted during architecture exploration. Moreover, because of simulator and/or simulation host limitations, it might not even be possible to simulate some of the largest system configurations.

Architectural simulation acceleration is obviously not a new topic. The most widely used techniques to speed up architecture simulation are to sample a workload's execution [8, 15, 16, 22, 32, 36, 40, 52, 53, 54, 57, 62, 63], consider reduced inputs [21, 34, 64], or employ FPGA-accelerated simulation [14, 18, 19, 37, 59]. *All prior work in simulation acceleration implicitly assumes access to a simulation model of the target system*, which might not be available, and if available, might have taken considerable time and effort to develop, and be extremely slow and resource-intensive to use.

Scale-model simulation was recently proposed as an alternative solution to the architectural simulation challenge [45]. *Scale-model simulation's key benefit is that it does not require access to the target system's simulation model; furthermore, scale-model simulation is fast and does not involve many hours of simulation on costly server farms.* The idea of scale-model simulation is (1) to construct a scaled-down version of the (much) larger target system, called the *scale model*; (2) to simulate the scale model to obtain its performance profile;

and (3) to extrapolate the scale model's performance profile to predict performance of the target system. By doing so, scale-model simulation does not require simulating the target system, saving considerable time and effort, and in some cases even circumventing the problem that the simulation model for the large target system is not available.

This paper proposes a scale-model simulation methodology for GPUs, in contrast to prior work which focused on scale models for general-purpose multi-core CPUs [45, 46] and specialized graph accelerators [25]. What makes GPU scale-model simulation challenging is that different workloads scale differently with system size: while some workloads scale linearly, others scale sub-linearly, or even super-linearly. As a result, the one-size-fits-all regression approach from prior work leads to inaccurate performance prediction. Instead, the GPU scale-model simulation methodology proposed in this work assumes two inputs: (1) performance profiles for at least two scale models, and (2) miss curves that quantify the number of LLC misses per thousand instructions as a function of cache size. Using the performance profiles of the scale models, alongside the miss curves allows for accurately predicting GPU target system performance.

Our experimental evaluation covering a wide range of GPU-compute workloads clearly demonstrates the accuracy and effectiveness of GPU scale-down simulation for both weak-scaling and strong-scaling workload scenarios. Under strong scaling, using scale models with 8 and 16 SMs (with commensurate LLC capacity, NoC and memory bandwidth), GPU scale-down simulation predicts performance for 64- and 128-SM target configurations with an average error of 3.5% (13% max error) and 4% (17% max error), respectively. In comparison, a naive approach that assumes that performance increases proportionally with system size leads to an average error of 22% (and up to 113%); similarly, linear and power-law regression yields average errors of 17% and 12% (and up to 68% and 55%), respectively. Under weak scaling, GPU scale-model simulation predicts 128-SM target system performance with an average error of 1.7% (maximum 4.5% error) while at the same time yielding a 9.3× simulation speedup. Furthermore, we demonstrate how scale-model simulation can accurately predict multi-chiplet GPU performance: using scale models with 4 and 8 chiplets, scale-model simulation predicts 16-chiplet performance with an average error of 2.5% (and at most 4.5%). The overall conclusion is that scale-model simulation is a novel and useful complement to the GPU architect's and performance analyst's toolbox.

II. Prior Work in Scale-Model Simulation

Scale models are widely used in various engineering disciplines, including civil engineering, mechanical engineering, construction, architecture, etc. Miniatures are often used scale models to study (much) larger target systems. A critical property of a scale model hence is that its key properties and characteristics are the same (or at least similar) as in the target system. For a scale model to be useful, this needs to be true along a number of important dimensions, but not necessarily all. As such, one can use the scale models to study the behavior of a large target system using a much smaller and better manageable scale model.

As mentioned in the introduction, scale models were just recently introduced in the field of computer architecture for general-purpose multi-core CPUs [45, 46] and specialized graph accelerators [25]. These scale models are not exact miniatures of the target system though, but at least some of the dimensions are scaled down such that the scale models are predictive for the larger target system. In particular, prior work found that the shared hardware resources are best proportionally scaled in the scale models compared to the target system. This means that when scaling the number of cores in a multi-core system for example, shared cache capacity, the on-chip interconnection network bandwidth, and the off-chip memory bandwidth should be scaled proportionally. For example, if the scale model features a factor F fewer cores than the target system, the shared LLC, the interconnection network, and memory bandwidth should be scaled down by a factor of F as well, so that the performance impact of the shared resources is (somewhat) similar in the scale model compared to the target system, relatively speaking. A component that does not change when scaling system size, e.g., a core's internal organization, is kept unchanged in the scale models versus the target system.

Because a scale model does not precisely capture the interference in the shared resources as observed in the target system, extrapolation is needed to further fine-tune and improve the scale model's predictive power. Indeed, interactions in shared resources could be beneficial (i.e., positive interference) or detrimental (i.e., negative interference) to performance, which may manifest themselves only at scale in the target system. Prior work [45, 46] used machine learning and regression techniques for scale-model extrapolation. In particular, a machine learning model is first trained using a set of benchmarks, and a regression model is then fit to the training data to yield the final extrapolation model. For the workload of interest, a number of scale models are simulated, and their performance figures are then used as input to the extrapolation model to predict target system performance.

The methodology proposed in prior work faces at least three limitations. First, prior work focused on general-purpose CPUs [46] and specialized graph accelerators [25], and can not directly be applied to GPUs. Second, it relies on a training phase that involves the simulation of various scale models using a broad set of training benchmarks. Even though this is a one-time cost, it can be time-consuming and hence impractical. Moreover, the benchmarks used during training may not necessarily be representative of the workload of interest for which we want to predict target system performance. As a result, scale-model extrapolation for a workload of interest that vastly differs from the training benchmarks may lead to inaccurate predictions. Third, prior work uses the same regression model for all workloads of interest — logarithmic regression was found to be more accurate than linear and power-law regression for general-purpose multi-core CPUs running multi-program workloads [46]. The problem is that

not all workloads follow a similar scaling trend, breaking the one-size-fits-all regression approach in prior work. In fact, we find that different GPU workloads exhibit vastly different scaling behavior, i.e., some workloads scale linearly while others scale sub-linearly or even super-linearly with system size. As a result, a one-size-fits-all regression approach leads to inaccurate performance predictions, as we demonstrate in this work.

In conclusion, we need a GPU scale-model simulation methodology that (1) does not rely on an elaborate training phase — to mitigate the risk of being non-representative for a workload of interest — and (2) does not rely on a general regression model — to mitigate the risk of being unable to accurately capture the specific trend for the workload of interest. In contrast, *we need a methodology that builds a per-workload regression model that extrapolates GPU performance based solely on the workload's scale-model performance profile from which performance is then extrapolated to the larger scale target system.* The GPU scale-down simulation methodology proposed in this work achieves exactly this.

III. Problem Statement

Before presenting our GPU scale-model simulation methodology in detail, we first elaborate on the exact problem statement that we address in this work. Recall that the end goal of scale-model simulation is to predict performance of a large target system based on scale-model simulation results. The largest target system considered in this work is a 128-SM GPU with a 34 MB LLC, 2.7 TB/s on-chip interconnection bisection bandwidth, and 2.3 TB/s memory bandwidth interface. In addition, we also consider target systems with 32 and 64 SMs, and commensurate LLC capacity, on-chip interconnection bandwidth, and memory bandwidth. The scale models feature only 8 and 16 SMs. We further consider both strong-scaling and weak-scaling workload scenarios. Strong scaling assumes that the workload is fixed and independent of system size. Hence, one can expect the execution time to reduce with increasing system size. In contrast, under weak scaling, the workload scales with system size: the larger the system, the larger the problem size the workload runs.

The first challenge for scale-model simulation is how to construct scale models such that they enable accurate performance predictions for the larger target system(s). We follow prior work [46] and scale the GPU's shared resources proportionally with system size. For example, a scale model with 16 SMs that is supposed to be a miniature of a target system with 128 SMs features an LLC with 1/8th the size of the target system's LLC; an on-chip interconnection network with 1/8th the bisection bandwidth of the target system's on-chip network; and an off-chip memory interface with 1/8th the bandwidth of the target system's memory bandwidth. See Table I for how we derive the scale models with 8 and 16 SMs from the largest target system with 128 SMs through proportional resource scaling.

The second challenge is to predict target-system performance based on the simulation results obtained for the scale models, i.e., how to extrapolate scale-model performance results to the target system, for example predict performance for the 128-SM target system using the performance numbers obtained for the 8-SM and 16-SM scale models. Under a strong-scaling workload scenario, i.e., the workload remains constant while scaling system size, this challenge is severely complicated by the fact that performance scales differently with system size for different types of workloads. This is illustrated in Figure 1 which shows system performance (i.e., instructions per cycle or IPC) as a function of system size for the 8-SM and 16-SM scale models, and the 32-SM, 64-SM, and 128-SM target systems, for three benchmarks. These three benchmarks exhibit the three fundamentally different scaling behaviors as a function of system size. (We show only three benchmarks due to space constraints; other benchmarks follow similar trends.) While performance scales linearly for pf (right-most graph), we observe sub-linear scaling for bfs (middle graph), and super-linear scaling for dct (left-most graph). The fact that different workloads scale differently calls for an approach that differs from prior work [46] which assumed a one-size-fits-all regression approach (namely, logarithmic regression). It is clear from Figure 1 that different workloads need a different regression strategy: the linear regression model illustrates how far off real benchmark scaling deviates from linear scaling.

