Quantifying and Mitigating the Cold-Start Problem in GPU Simulation

Jonas Sys[1], Mahmood Naderan-Tahan[2], Seyyed Hossein SeyyedAghaei Rezaei[2], Lieven Eeckhout[2]

Abstract

Every year, more and more AI and machine learning papers are published [1]. Many of these machine learning algorithms rely on GPUs to train their models. Additionally, many other workloads, ranging from molecular simulation to graph traversal, benefit greatly from the parallel processing power of GPUs. This increased demand for parallel processing has led to an increasing need for innovation, constantly improving GPUs. In order to verify that these changes are improvements, simulators [2] are used. Modern simulation techniques rely on sampling [3, 4]. These sampling techniques often assume a perfectly warmed-up hardware state, however this is not always the case. This article measures the impact of the cold-start problem on GPU workloads (both in hardware and in simulator), and proposes some mitigations.

1. Introduction

Since their inception in 1968, by the *Evans and Sutherland Computer Corporation*, GPUs have come a long way [5]. Gone are the fixed-function pipelines of the past, replaced by the programmable shaders of today. Machines that used to be the size of a room are now small enough chips to be embedded onto CPUs.

2007 introduced the CUDA platform [6, 7], which allowed for general-purpose computing on GPUs. This was a turning point for the GPU industry, as it allowed for the use of GPUs in a wide range of applications. Today, GPUs are used in a wide range of applications, from machine learning to molecular dynamics simulations.

This increase in demand for parallel processing has led to an increase in the complexity of GPUs. To further improve on the design and micro-architecture of these complex chips, changes have to be rigorously verified. This is where simulators come in. Building a new GPU from the ground up, in silicon, is costly and very time-consuming, which does not lend itself to a fast-paced feedback loop. Instead, simulators are used to model the behavior of the chip, allowing for quick verification that changes are, in fact, improvements.

Many of these simulators are proprietary, unreleased to the public. However, one that is frequently used in academia is Accel-Sim [2]. Accel-Sim is a very flexible simulator, allowing for a wide range of hardware platforms to be simulated. It is also rather accurate, within the limits set by opaque hardware specifications.

The main drawback is that simulating an entire workload or benchmark could easily take weeks, if not months [3]. To speed up this process, sampling techniques are used. A subset of the kernels in the workload is selected, and only these kernels are simulated. If these kernels are sampled correctly and are representative of the entire workload, the results can be extrapolated to the entire workload.

Some of these techniques are *Principal Kernel Analysis* [3] and *Sieve* [4]. Based on profiler data, they cluster all the kernels in the workload. From each of these clusters, a representative is selected to be simulated.

The PKA technique goes even further, by projecting the performance of the first part of a kernel onto the rest of the kernel. The main idea is that, for almost any kernel, the IPC stabilizes after a certain number of instructions. Once this stabilization is detected, the simulation is halted, and it is assumed that the rest of the kernel will have the same IPC.

However, these sampling techniques assume that the hardware is in a perfectly warmed-up state. This is not always the case, since intermittent kernels are omitted. These omitted kernels might affect the hardware state, most notable the caches and/or TLBs, which in turn affects the performance of the kernels that are simulated. This problem is not unique to GPU simulation, but is also present in CPU simulation [8], and it is known as the *cold-start problem.*

While the cold-start problem has been extensively studied in CPU simulation, little to no research has been done on its impact on GPU simulation. In the context of CPU simulation, solutions to the cold-start problem are based on warming up the data structures in the simulator, by simulating additional instructions. However, additional simulation is expensive, as simulation is a lot slower than execution in hardware. To mitigate this problem in CPU simulation, a number of techniques have been proposed [9, 10, 11, 12, 13, 14, 8]. Another avenue that has been explored is the search for a correction factor [15], which avoids additional simulation by analyzing the instruction trace.