A weak-scaling workload scenario, i.e., the workload scales with system size, poses fewer challenges for scale-model extrapolation than strong scaling. The reason is that under weak scaling the workload's input, and thus its working set, scales proportionally with system size. While we observe super-linear, sub-linear and linear scaling behavior under strong scaling, as illustrated in Figure 1, we only notice linear and sub-linear scaling under weak scaling. The fundamental reason is that, under strong scaling, super-linear scaling behavior occurs when the application's memory footprint suddenly fits within the on-chip caches as we increase system size from the scale model(s) to the target system. This effect is absent under weak scaling assuming that the memory footprint scales linearly with system size (and its respective cache size); in other words, the memory footprint remains constant in relative terms, hence no super-linear scaling behavior occurs.

IV. Scaling Behavior

As mentioned in the previous section, workloads scale either linearly, sub-linearly, or super-linearly. We now explain the execution characteristics that result in the different scaling behaviors. We find that scaling behavior strongly (inversely) correlates with last-level cache behavior, see Figures 1 and 2. In particular, the number of LLC misses per thousand instructions (MPKI) is a strong indicator for how performance is affected by the shared resources, including the NoC, the LLC and main memory. Indeed, MPKI is affected by (1) the number of accesses to the LLC, i.e., indicating the number of transfers over the NoC, (2) the LLC miss rate, i.e., indicating the impact of LLC capacity, and (3) the number of memory accesses, i.e., impacting main memory bandwidth. While all scaling behaviors may occur under strong-scaling workload scenarios,

TABLE I: Scale models are derived from the target systems through proportional resource scaling: LLC capacity in MB (each LLC slice is 64-way set-associative with 64 sets and 128 B cachelines); NoC bisection bandwidth in GB/s; main memory bandwidth in GB/s: number of memory controllers (MCs) and bandwidth per MC.

	#SMs	LLC size and configuration	NoC bisection BW	Main memory bandwidth
Target systems	128	34 MB, 32 slices	2,696 GB/s	2,320 GB/s, 16 MCs, 145 GB/s per MC
	64	17 MB, 16 slices	1,348 GB/s	1,160 GB/s, 8 MCs, 145 GB/s per MC
	32	8.5 MB, 8 slices	674 GB/s	580 GB/s, 4 MCs, 145 GB/s per MC
Scale models	16	4.25 MB, 4 slices	358 GB/s	337 GB/s, 2 MCs, 145 GB/s per MC
	8	2.125 MB, 2 slices	168.5 GB/s	145 GB/s, 1 MCs, 145 GB/s per MC

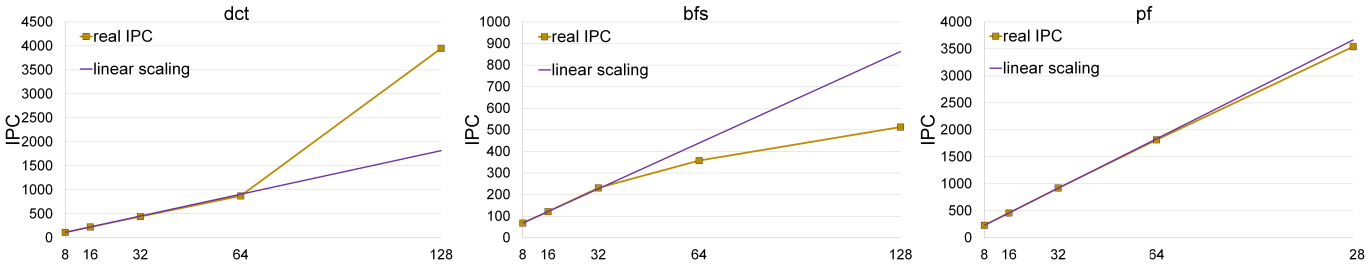


Fig. 1: Performance as a function of system size under a strong-scaling workload scenario. *Different benchmarks scale differently with system size: super-linear scaling (*dct*, left), sub-linear scaling (*bfs*, middle), and linear scaling (*pf*, right).*

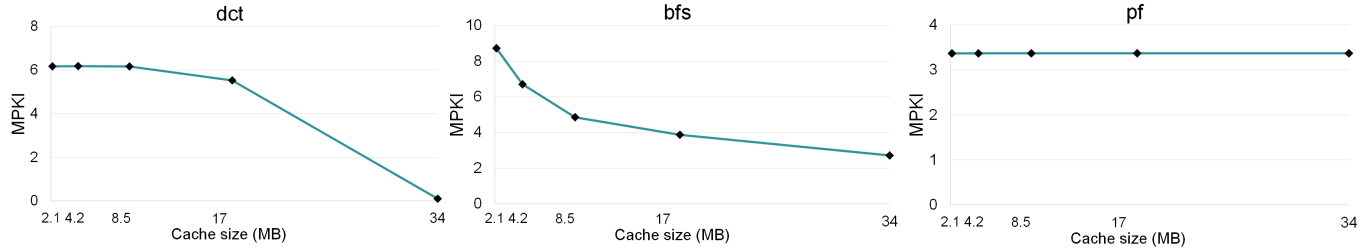


Fig. 2: Miss rate curves: misses per thousand instructions (MPKI) as a function of last-level cache (LLC) capacity, assuming strong scaling. *Different benchmarks exhibit differing miss rate curves: sharp decrease (*dct*, left), gradual decrease (*bfs*, middle), and constant (*pf*, right).*

only linear and sub-linear scaling occurs under weak scaling. These observations underpin the design choices we made when devising our proposed GPU scale-model simulation method. We now discuss the three scaling behaviors: linear, super-linear and sub-linear scaling, and how it is affected by and correlates with LLC MPKI.

1) Linear Scaling: A compute-intensive workload typically exhibits linear scaling behavior, i.e., performance scales linearly with system size. Indeed, performance of a compute-intensive workload mostly depends on the available compute resources (i.e., the number of SMs), while cache capacity, on-chip network bandwidth, as well as off-chip memory bandwidth have only minimal impact on performance, simply because the workload does not stress those parts of the system. As a result, as we scale system size, the performance of a compute-intensive workload is expected to scale linearly with performance.

Counter-intuitively perhaps, some memory-intensive workloads also scale linearly with system size. In particular, when the footprint of the workload largely exceeds the available cache capacity across the entire system scale, i.e., from the smallest scale model to the largest target system of interest, the footprint will never fit inside the available on-chip caches. Hence, a (significant) fraction of L1 cache misses will need to traverse the on-chip interconnection network to access the last-level cache (LLC), and may need to go to off-chip memory in case they miss in the LLC as well. The extent to which performance depends on the memory subsystem hinges on the memory access pattern by the workload as well as on the available shared resources, i.e., on-chip cache capacity, NoC bisection bandwidth, and off-chip memory bandwidth. In any case, the shared resources scale proportionally with system size — this was done deliberately so that the scale models are predictive of the target systems. As a result, the relative impact of the memory accesses remains constant irrespective of system size. Overall performance hence scales linearly with system size.

A benchmark that illustrates the linear scaling behavior is pf, see Figure 1 (rightmost graph). This benchmark features a flat miss rate curve, see Figure 2 (rightmost graph) which reports the number of LLC misses per thousand instructions (MPKI) as a function of system size. The flat miss rate curve indicates that cache capacity has no measurable impact on the number of misses. The reason is that pf's footprint is quite large (404 MB,

see Table II). High data reuse leads to a relatively high and constant MPKI, ultimately leading to linear scaling behavior.

2) Super-Linear Scaling: As mentioned before, super-linear scaling may occur under a strong-scaling workload scenario. In particular, a memory-intensive workload with a working set that is somewhat comparable to the cache capacity of the target system may witness super-linear scaling as system size scales, especially when there is substantial data reuse. While the working set may be too big to fit in the on-chip cache for the scale models, it might fit the cache of the target system. As a result, as soon as the working set of an application with high data reuse fits inside the on-chip cache, a significant performance boost can be observed. This is illustrated in Figure 2 (leftmost graph) for the dct benchmark: the miss rate curve shows a dramatic drop when on-chip LLC capacity changes from 17 MB to 34 MB. This can be explained by considering dct's working set size of 33 MB, see Table II. This is an example of the cliff behavior previously reported for CPU workloads [11, 12, 23] where miss rate curves experience a sudden drop as the working set fits in the available cache capacity. The cliff in the miss rate curve leads to a commensurate increase in performance, as illustrated in Figure 1 (leftmost graph). This explains the super-linear scaling behavior.

Note that the size of the working set being smaller than the on-chip LLC is a necessary condition but not a sufficient condition for super-linear scaling behavior to occur. Indeed, the working set needs to exhibit sufficient data reuse as well. Take for instance the ht benchmark as an example. Its footprint equals 12.5 MB, see Table II, which is smaller than the 17 MB and 34 MB LLC for the 64-SM and 128-SM target systems. However, because there is almost zero data reuse, no super-linear scaling behavior is observed. Instead, we observe linear scaling behavior, see the rightmost column in Table II for the benchmarks' classification.

Note that predicting super-linear scaling behavior is difficult, if not impossible, by simply looking at the performance profile of the scale models. Indeed, the performance across the two scale models for dct increases at a particular rate, i.e., there is a 1.98× performance improvement comparing the largest (16-SM) scale model versus the smallest (8-SM) scale model. The performance increase observed when transitioning across the cliff is much higher, i.e., performance increases by more than 4× when comparing the 128-SM target system against the 64-SM scale model. This sharp increase is impossible to predict solely based on the scale-model performance profile. The miss rate curve on the other hand provides a hint that one may expect a significant performance boost when transitioning across the cliff.

3) Sub-Linear Scaling: There are at least two mechanisms that could lead to sub-linear performance scaling with system size. The first mechanism relates to workload-architecture imbalance under which the number of execution stalls increases with system size. For example, if the workload is too small, i.e., there are not enough CTAs to keep the SMs busy all the time, the number of stall cycles is likely to increase with system size. In other words, the SMs can no longer fully hide the execution latency of warps by executing other warps, simply because there are not enough CTAs to select warps from to execute.

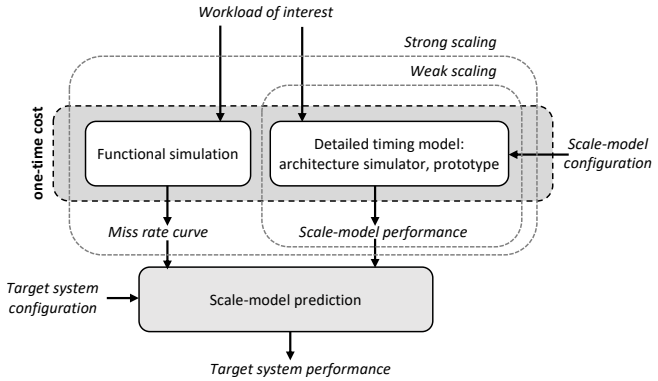


Fig. 3: GPU scale-model simulation workflow under strong and weak scaling. *The one-time cost involves collecting the miss rate curve and scale-model performance numbers, after which performance is predicted for the target system of interest. (The miss rate curve is only needed under a strong-scaling workload scenario, not under weak scaling.)*

The second mechanism that could lead to sub-linear scaling is shared data congestion. CTAs executing on different SMs accessing the same shared data around the same time, leads to camping in front of the LLC slices [65, 66]. Indeed, the LLC is a shared cache, i.e., all SMs can access all LLC slices, and a cache line is stored in only one of the LLC slices — the LLC slice where it is stored is determined by its address. Because of the data sharing, more SMs will access the shared data around the same time as we increase system size. This exacerbates the camping problem, which could lead to sub-linear scaling.

The bfs benchmark (middle graph in Figure 1) is a workload that exhibits sub-linear scaling because of workload-architecture imbalance. While the largest (16-SM) scale model configuration is able to achieve a substantial 1.8× performance improvement compared to the smallest (8-SM) scale model, the improvement diminishes for the 64-SM target system relative to the 32-SM scale model (1.55×), and for the 128-SM target system relative to the 64-SM target system (1.43×).

V. Scale-Down Model

We now present our GPU scale-model simulation method. Figure 3 provides a high-level block diagram. The workflow starts by collecting performance numbers for the scale models, which involves simulating the workload of interest for the scale-model configurations. This is required for both the strong and weak-scaling workload scenarios. Strong scaling also requires collecting the miss rate curve. This is a one-time cost that can be amortized to predict the performance of various target systems. The workload's miss rate curve and scale-model performance numbers are served as input to the scale-model prediction tool, which then produces a performance

prediction for the target system. We now elaborate on the various steps in this workflow.

A. Collecting Miss Rate Curves

The first step in the workflow, under the strong-scaling workload scenario, is to compute the cache miss rate curve. This curve, as previously illustrated in Figure 2, reports MPKI as a function of LLC cache capacity. (Note we need to consider LLC miss rate curves only because we scale the shared LLC and not the SM-private L1 caches in the scale models.) The data points included in the miss rate curve should be such that they cover the cache capacity of the scale models as well as the envisioned target systems. It is important to note that these miss rate curves can be obtained efficiently, much faster (at least two orders of magnitude faster) than detailed architectural simulation. In fact, there is a long history of highly efficient algorithms and implementations to collect and compute miss rate curves. In particular, Conte et al. [20] propose a single-pass algorithm to compute miss rate curves for fully-associative caches based on the stack distance or reuse distance, which is defined as the number of unique memory accesses between two accesses to the same memory location. More recent work by Berg et al. [13] leverages interrupt-based instrumentation and statistical modeling to collect miss rate curves at low overhead. Eklöv et al. [23] approximate miss rate curves using statistical techniques based on the number of (non-unique) references between two accesses to the same memory location, which is substantially faster to compute than the number of unique accesses needed for the stack-distance based approach.

While this body of prior work focused on miss rate curves for CPU systems, only recently did researchers propose a method to compute miss rate curves for GPUs. Collecting miss rate curves is much more challenging for GPUs compared to CPUs because of the large number of concurrent threads and the impact the relative timing and interleaving of memory accesses from the various threads have on the resulting miss rate. In particular, Nugteren et al. [49] propose a method to compute accurate miss rate curves based solely on a list of memory addresses obtained through functional simulation. The model takes into account the degree of thread-level parallelism across warps and thread blocks running on the same or different SMs, the effect warp and memory divergence has on the interleaving of accesses, as well as uniform versus non-uniform access latencies, cache associativity, etc. The end result is a model that predicts cache miss rates within 6% to 8% compared to real hardware, while being 268× faster than detailed architecture simulation.

B. Scale-Model Performance Profile

The second step in the workflow is to obtain the performance profile for the scale-model configurations through detailed architectural simulation — required under both strong and weak scaling. This is done by configuring the number of SMs, the shared LLC capacity, NoC bisection bandwidth, and memory bandwidth in the detailed timing simulator such that the scale models provide a performance profile that is predictive of the envisioned target systems. As aforementioned in Section II, we scale the shared resources proportionally with the number of compute units (SMs). It is important to note that collecting the scale-model performance profile is fairly straightforward using existing simulator infrastructure. The scale models are relatively small in size. By consequence, the simulators and simulation hosts are powerful enough to simulate these scale models with reasonable effort. Moreover, simulating the scale models can typically be done within an affordable time budget compared to simulating the much larger target systems, if at all possible.

C. Target-System Performance Prediction

The miss rate curve and the scale-model performance profile serve as input to the prediction model which forms the core of the GPU scale-model simulation methodology. The prediction model, in its most general form for a strong-scaling workload scenario, considers three regions based on the miss rate curves: the pre-cliff region, the cliff region, and the post-cliff region. Under a weak-scaling scenario, the model considers only the pre-cliff region — which is an oxymoron given there is no cliff under weak scaling because the workload (and its working set) scales proportionally with system size. The three regions are easily defined based on the miss rate curves. The cliff, as explained in Section IV, marks a disproportional drop in the miss rate curve, i.e., the miss rate reduces by more than 2× when doubling cache size and in some cases it even drops to (near) zero as the workload's footprint fits within the available cache capacity. This implies that in the post-cliff region, the majority of, or even all, misses are cold misses. Looking back at the miss rate curves shown in Figure 2, the miss rate curves are in the pre-cliff regions for bfs and pf. For dct on the other hand, we can clearly identify a pre-cliff region up until 17 MB; the cliff region is situated between 17 and 34 MB; the post-cliff region is situated beyond 34 MB. While in theory there could be multiple cliffs, as (parts of) the working sets progressively fit within a particular cache level (first L2, then L1) when scaling system size, we did not observe this behavior for our workloads; we hence assume at most one cliff without loss of generality. We now explain the prediction model in each of these three regions.

1) Pre-Cliff Region: In the pre-cliff region, the miss rate curve is evolving following a steady pace, by definition, i.e., there is no sudden drop in miss rate. This suggests that the performance trend is likely to scale at a similarly steady pace with system size. We leverage this insight to make a prediction in the pre-cliff region. We denote $IPC_{sm;S}$ as the performance number (IPC) for the smallest scale model of size S, and $IPC_{sm;L}$ as the IPC for the largest scale model of size L. The relative scale difference between the smallest and largest scale model is hence equal to L/S. In this work, the smallest scale model features 8 SMs, while the largest scale model features 16 SMs; the relative scale difference hence equals 2×.

Ideally, one would hope that the performance of the largest scale model is L/S times the performance of the smallest

scale model. While this might be the case for a (purely) compute-intensive workload, most workloads involve non-trivial memory accesses and hence deviate from this ideal scaling pattern. The performance of the largest scale model might be lower than L/S times the performance of the smallest scale model. Also, and surprisingly perhaps, the performance of the largest scale model could also be higher than L/S times the performance of the smallest scale model. We define the deviation from ideal scaling using a correction factor C_{sm} measured as follows using the scale models:

$$C_{sm,L/S} = \frac{IPC_{sm,L}/IPC_{sm,S}}{L/S}. \quad (1)$$

A value higher than one, i.e., $C_{sm} > 1$, means that performance scales super-linearly, while a value lower than one, i.e., $C_{sm} < 1$ means sub-linear scaling.

We predict performance of a target system of size T to be equal to the performance of the largest scale model ($IPC_{sm,L}$) times the relative scale difference between the target system and the largest scale model (T/L) times the correction factor $C_{sm,L/S}$ obtained from the scale models:

$$IPC_{ts,T} = IPC_{sm,L} \times \frac{T}{L} \times C_{sm,L/S}. \quad (2)$$

This formula implicitly assumes that performance continues to scale as it did for the smaller scale models.

2) Cliff Region: In the cliff region, the miss rate curve is marked by a sudden drop. At cache sizes beyond the cliff, the workload's footprint resides in the cache and, by consequence, there are very few cache misses apart from cold misses. As a result, the workload will no longer be stalled waiting for memory. The fraction of the execution time that was spent waiting for memory at a small (pre-cliff) cache size is hence eliminated for cache sizes beyond the cliff. The extent to which this impacts performance depends on the relative impact memory stalling has on overall performance. In other words, the larger the memory stall fraction, the more significant the performance impact of surpassing the cliff. We leverage this insight in our prediction model.