[1]Ghent University
[2]Department of Electronics and Information Systems, Faculty of Engineering and Architecture, Ghent University

This work makes the following contributions:

- It quantifies the impact of the cold-start problem on GPU workloads. To this end, we will look at both hardware (in-silicon) results, and simulator results.
- It proposes a number of mitigations to the cold-start problem in GPU simulation, inspired by the main avenues used in CPU simulation. We will also compare these methods against one another, to see which one is the most effective.

2. The cold-start problem in hardware

Using NVIDIA's Nsight Compute [16] tool, we were able to profile a number of workloads[3]. To profile these workloads, an NVIDIA GeForce RTX 3080 [17] was used. This GPU has 68 SM cores, and 6 MB of L2 cache.

Each workload was profiled twice: once normally, as it would run in hardware, and once with the caches flushed between kernel invocations. Nsight Compute supports this option by using the `--cache-control=all` argument.

Below is a list of workloads that were profiled:

- The following benchmarks from the Cactus suite [18]:
  - Gromacs [19] and LAMMPS [20] (with both rhodo (LMR) and colloid (LMC) inputs); two molecular simulation workloads,
  - Gunrock [21] on both road (GRU) and social networks (GST),
  - DCGAN, neural style transform (NST) [22], reinforcement learning (RFL), spatial transformer (SPT) [23] and language translation (LGT) from PyTorch, and
- The following MLCommons benchmarks (from their MLPerf® Inference v2.0 collection):
  - The ResNet50 model [24],
  - Both MobileNet and ResNet34 variants of the SSD model,
  - The Bidirectional Encoder Representations for Transformers (BERT) [25], and
  - The 3D U-Net model [26]
- The DCT implementation present in the CUDA Samples [27].

You can find the results of this experiment in Figure 1. This figure shows the relative IPC difference (relative to the non-flushed IPC: $\frac{IPC_f - IPC_{nf}}{IPC_{nf}} \cdot 100\%$) for each workload listed above. We note that the most promising workloads are DCGAN and Gunrock (on road traversal) from the Cactus suite, along with DCT from the CUDA Samples.

[3]Due to time constraints, only the first 20 000 kernels of each workload were executed.

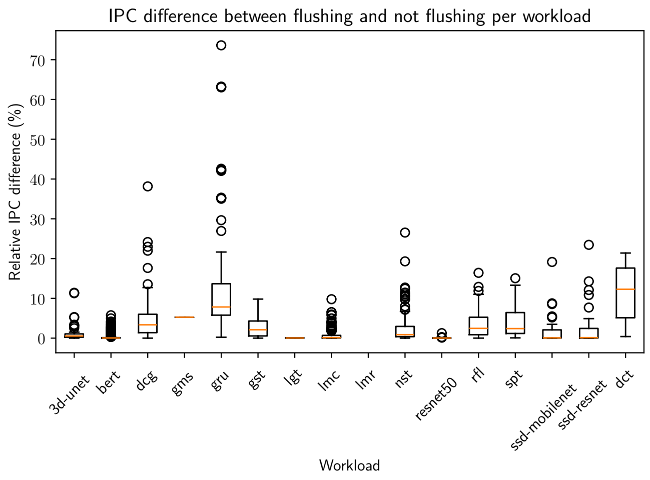


Figure 1: Hardware IPC differences

However, sampling methods like *Sieve* [4] rely on a kernel's weight; i.e. the number of instructions in that kernel relative to the total number of instructions in the workload. Additionally, for each cluster computed by such methods, the weights of the kernels in that cluster are summed to get the cluster's weight. That cluster weight is then used when generalizing the results to the entire workload; i.e. a weighted average is taken. When we take these weights into account, we get a different result.