We estimate performance of a target system of size T to be equal to the performance of the largest scale model ($IPC_{sm,L}$) times the relative scale difference (T/L) times the inverse of the fraction of time not stalled for memory on the largest scale model:

$$IPC_{ts,T} = IPC_{sm,L} \times \frac{T}{L} \times \frac{1}{1 - f_{mem;sm,L}}. \quad (3)$$

In this formula, $f_{mem;sm,L}$ is the fraction of time an SM in the largest scale model is unable to fetch an instruction because all available warps are waiting for data to come from memory, i.e., it is computed as the ratio of the number of cycles that no instructions were fetched due to a memory stall for all warps divided by the total number of cycles. Dividing the ideal performance ($IPC_{sm,L} \times T/L$) with the fraction of time that an SM does not stall waiting for memory, provides a prediction for the expected performance on the target system.

3) Post-Cliff Region: Note that the cliff region is intermediate, i.e., once beyond the cliff, the miss rate curve is (pretty much) flat as we increase cache capacity further. The reason is that the workload's footprint fits inside the cache, and making the cache bigger does not affect the miss rate. This implies that the miss rate curve in the post-cliff region is constant or follows a steady pace, similarly to what happens in the pre-cliff region. We leverage this insight to estimate performance beyond the cliff. In the post-cliff region, we follow the same model as in the pre-cliff region except that we need to estimate target system performance starting from the smallest system in the post-cliff region, rather the largest scale model.

We hence estimate target system performance as follows:

$$IPC_{ts,T} = IPC_{s,K} \times \frac{T}{K} \times C_{sm,L/S}. \quad (4)$$

We first determine the performance of the smallest system configuration of size K beyond the cliff (i.e., $IPC_{s,K}$) times the relative scale with the target system (i.e., T/K). We subsequently correct this prediction by multiplying it with the correction factor observed for the scale models (i.e., $C_{sm,L/S}$).

D. Discussion

As with any other modeling or simulation approach [48], it is important to understand scale-model simulation's capabilities and limitations. First, scale-model simulation builds on the assumption that the scale models are proportionally scaled down configurations of the target system. More precisely, scale-model simulation assumes that (1) the private per-SM resources (i.e., functional units, private L1 caches, scratch-pad memory, etc.) are the same in the scale models as in the target system; and (2) the resources that are shared across SMs (i.e., LLC, on-chip network, main memory, and inter-chiplet network if applicable) are proportionally scaled down in the scale models relative to the target system. In other words, the scale models are proportionally scaled down versions of the target system while keeping the per-SM configuration unchanged. In particular, if the scale model is a factor F times smaller than the target system, this implies that the scale model features F times fewer SMs, the LLC is F times smaller, the interconnection network features F times less bandwidth, main memory bandwidth is F times less. As the same time, the SMs remain unchanged in the scale model compared to the target system. This implies that the aggregate per-SM resources scale proportionally with system size: the aggregate number of functional units, the aggregate L1 cache size, the aggregate scratch-pad memory size, etc. are a factor F times smaller in the scale model compared to the target system. The proposed scale-model simulation paradigm cannot be readily used when the scale models' SM configuration differs and/or when the shared resources change disproportionally relative to the target system. For example, if one were to predict performance for a next-generation target system in which both the per-SM private resources and the shared resources differ from the previous generation, one would need to construct a scale model with the same per-SM resources and proportionally scaled-down shared resources as in the target system.

TABLE II: Benchmarks under strong scaling: CTA size for the different kernels, footprint (in MB), number of simulated instructions (in millions), and scaling behavior.

Benchmark Name	Abbr.	CTA Size	Footprint (MB)	#Insns (M)	Scaling Behavior
Discrete Cosine Transform [3]	dct	2,304; 36,864; 512	33.0	10,270	super-linear
FastWalsh Transform [3]	fwt	8,192; 4,096; 128	67.1	4,163	super-linear
Back Propagation [17]	bp	8,192	18.8	424	super-linear
Vector Add [3]	va	16,384	50.3	92	super-linear
Async [3]	as	32,768	67.1	218	super-linear
LU decomposition [27]	lu	16,384	16.8	146	super-linear
Stencil [55]	st	2,096	131.9	557	super-linear
Breadth-First Search [17]	bfs	1,024	20.4	257	sub-linear
3D-unet [51]	unet	from 128 to 21,846	615.0	20,071	sub-linear
Sradv2 [17]	sr	4,096	25.2	661	sub-linear
Gradient [3]	gr	4,096; 816; 1,536; 2,048	46.1	318	sub-linear
B+trees [17]	btree	6,000; 10,000	17.4	670	sub-linear
Path Finder [17]	pf	4,630	404.1	4,037	linear
Resnet50 [51]	res50	from 64 to 66,904	1388.1	85,067	linear
SSD-Resnet34 [51]	res34	from 32 to 306,383	845.8	47,369	linear
HotSpot [17]	ht	7,396	12.5	421	linear
Aligned Types [3]	at	2,048	100.0	2,150	linear
Matrix-multiply C=alpha.A.B+beta.C [27]	gemm	4,096	12.6	7,030	linear
2 Matrix Multiplications [27]	2mm	8,192	21.0	12,921	linear
Lattice-Boltzmann Method [55]	lbm	18,000	359.4	553	linear
Black Scholes [3]	bs	15,625	80.1	863	linear

TABLE III: Baseline 128-SM target system.

Parameter	Value
SM clock frequency	1.0 GHz
No. threads per SM	48 warps/SM, 32 threads/warp, 1,536 threads/SM
CTA scheduling	Round-robin
Warp scheduling	Greedy-Then-Oldest (GTO)
L1 cache per SM	48 KB, 6-way, LRU, 384 MSHRs
LLC	34 MB total, 64 slices, 64-way per slice
DRAM bandwidth	2.3 TB/s
NoC	Crossbar, 2.7 TB/s

Second, scale-model simulation is conceived to perform well for a broad set of representative workloads and system configurations. In particular, in this work, we consider a variety of workloads that exhibit linearly, super-linearly and sub-linearly scaling behavior on a typical modern-day GPU architecture. Some of the super-linear scaling benchmarks exhibit a performance cliff as previously mentioned, namely dct and fwt. While a workload may potentially exhibit multiple cliffs, as different sets of the data set progressively fit inside the various cache levels, we observe only a single cliff for our workloads and system configuration. Presumably, the reason is that there is only a single shared cache level (L2) in our system setup — this is in line with modern-day GPUs in which the L2 cache is the last-level cache shared by all SMs, see for example Nvidia's Hopper H100 GPU [1]. Extending scale-model simulation to support workloads and system configurations with multiple cliffs is left for future work but could possibly be accounted for by estimating how each cliff individually affects the respective memory stall fraction. For example, in a system configuration with three cache levels in which L2 and L3 are shared, the cliffs around the L2 and L3 capacities will drastically reduce the respective stall components which can be modeled similarly to what is described above for a single cliff. More generally, evaluating (and possibly extending) scale-model simulation for an even broader set of workloads and system configurations is an interesting avenue for future work.

VI. Experimental Setup

1) Workloads: The benchmarks we use in this work are taken from a variety of benchmark suites, including Rodinia [17], Polybench [27], Parboil [55], and CUDA SDK [3]. We also include workloads from the MLPerf Inference suite [2, 51], for which we use the Sieve sampling methodology [47] to identify representative kernel invocations. We make a distinction between strong versus weak scaling, see Tables II and IV, respectively, which report the benchmarks' CTA size, footprint (in MB), number of dynamically executed instructions (in millions), input data set, and scaling behavior. The set of benchmarks is diverse enough to include benchmarks with varying scaling behavior (sub-linear, linear, and super-linear), see the rightmost column in the respective tables. Note further that the weak-scaling workloads are a subset of the strong-scaling workloads: the reason is that to support weak scaling, the workloads' inputs need to be scalable, which is only possible for a subset of the workloads.

2) Simulator: We evaluate the proposed GPU scale-model simulation methodology through detailed architectural simulation using the Accel-Sim simulation infrastructure [39]. Simulation provides the flexibility to configure scale models that are proportionally scaled down versions of the target system. The largest target system considered in this work is a 128-SM GPU with a 34 MB LLC, a crossbar NoC with 2.7 TB/s bisection bandwidth, and a 2.3 TB/s memory interface. Table III provides details about the other system configuration parameters. The smaller 32-SM and 64-SM target systems, as well as the 8-SM and 16-SM scale models, are proportionally scaled-down versions of the 128-SM baseline

as previously reported in Table I. We obtained per-benchmark miss rate curves using Accel-Sim.

VII. Evaluation

We now evaluate our proposed GPU scale-model simulation methodology. As mentioned before, we consider two scale models, the 8- and 16-SM configurations, to predict the 128-SM and 64-SM target systems. We compare our GPU scale-model simulation method against four other scaling models:

- **Proportional scaling** assumes that the performance achieved on a target system is $S\times$ the performance of the scale model that is $S\times$ smaller. For example, the performance of a 128-SM target system is assumed to be $8\times$ as high as the performance of the 16-SM scale model.
- **Linear regression** assumes that performance scales linearly based on the scale models. A linear regression model (i.e., $y = a \cdot x + b$) is built based on the scale models, which is then used to predict performance for the target systems.
- **Power-law regression** assumes power-law performance scaling (i.e., $y = a \cdot x^b$) with system size.
- **Logarithmic regression** assumes logarithmic performance scaling (i.e., $y = a \cdot log_b(x)$) based on the scale models. Although logarithmic regression performs poorly compared to the other regression techniques, as we will see in the evaluation, we include it here because this is what prior work proposed for CPU scale-model extrapolation [46].