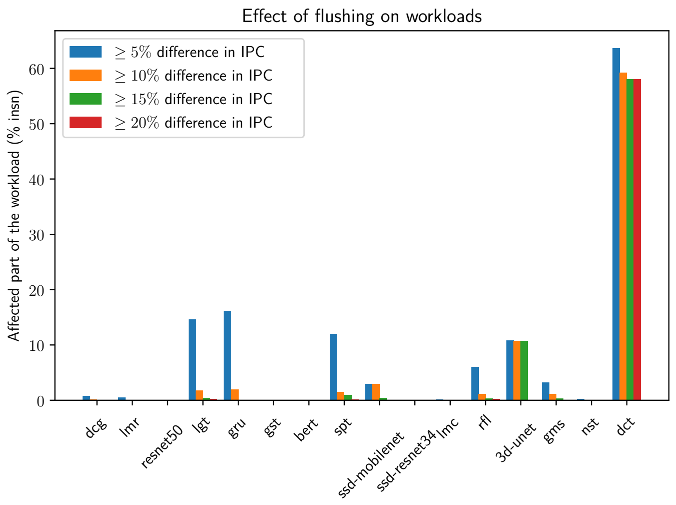


Figure 2: Weighted hardware IPC differences

In Figure 2, you can see the weighted IPC differences for each workload. For each workload, we have identified all kernel invocations with at least 5% (resp. 10%, 15%, and 20%) IPC difference. The weights of all of those kernels were summed up, which is what you see. For 3D-

UNet, for example, kernels worth around 10% of the workload have an IPC difference of between 15% and 20%.

Using that figure, we get a slightly different set of affected workloads. Moving forward, we will focus mostly on the DCT and 3D-UNet workloads, but some of the Cactus benchmarks are still mildly affected.

2.1. Data Reuse

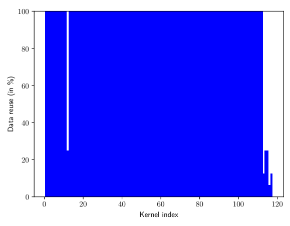
A factor that will prove to be rather important, is the degree of data reuse between kernels. We will focus mostly on the forward data reuse; i.e. the data that is used by a kernel, then used again by the next. For any kernel k_i, we define the forward data reuse fwd_i as:

$$fwd_i = \frac{|M_i \cap M_{i+1}|}{|M_i|} \tag{1}$$

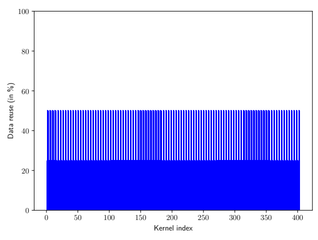
Where M_i is the memory footprint (the set of unique memory addresses accessed) of kernel k_i.

In Figure 3, we have visualized the forward data reuse for two workloads. The first (Figure 3a) is the DCT implementation which we already mentioned, and the second is the recursiveGaussian implementation from the CUDA SDK (Figure 3b).

As you can see, the inter-kernel data reuse in the DCT application is rather high, consistently hitting 100%. On the other hand, the recursiveGaussian application caps out around 50%.



(a) Forward data reuse for DCT



(b) Forward data reuse for recursiveGaussian

Figure 3: Forward data reuse

3. The cold-start problem in Accel-Sim

After profiling the workloads in hardware, we moved on to simulating them in Accel-Sim [2]. Accel-Sim is a very flexible simulator, allowing for a wide range of hardware platforms to be simulated. It is built on top of the GPGPU-Sim simulator [28], which is a cycle-accurate simulator.

Before discussing the results, we will first explain the simulator setup. We used the (pre-tested) NVIDIA GeForce RTX3070 configuration. Some of the more interesting configuration parameters are shown in Figure 4.

Once again, we ran each workload twice: once without flushing the caches between kernel invocations, and once with the caches flushed. Accel-Sim supports this option by using the `--flush-l1 --flush-l2` arguments.

Configuration parameter	Value
L2 cache size	4 MB
Number of sets in L2 cache	64
L2 cache block size	128 B
L2 cache associativity	16
Number of memory controllers	16
Number of SMs	46
L2 Latency	187 cycles
DRAM Latency	254 cycles

Figure 4: Simulator configuration parameters

We decided to limit our experiments to the DCT and 3D-UNet workloads, as those were the most promising in hardware. Another workload that showed promise was the OceanFFT workload from the CUDA SDK. For feasibility, we limited each workload's execution to 130 kernels (which is only a limit for the 3D-UNet workload).

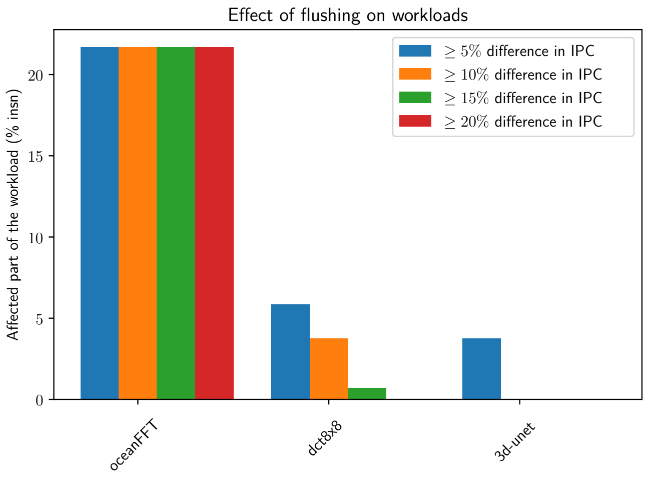


Figure 5: Weighted simulated IPC differences

The results of this experiment are shown in Figure 5. Once again, we took into account the weights of each kernel, showing the weighted IPC differences for each workload.

From this, we gather that the cold-start problem is much less severe in the simulator than in hardware. We assume that this is caused by the fact that the simulator is an idealized version of the hardware. Some details might have gotten lost, which in turn might have mitigated the cold-start problem. A lot of hardware details are gated by the industry, and are not publicly available, so the simulator developers have to rely on micro-benchmarks to attempt to reverse-engineer these details.

3.1. OceanFFT

Besides suffering a lot from the cold-start problem, the OceanFFT workload also has an oddity. As you can see in Figure 6, the IPC values are higher when we flush the caches.

Index	IPC (flushing)	IPC (no flushing)	Forward Reuse (%)
1	2482.26	2487.96	12.50%
2	640.58	439.84	50.00%
3	1401.63	1397.86	0.00%
4	683.88	684.10	25.00%
5	1161.11	1160.73	n/a

Figure 6: OceanFFT IPC values

This is something we would not expect; we expect the caches to speed up execution, not slow it down. The kernel that is impacted the most by the cold-start problem is the second one, which suffers approximately 31% difference.

We assume that it runs faster with flushed caches because it only reuses a small fraction of the data from the previous kernel. The first kernel has a memory footprint of 33.5 million addresses, of which the second kernel reuses only 12.5% (4 million addresses).

If a lot of the cache lines are dirty, this causes a lot of write-backs, which might pile up in the memory hierarchy. This can stall the memory pipeline, which in turn causes the processors to stall, reducing the IPC.

4. Mitigations

In order to mitigate the cold-start problem, we have come up with a number of solutions:

- **Perfect warmup:** a naive approach would be to simulate all preceding kernels, forcing the caches to be warm. However, this goes directly against the idea of sampling, and slows down the simulation significantly.

- **Memory-only warmup:** a more refined approach would be to only simulate the memory instructions of the preceding kernels. This way, the caches are warmed up, but the simulation is still faster than simulating the entire kernel. Additionally, instead of simulating each preceding kernel, we limit ourselves to the most recent ones, as we expect these to have the biggest impact, due to temporal locality.

- **Correction factor:** using data from the instruction trace, as well as the simulator output, we can compute a correction factor. This will prove to be able to increase the accuracy of very inaccurate kernels, at the cost of a slightly lower median accuracy.

4.1. Memory simulation

Using a modified version of the Accel-Sim tracing tool, we were able to generate an additional trace for each kernel. This trace includes only the memory instructions, which can be used to quickly warm up the caches.

We have identified four DCT kernels which suffer a lot from the cold-start problem, and have simulated them with and without the memory-only warmup. Up to ten preceding kernels were simulated (in memory-only mode). Additionally, we compare these with the perfect warmup (where every single preceding instruction is simulated), as well as the full memory warmup (where all memory instructions from all preceding kernels are simulated).