A. Prediction Accuracy: Strong Scaling

Figure 4 reports the prediction error for our proposed GPU scale-model simulation method assuming a strong-scaling workload scenario for (a) the 128-SM target system and (b) the 64-SM target system, compared to proportional scaling and the various regression approaches. The overall conclusion is that scale-model simulation is substantially more accurate. In particular, for the 128-SM target system, the average error for scale-model simulation equals 4% and at most 17%. In contrast, the other approaches are (much) less accurate. Logarithmic regression is the least accurate approach with an average error of 69% and up to 86%. Proportional scaling achieves a lower 22% average prediction error, but the maximum error goes up to 113%. Linear regression achieves a 17% average prediction error, and up to 68%. As expected, proportional scaling and linear regression achieve good accuracy for the linearly scaling benchmarks, while being largely inaccurate for the super-linearly scaling workloads (e.g., dct and fwt) and the sub-linearly scaling workloads (e.g., bfs). Power-law regression is the most accurate of the regression techniques with an average prediction error of 12% and up to 55%. Power-law regression is accurate for the linearly scaling benchmarks, somewhat accurate for the sub-linearly scaling benchmarks (except for bfs), but inaccurate for several super-linearly scaling benchmarks (e.g., dct and fwt). These results confirm that a one-size-fits-all regression strategy is inaccurate. Our proposed GPU scale-model prediction method, which provides a per-workload prediction, is the most accurate approach.

We observe similar results for the 64-SM target system. Scale-model simulation is the most accurate approach with an average prediction error of 3.5% (and at most 13%), substantially outperforming proportional scaling and the regression approaches. The average error equals 48%, 10%, 6%, and 4% for logarithmic regression, proportional scaling, linear regression, and power-law regression, and max errors of 55%, 52%, 23%, and 13%, respectively.

B. Benchmark Behavior under Strong Scaling

Figure 5 visualizes performance for a subset of the benchmarks as a function of system size, again assuming strong scaling. In addition to the real performance curves and the performance curves as predicted with our proposed scale-model simulation method, we also report the predicted performance curves for the two most accurate regression approaches, namely linear regression and power-law regression, as well as proportional scaling. It is interesting to discuss the different categories of workloads based on their scaling behavior. Note that the top row in Figure 5 shows super-linearly scaling workloads; the middle row shows sub-linearly scaling workloads; and the bottom row shows linearly scaling workloads.

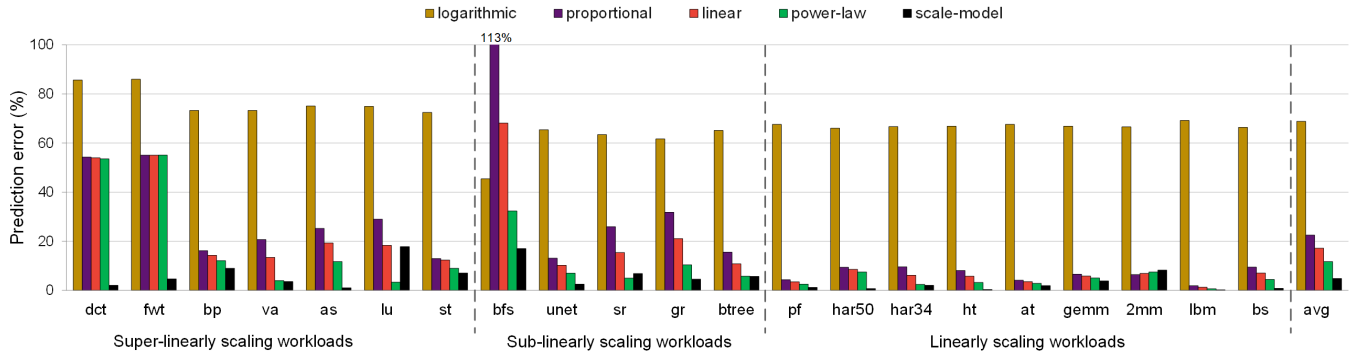
1) Super-Linearly Scaling Workloads: Proportional scaling and the various regression techniques are (fundamentally) unable to predict the super-linear scaling behavior. Our proposed scale-model simulation method on the other hand accurately predicts the super-linear scaling behavior because it is able to anticipate when the active working set starts fitting in the on-chip caches as we scale system size (i.e., the cliff region).

2) Sub-Linearly Scaling Workloads: The proportional-scaling prediction technique is overly optimistic in predicting target system performance. Linear regression is somewhat more accurate, but is still unable (fundamentally so) to predict the sub-linear scaling trend. Power-law regression beats linear regression, but is still not as accurate as our proposed GPU scale-model simulation method.

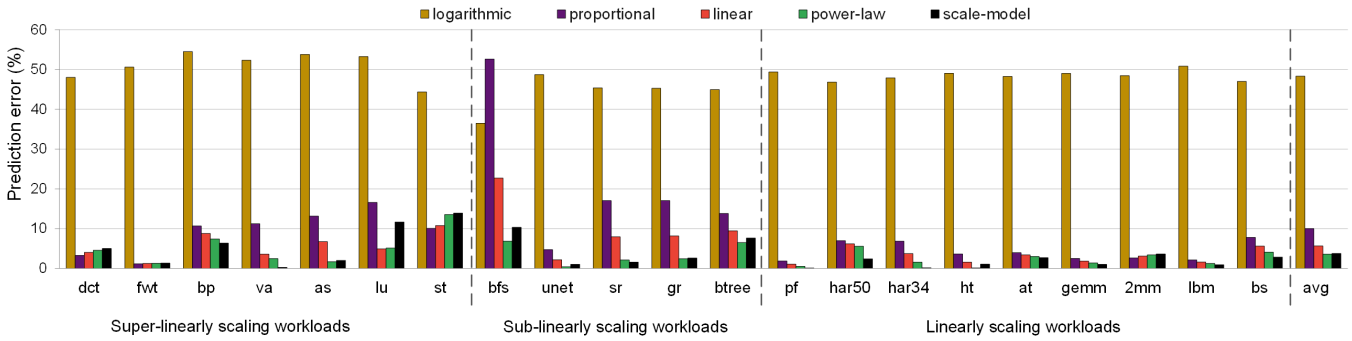
3) Linearly Scaling Workloads: As expected, all methods accurately predict the performance trend of linearly-scaling workloads, even power-law regression. As expected, proportional scaling and linear regression are accurate, and so is scale-model simulation.

C. Prediction Accuracy: Weak Scaling

Predicting target-system performance under weak scaling is somewhat easier than under strong scaling. Indeed, as reported in Figure 6, the different prediction techniques achieve higher accuracy. Nevertheless, scale-model simulation still is the most accurate approach among all, with an average error of 1.7% (and max error of 4.5%) for the 128-SM target system, versus 67%, 7%, 5.2%, and 3.6% average error (and 70%, 18%, 12%, and 9% max error) for logarithmic regression, proportional scaling, linear regression and power-law regression, respectively. The highest errors are observed for bfs and bs, which



(a) 128-SM target system



(b) 64-SM target system

Fig. 4: IPC prediction error under strong scaling for (a) the 128-SM target system and (b) the 64-SM target system. *The proposed GPU scale-model simulation method is substantially more accurate than proportional scaling and regression.*

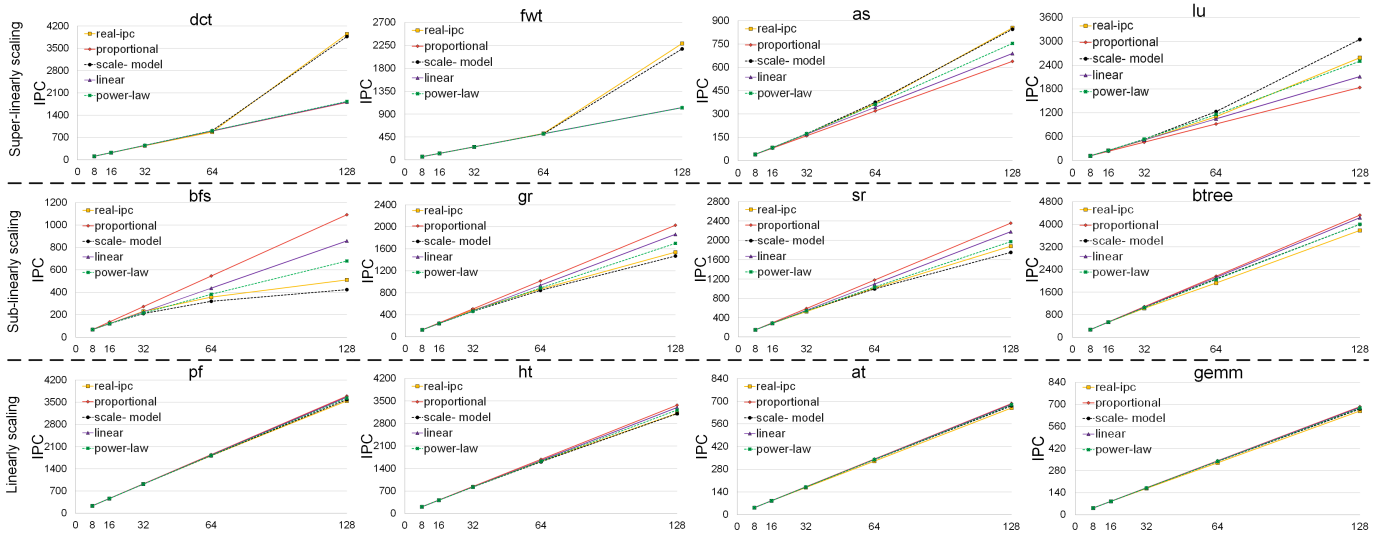


Fig. 5: Performance for select benchmarks as a function of system size assuming strong scaling: super-linearly scaling workloads (top row), sub-linearly scaling workloads (middle row), and linearly scaling workloads (bottom row). *GPU scale-model simulation tracks the real scaling trend accurately in contrast to proportional scaling, linear regression and power-law regression across different workload types with linear, sub-linear, and super-linear scaling behavior.*

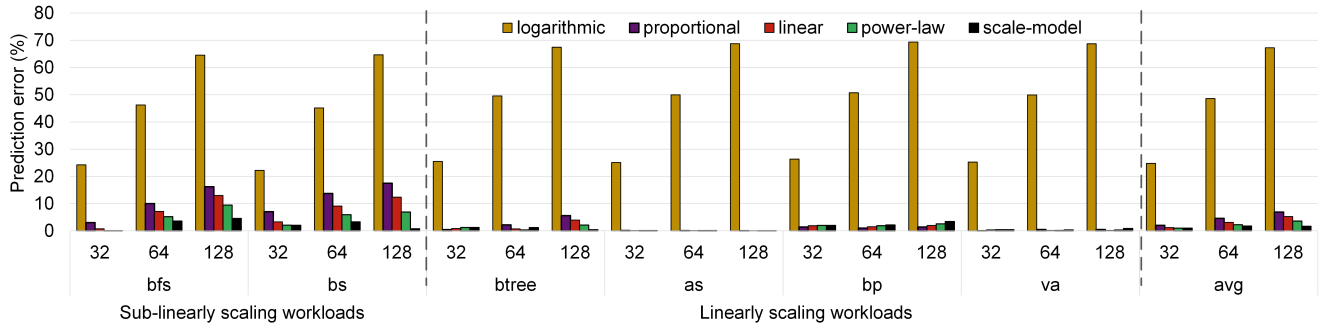


Fig. 6: IPC prediction error under weak scaling for the 32-SM, 64-SM, and 128-SM target systems. *The proposed GPU scale-model simulation method is substantially more accurate than proportional scaling and the regression regression techniques.*

TABLE IV: Benchmark configurations under weak scaling: total CTA size, footprint (in MB), number of simulated instructions (in millions), and scaling behavior. The MCM column denotes the workload configurations used for the multi-chip module (MCM) GPU experiments in Section VII-D.

Benchmark	MCM	CTA	MB	#Insns	Scaling
bfs [17]		128	9.4	30	sub-linear
		256	5.1	61	
	✓	512	10.2	128	
	✓	1,024	20.4	257	
	✓	2,046	40.9	549	
bs [3]	✓	15,625	40	431	sub-linear
	✓	31,250	80	862	
	✓	62,500	160	1,724	
		125,000	320	3,448	
		250,000	640	6,898	
btree [17]		2,500	4.3	167	linear
		5,000	8.7	335	
		10,000	17.4	670	
		20,000	34.7	1,341	
		40,000	69.4	2,682	
as [3]		2,048	4.2	13.5	linear
		4,096	8.7	27	
	✓	8,192	16.78	54	
	✓	16,384	33.6	109	
	✓	32,768	67.1	218	
bp [17]		4,096	2.5	212	linear
	✓	8,192	18.9	424	
	✓	16,384	37.7	848	
	✓	32,768	75.5	1,696	
		65,536	151.0	3,392	
va [3]		1,024	3.1	5.8	linear
		2,048	6.3	11.5	
	✓	4,096	12.6	23	
	✓	8,196	25.2	46	
	✓	16,384	50.3	92	

is not unexpected given their sub-linear scaling behavior, see also Table IV.

Scale-model simulation leads to a substantial speedup under weak scaling, see Figure 7 which reports scale-model speedup relative to simulating both the 8-SM and 16-SM scale models.[1]

[1]We do not observe a substantial simulation time impact under strong scaling because the workload simulated on the scale models is the same as the one simulated on the target system.

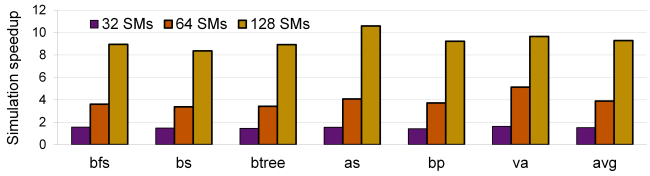


Fig. 7: Speedup through scale-model simulation under weak scaling as a function of target system size normalized to running simulations for the 8-SM *and* 16-SM scale models. *Scale-model simulation leads to substantial simulation speedups under a weak-scaling workload scenario.*

The speedup varies from 1.5× to 3.9× and 9.3× for the 32-SM, 64-SM, and 128-SM target systems, respectively. The reason is the much smaller size of the workload being simulated on the scale models compared to the target system.

D. Case Study: Multi-Chiplet GPUs

We considered monolithic GPU so far, which, as alluded to in the introduction, cannot be scaled beyond the reticle limit. Multi-chip module (MCM) GPUs, which integrate multiple GPU chiplets within a single package using interposer technology, provide a way to continue to scale GPU performance. We now apply the scale-model simulation to predict multi-chiplet GPU performance scaling. We consider scale models with 4 and 8 chiplets to predict 16-chiplet performance (16 chiplets with 64 SMs each, for a total of 1,024 SMs). We consider the multi-chiplet target system configuration from Table V using the (single-kernel) weak-scaling workloads from Table IV except for btree due to simulator limitations. The work within a kernel is scheduled across chiplets for both the scale models and the target system.

Following the general principle of scale-model simulation, the scale models are proportionally down-scaled versions of the target system. In particular, in this case study, we consider the chiplet's configuration to be fixed, and we scale the inter-chiplet network accordingly with system size. More specifically, the 4-chiplet system features an inter-chiplet network with 1/4th the bisection bandwidth of the 16-chiplet target system. The aggregate memory bandwidth and total number of SMs also scale linearly with system size.

TABLE V: The simulated 16-chiplet target system.

Parameter	Value
#SMs/chiplet	64
SM clock frequency	1.7 GHz
CTA scheduling	Distributed [7]
Page allocation	First-touch [7]
LLC	18 MB per chiplet, 64 slices, 64-way per slice
Intra-chiplet NoC	crossbar topology, 1.7 TB/s
Inter-chiplet NoC	fly topology, 900 GB/s per chiplet
Memory	8 memory controllers, 1.2 TB/s per chiplet

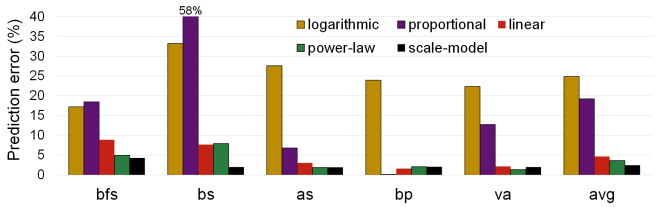


Fig. 8: Multi-chiplet IPC prediction error under weak scaling for the 16-chiplet system. *Scale-model simulation accurately predicts multi-chiplet performance.*

Figure 8 reports IPC prediction error for scale-model simulation compared to the various prediction techniques. Scale-model simulation achieves an average prediction error of 2.5% (and at most 4.3%). Logarithmic regression and proportional scaling are highly inaccurate with an average error of 25% and 20%, and up to 33% and 58%, respectively. Although linear and power-law regression are relatively more accurate (4.7% and 3.7% average error, respectively), the maximum error (9% and 8%) is higher than for scale-model simulation. We also note an average 2.2× (and up to 2.8×) simulation speedup through scale-model simulation for predicting 16-chiplet performance relative to the simulation of the 4-chiplet *and* 8-chiplet scale-model systems.

VIII. Related Work

Architecture scale-model simulation was recently proposed for general-purpose multi-core processors running multi-program workloads. Liu et al. [45, 46] point out that scale models should be proportionally scaled down versions of the target system to be able to accurately model the interference caused by shared resources. To more accurately predict target-system performance based on the scale-model simulation data, Liu et al. leverage machine learning techniques alongside regression. High accuracy is reported when combining support vector machines with logarithmic regression.

Eyerman et al. [25] propose a scale model for Intel's experimental specialized graph accelerator, called PIUMA. The lack of resource sharing among processor cores in the PIUMA fabric makes the development of scale models relatively straightforward, because there are no shared caches, each core has a dedicated memory controller, and a highly scalable interconnection network provides high bandwidth and low latency to each individual core.

In contrast to this prior work in scale-model simulation, this work (1) targets GPUs, (2) lacks the need for training a machine learning model, (3) does not rely on a one-size-fit-all regression model, but instead (4) builds on a per-workload performance model that accurately predicts linear, sub-linear and super-linear performance scaling.