Two main observations can be made from Figure 7:

1. A single kernel warmup is usually more than enough to warm up the caches. The accuracy of all the kernels above increased to at least 97%.
2. There is still a difference between the perfect accuracy, and the full memory accuracy. We assume that this is due to the instructions being re-ordered: since intermittent instructions (e.g. ALU instructions) are removed from the traces, some memory instructions might get re-ordered. This results in a different cache state, which might in turn affect the simulator results.

4.2. Correction factor

We have come up with a formula that is able to partially mitigate the cold-start problem, without any additional simulation. It is based on the following observations:

- We can estimate an upper bound for the impact of the cold-start problem by looking at the number of cold misses. A subset of these misses will be due to the cold-start problem; while another part will be due to the inherent nature of the workload.

- The fraction of cold misses that are due to the cold-start problem is proportional to the degree of forward data reuse. Only reused accesses can be sped up by caches, so these are the only ones that can be affected by the cold-start problem.

- Multiple memory requests can be served in parallel by the DRAM controllers. This means that we should take into account the memory controller occupancy.

- The number of cycles lost due to the cold-start problem is proportional to the difference in latency between DRAM and the L2 cache.

Using these observations, we have come up with the following formula:

$$IPC = \frac{insn}{cycles_f - \Delta} \quad (2)$$

$$\Delta = \frac{accesses}{controllers \cdot line} \cdot fwd_{i-1} \cdot (DRAM - L2) \quad (3)$$

[4]

Where:

- IPC is the final, corrected IPC;

[4]Since we use a subtractive factor for the number of cycles, this formula can not be used for OceanFFT.