A wide variety of analytical performance models have been proposed for GPUs. Analytical performance modeling can generally be classified in white-box versus black-box models. Black-box models, e.g., machine-learning based models, are relatively easy to construct, are generally accurate, but provide limited insight and often require a significant training effort. White-box models aim at providing insight by modeling first-order principles, but they are elaborate to develop. Several white-box analytical GPU performance models have been proposed over the past decade with different capabilities, see in particular [30, 31, 42, 60]. Black-box models have been proposed as well, see for example [35]. Some machine-learning based techniques aim at predicting GPU performance based on CPU implementations [6, 10].

Architectural simulation is and remains the most prevalent performance evaluation technique. Unfortunately, developing and maintaining a detailed architectural simulator involves substantial time and effort, and running simulations is extremely time-consuming and resource-intensive, i.e., server farms are extensively used to drive architecture exploration and analysis during various stages of the design cycle. A variety of GPU architecture simulators exist, see for example GPGPU-Sim [9], Accel-Sim [39], MGPUSim [56], gem5-GPU [28]. These simulators are widely used, but they are slow, e.g., a typical simulation speed of 6 KIPS is reported for Accel-Sim [39]. NVArchSim (NVAS) [58] is an Nvidia trace-driven simulator that can trade off the level of simulation detail to balance speed and accuracy. MGPUSim [56] exploits parallelism to speed up multi-GPU system simulation on general-purpose multi-core CPU hardware. Sampling is a widely used technique to speed up simulation, and several proposals have been made that are specifically tailored to GPU simulation, see for example [8, 32, 38, 44]. Synthetic miniature benchmarks have been proposed as well to accelerate GPU simulation [64]. The key advantage of scale-model simulation is that it does not rely on a detailed simulation model of the target system to predict its performance unlike sampling, simulator parallelization, and synthetic workloads.

IX. Conclusion

This paper proposed a scale-model simulation methodology for GPUs. Unlike prior work which focused on general-purpose multi-core CPUs or specialized graph analytics accelerators, we find that GPU workloads exhibit different scaling behavior with system size: some scale linearly, while others scale sub-linearly or super-linearly. Using miss rate curves and scale-model performance numbers, the proposed framework predicts performance for a larger-scale target system under both strong-scaling and weak-scaling workload scenarios. Our experimental evaluation demonstrates high accuracy: based on 8- and 16-SM scale models, our framework is able to predict 128-SM target system performance within 4% and 1.7% (and

at most 17% and 4.3%) for strong-scaling and weak-scaling workload scenarios, respectively. Under weak scaling, scale-model simulation also yields a 9.3× simulation speedup. We further illustrate that scale-model simulation predicts multi-chiplet GPU performance scaling with an average error of 2.5% (and at most 4.3%).

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback and suggestions. This work is supported through UGent-BOF-GOA grant No. 01G01421, Research Foundation Flanders (FWO) grant No. G018722N, and European Research Council (ERC) Advanced Grant agreement No. 741097.

REFERENCES

[1] NVIDIA H100 tensor core GPU architecture. https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper. Accessed: 2023.

[2] MLPerf inference v2.0 nvidia-optimized implementations. https://github.com/mlcommons/inference_results_v2.0/tree/master/closed/NVIDIA. Accessed: 2022.

[3] Nvidia CUDA SDK code sample. https://docs.nvidia.com/cuda/cuda-samples/index.html. Accessed: 2022.

[4] M. Abraham, T. Murtola, R. Schulz, S. Páll, J. Smith, B. Hess, and E. Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015.

[5] M. Ahmad and O. Khan. GPU concurrency choices in graph analytics. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 1–10, 2016.

[6] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 725–737, 2015.

[7] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans. MCM-GPU: Multi-chip-module GPUs for continued performance scalability. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 320–332, 2017.

[8] C. A. Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers. Principal kernel analysis: A tractable methodology to simulate scaled GPU workloads. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 724–737, 2021.

[9] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 163–174, 2009.

[10] I. Baldini, S. J. Fink, and E. R. Altman. Predicting GPU performance from CPU runs using machine learning. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing*, pages 254–261, 2014.

[11] N. Beckmann and D. Sanchez. Talus: A simple way to remove cliffs in cache performance. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 64–75, 2015.

[12] E. Berg and E. Hagersten. StatCache: a probabilistic approach to efficient and accurate data locality analysis. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 20–27, 2004.

[13] E. Berg and E. Hagersten. Fast data-locality profiling of native execution. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 169–180, 2005.

[14] D. Biancolin, S. Karandikar, D. Kim, J. Koenig, A. Waterman, J. Bachrach, and K. Asanovic. FASED: FPGA-accelerated simulation and evaluation of DRAM. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 330–339, 2019.

[15] T. E. Carlson, W. Heirman, and L. Eeckhout. Sampled simulation of multi-threaded applications. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, 2013.

[16] T. E. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout. BarrierPoint: Sampled simulation of multi-threaded applications. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, 2014.

[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009.

[18] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, and K. Mai. PROTOFLEX: FPGA-accelerated hybrid functional simulator. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–6, 2007.

[19] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, and K. Mai. A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 77–86, 2008.

[20] T. M. Conte, M. A. Hirsch, and W. mei W. Hwu. Combining trace sampling with single pass methods for efficient cache simulation. *IEEE Transactions on Computers (TC)*, 47(6):714–720, 1998.

[21] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 83–94, 2002.

[22] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 2–12, 2005.

[23] D. Eklov and E. Hagersten. StatStack: Efficient modeling of LRU caches. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 55–65, 2010.

[24] A. C. Elster and T. A. Haugdahl. Nvidia Hopper GPU and Grace CPU highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.

[25] S. Eyerman, W. Heirman, Y. Demir, K. Du Bois, and I. Hur. Projecting performance for PIUMA using down-scaled simulation. In *Proceedings of the International High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2020.

[26] D. Foley and J. Danskin. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro*, 37(2):7–17, 2017.

[27] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. Auto-tuning a high-level language targeted to GPU codes. In *Proceedings of Innovative Parallel Computing (InPar)*, 2012.

[28] A. Gutierrez, B. M. Beckmann, A. Dutu, J. Gross, M. LeBeane, J. Kalamatianos, O. Kayiran, M. Poremba, B. Potter, S. Puthoor, M. D. Sinclair, M. Wyse, J. Yin, X. Zhang, A. Jain, and T. G. Rogers. Lost in abstraction: Pitfalls of analyzing GPUs at the intermediate language level. In *Proceedings of the International Conference on High-Performance Computer Architecture (HPCA)*, pages 608–619, 2018.

[29] S. Hong and H. Kim. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, page 152–163, 2009.

[30] S. Hong and H. Kim. An integrated GPU power and performance model. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 280–289, 2010.

[31] J.-C. Huang, J. H. Lee, H. Kim, and H.-H. S. Lee. GPUMech: GPU performance modeling technique based on interval analysis. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 268–279, 2014.

[32] J.-C. Huang, L. Nai, H. Kim, and H.-H. S. Lee. TBPoint: Reducing simulation time for large-scale GPGPU kernels. In *Proceedings of the International Conference on Parallel and Distributed Processing Symposium (IPDPS)*, pages 437–446, 2014.

[33] A. Ishii and R. Wells. The NVLink-network switch: Nvidia's switch chip for high communication-bandwidth superpods. In *Proceedings of the IEEE Hot Chips Symposium (HCS)*, 2022.

[34] 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)*, 16 (1):1–24, 2019.

[35] W. Jia, K. A. Shaw, and M. Martonosi. Stargazer: Automated regression-based GPU design space exploration. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–13, 2012.

[36] A. Joshi, J. Yi, R. H. Bell, L. Eeckhout, L. John, and D. Lilja. Evaluating the efficacy of statistical simulation for design space exploration. In

Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 70–79, 2006.

[37] H. M. Kamali, K. Z. Azar, and S. Hessabi. DuCNoC: A high-throughput FPGA-based NoC simulator using dual-clock lightweight router micro-architecture. *IEEE Transactions on Computers (TC)*, 67(2):208–221, 2018.

[38] 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 *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 76–86, 2015.

[39] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers. Accel-Sim: An extensible simulation framework for validated GPU modeling. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 473–486, 2020.

[40] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder. Motivation for variable length intervals and hierarchical phase behavior. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 135–146, 2005.

[41] A. Lavin and S. Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4013–4021, 2016.

[42] J. Lee, Y. Ha, S. Lee, J. Woo, J. Lee, H. Jang, and Y. Kim. GCoM: A detailed GPU core model for accurate analytical modeling of modern GPUs. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 424–436, 2022.

[43] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[44] C. Liu, Y. Sun, and T. E. Carlson. Photon: A fine-grained sampled simulation methodology for GPU workloads. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 1227–1241, 2023.

[45] W. Liu, W. Heirman, S. Eyerman, S. Akram, and L. Eeckhout. Scale-model simulation. *IEEE Computer Architecture Letters (CAL)*, 20(2): 175–178, 2021.

[46] W. Liu, W. Heirman, S. Eyerman, S. Akram, and L. Eeckhout. Scale-model architectural simulation. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 58–68, 2022.

[47] M. Naderan-Tahan, H. SeyyedAghaei, and L. Eeckhout. Sieve: Stratified GPU-compute workload sampling. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 224 – 234, 2023.

[48] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam. Architectural simulators considered harmful. *IEEE Micro*, 35(6):4–12, 2015.

[49] C. Nugteren, G.-J. van den Braak, H. Corporaal, and H. E. Bal. A detailed GPU cache model based on reuse distance theory. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 37–48, 2014.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

[51] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, M. J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou. MLPerf inference benchmark. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 446–459, 2020.

[52] A. Sabu, H. Patil, W. Heirman, and T. E. Carlson. LoopPoint: Checkpoint-driven sampled simulation for multi-threaded applications. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 604–618, 2022.

[53] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 3–14, 2001.

[54] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, 2002.

[55] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. mei W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. Technical Report IMPACT-12-01, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 2012.

[56] 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 International Symposium on Computer Architecture (ISCA)*, pages 197–209, 2019.

[57] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Efficient sampling startup for sampled processor simulation. In *Proceedings of the International Conference on High Performance Embedded Architectures and Compilers (HiPEAC)*, pages 47–67, 2005.

[58] O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, N. Chat-terjee, N. Jiang, and D. W. Nellans. Need for speed: Experiences building a trustworthy system-level GPU simulator. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 868–880, 2021.

[59] D. Wang, C. Lo, J. Vasiljevic, N. E. Jerger, and J. G. Steffan. DART: A programmable architecture for NoC simulation on FPGAs. In *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pages 145–152, 2011.

[60] L. Wang, M. Jahre, A. Adileh, and L. Eeckhout. MDM: The GPU memory divergence model. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 1009–1021, 2014.

[61] Y. Wang, Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel, and J. D. Owens. Gunrock: GPU graph analytics. *ACM Transactions on Parallel Computing (TOPC)*, 4(2):1–49, 2017.

[62] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, 2006.

[63] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 84–97, 2003.

[64] Z. Yu, L. Eeckhout, N. Goswami, T. Li, L. K. John, H. Jin, C. Xu, and J. Wu. GPGPU-MiniBench: Accelerating GPGPU micro-architecture simulation. *IEEE Transactions on Computers (TC)*, 64(11):3153–3166, 2015.

[65] X. Zhao, A. Adileh, Z. Yu, Z. Wang, A. Jaleel, and L. Eeckhout. Adaptive memory-side last-level GPU caching. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 411–423, 2019.

[66] X. Zhao, M. Jahre, and L. Eeckhout. Selective replication in memory-side GPU caches. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 967–980, 2020.

ARTIFACT APPENDIX

A. Abstract

We now discuss how to redo the experiments presented in the paper. We describe how to collect the scale-model simulation results — we provide the scale-model (and target-system) AccelSim configuration files as well as the workload inputs — and we provide a tool to predict target-system performance. To enable reproducing the target-system predictions without having to rerun the scale-model simulations, we also provide the performance numbers for the scale models as well as the miss rate curves for all the benchmarks and workload-scenarios (strong scaling and weak scaling) discussed in the paper. We also provide target-system performance results such that the errors for the various prediction methods reported in the paper can be verified. Finally, we also report simulation times for the scale models and target systems such that the reported speedup numbers can be verified. The appendix mostly focuses on the prediction tool, i.e., we do not repeat how to run GPU architecture simulations using AccelSim to collect the inputs for the prediction model.

B. Artifact check-list (meta-information)

- **Run-time environment:** AMD Ryzen 7 3700X 8-core processor; Nvidia RTX 3080 GPU; Ubuntu 18.04; Python 3 or above.
- **Execution:** Collecting benchmark traces and simulating the scale models using AccelSim takes time, ranging from less than an hour to several weeks depending on the benchmark. The prediction step is instantaneous.
- **Metrics:** The essential metrics to extract from the scale models include their IPC. Furthermore, as discussed in Section V, miss rate curves (i.e., MPKI as a function of system size) are needed to determine the presence of any cliff regions.
- **Disk space required:** Because AccelSim is trace-based, sufficient disk space is needed for storing the workload traces. The prediction tool uses very little disk space.
- **Publicly available:** Yes, open-source.

C. Description

1) How to access: A public repository is available at https://github.com/scaleDownGPU/scaleModel.git and https://figshare.com/projects/scaleDownGPU/187638, and contains the following: (1) the configuration files for both the scale models and the target systems, (2) the benchmarks' inputs for both strong and weak scaling, (3) the IPC numbers for the scale models (and target systems for prediction error evaluation purposes) for all benchmarks, (4) the miss rate curves for all benchmarks, and (5) the prediction model tool.

2) Input data sets: We use benchmarks from a variety of widely-used and well-recognized GPU benchmark suites, see also Section VI. For the strong-scaling scenario, we use a benchmark's default input data set. For the weak-scaling scenario on the other hand, we adjust the benchmark's input data set such that the amount of work performed scales proportionally with system size. We identified six benchmarks that could be appropriately scaled up with system size, either by modifying the input or by adjusting the code. We now describe how the input was scaled for these six benchmarks — the repository contains the five weak-scaling inputs for these six benchmarks:

- bfs [17]: The amount of work performed is scaled up by modifying the input file. Within the benchmark folder, the `graphgen.cpp` file can be customized to generate input files of different sizes.
- bp [17]: The bp benchmark takes a single input value (multiple of 16), which represents the number of input elements. By modifying this input value, one can alter the number of CTAs and, consequently, scale up the benchmark proportionally to system size.
- btree [17]: To adjust the amount of work performed by the B-tree benchmark, one can modify the values of j and k in the `command.txt` input file. j represents the order of the B-tree and determines the maximum number of keys a node in the B-tree can hold — a higher values of j results in a higher branching factor and shallower tree structure. k is the size of the working set, which determines the number of operations to be executed on the B-tree. Changing the j and k values allows for changing the amount of work performed.
- bs [3]: In contrast to the above three benchmarks, bs requires changes to the source code to modify the amount of work. Specifically, the *OPT_N* variable signifies the workload's load (array size).
- as [3]: The specific variable to be altered here is n which represents the workload's magnitude (number of elements to be processed). Note that this variable needs to be a multiple of 16.
- va [3]: The *numElements* variable allows for adjusting the number of array elements in the input.

3) Simulator configuration files: The repository contains the AccelSim configuration files of the scale models and the target systems. As mentioned in the paper, the scale models are obtained by proportionally scaling down the configuration of the (largest) target system of interest.

4) Scale-model performance results and miss rate curves: The repository also contains the performance results for the scale-models (and target systems for error evaluation purposes) as well as the miss rate curves for all benchmarks such that the target-system performance predictions can be reproduced without having to re-simulate the scale models.

D. Installation

We refer to the AccelSim simulator, see https://github.com/accel-sim/accel-sim-framework, for instructions regarding how to install and use the simulator. The prediction tool itself only requires having installed Python 3.

E. Experiment Workflow

To facilitate the reproducability of the prediction results, we provide the prediction model as a Python program.

1) Package Dependencies: To run the model, ensure that Python 3 is installed with the following packages:

```
sudo apt update
sudo apt install python3
python3 -m pip install numpy scipy scikit-learn ruptures
```

2) How to Run: The model can be run once the packages are installed. To do so, execute the following command:

```
python3 scaleModel.py <value1> <value2> <value3>
... <valueN>
```

As noted, the model needs various inputs:

- <**value1**>**:** the IPC of the smallest scale model.
- <**value2**>**:** the IPC of the largest scale model.
- <**value3**> to <**valueN**>**:** the miss rate curve, or MPKI numbers for the scale models and the various target systems. For example, assuming scale models with N and $2N$ SMs (or chiplets), and target systems with $4N$, $8N$ and $16N$ SMs (or chiplets), these values should be the respective MPKI numbers for these five systems, sorted from the smallest to the largest system.

When running the model, the user is asked to provide the number of SMs (or chiplets) for the smallest scale model — the largest scale model is assumed to be twice as big. The model then predicts performance for the target systems by doubling the number of SMs (or chiplets) for as many MPKI numbers as provided. For example, if 5 MPKI numbers are provided with the smallest scale model featuring 8 SMs, the model will predict performance for target systems with 32, 64 and 128 SMs. If the model detects a cliff, it will prompt the user to provide $f_{mem;sm,L}$ or the fraction of time an SM in the largest scale model is unable to fetch instructions due to a memory stall — this is the case for the dct and fwt benchmarks for which the cliff appears at 128 SMs. Note that the same model is used to predict multi-chiplet performance, e.g., predicting 16-chiplet performance based on 4- and 8-chiplet scale models is done by providing IPC numbers for the scale models and MPKI numbers for 4, 8 and 16 chiplets.

Note that the prediction tool is not limited to using the 8-SM and 16-SM scale models to predict 32-SM, 64-SM and 128-SM target systems, as done throughout the paper. One could also use the 16-SM and 32-SM scale models to predict 64-SM and 128-SM target system performance. We observed (during artifact evaluation) that the error is higher for the strong-scaling workload scenario. For the 64-SM target system, we note an average error of 5% for scale-model simulation, power-law and linear regression versus 10% for proportional scaling and 24% for logarithmic regression. For the 128-SM target system, scale-model simulation achieves an average error of 10% versus power-law (15%), linear (16%), proportional (19%) and logarithmic regression (50%). The higher error results from the 32-SM scale model being an outlier with respect to the general scaling trend for three benchmarks, namely bfs (23% and 56% error for scale-model simulation for the 64-SM and 128-SM target systems, respectively), fwt (25% and 39% error, respectively) and gr (14% and 30% error, respectively). Nevertheless, despite the higher absolute errors, the overall conclusion is that scale-model simulation outperforms the alternate approaches including proportional scaling, linear, power-law and logarithmic regression.

F. Evaluation and expected results

The output of the `scaleModel.py` program is threefold: (1) measured IPC for the scale models; (2) predicted IPC for the target system(s); and (3) a graph showing performance as a function of system size for the four prediction methods evaluated in the paper: logarithmic, linear, and power-law regression, as well as scale-model prediction.