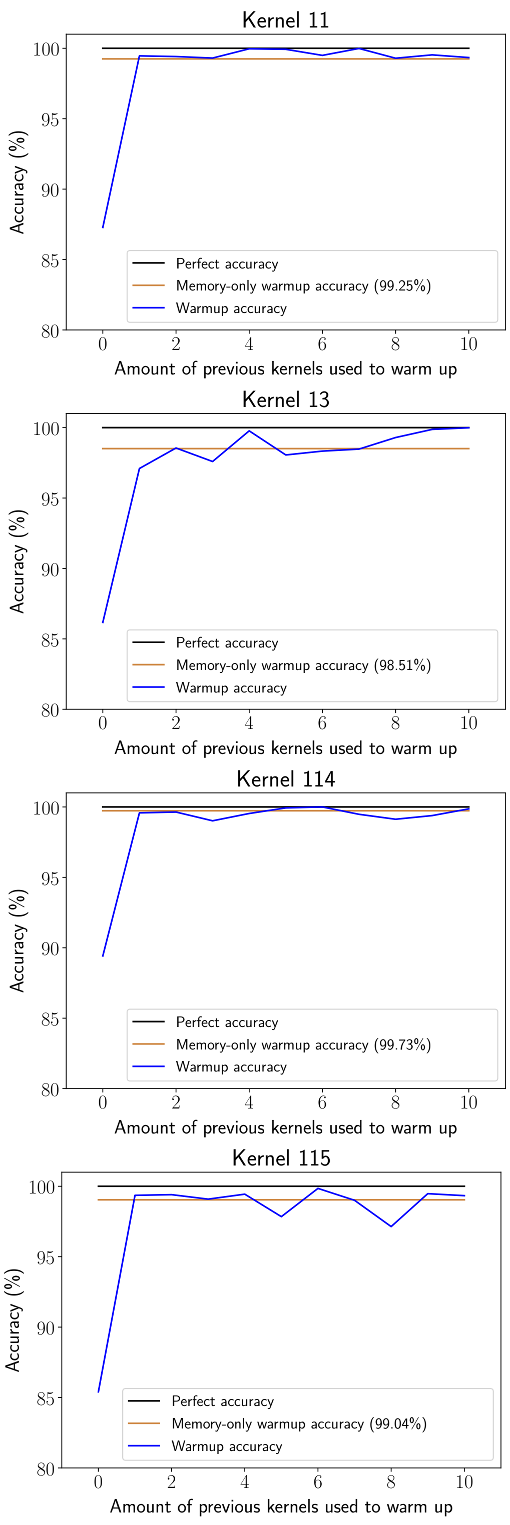


Figure 7: DCT mitigation results

- *insn* is the number of instructions dynamically executed by the kernel;
- $cycles_f$ is the number of cycles used in the flushed case;
- Δ is the correction factor;
- *accesses* is the number of unique DRAM accesses in the kernel (e.g. the number of memory requests due to cold misses);
- *controllers* is the number of memory controllers, weighted by their occupancy;
- *line* is the cache line size;
- fwd_{i-1} is the forward data reuse of the *previous* kernel; and
- $DRAM$ and $L2$ are the latencies of the DRAM and L2 cache, respectively.

4.3. Comparison

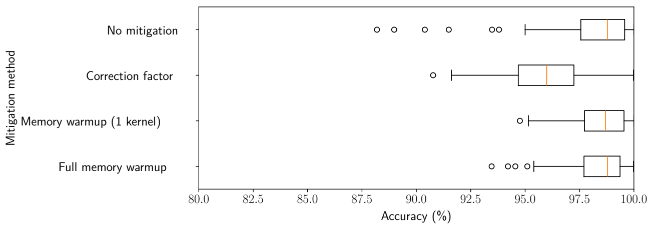


Figure 8: Comparison of mitigations

In Figure 8, we have compared the different mitigation strategies. As you can see, both mitigations are able to eliminate the suffering kernels. However, the correction factor does suffer from a lower median accuracy. Simulating the memory instructions from the previous kernel, however, is a very accurate approach for a rather low computational cost. This is the most promising mitigation strategy, and the one we recommend going forward.

5. Conclusion

In this article, we have quantified the impact of the cold-start problem on GPU workloads. We started by analyzing the impact in hardware, seeing that it is an issue for a number of workloads. We have made a note on the importance of data reuse, and how it affects the cold-start problem.

After that, we moved on to simulating the workloads in Accel-Sim. We have seen that the cold-start problem is much less severe in the simulator than in hardware, but it is still present. An unexpected result was the OceanFFT workload, which ran faster with flushed caches.

Finally, we have looked into some mitigation strategies. The most promising of these is based on additional simulation, and artificial warming of the caches. This method managed to improve the accuracy drastically, without a significant computational cost. The other avenue we explored was a correction factor, based on the number of cold misses and the degree of data reuse. This method was able to eliminate the suffering kernels, but at the cost of a slightly lower median accuracy.

References

[1] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto, Y. Shoham, J. Clark, and R. Perrault, "The ai index 2021 annual report," 2021.

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

[3] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled gpu workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, ACM, Oct. 2021.

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

[5] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (gpu)," *IEEE Micro*, vol. 41, p. 42–51, Nov. 2021.

[6] NVIDIA Corporation, "Cuda," 2022.

[7] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?," *Queue*, vol. 6, p. 40–53, Mar. 2008.

[8] L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John, "Blrl: Accurate and efficient warmup for sampled processor simulation," *The Computer Journal*, vol. 48, no. 4, pp. 451–459, 2005.

[9] G. Lauterbach, "Accelerating architectural simulation by parallel execution of trace samples," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 1, pp. 205–210, 1994.

[10] R. Kessler, M. Hill, and D. Wood, "A comparison of trace-sampling techniques for multi-megabyte caches," *IEEE Transactions on Computers*, vol. 43, no. 6, pp. 664–675, 1994.

[11] T. Conte, M. Hirsch, and K. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 468–477, 1996.

[12] T. Conte, M. Hirsch, and W.-M. Hwu, "Combining trace sampling with single pass methods for efficient cache simulation," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 714–720, 1998.

[13] J. Haskins and K. Skadron, "Minimal subset evaluation: rapid warm-up for simulated hardware state," in *Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors. ICCD 2001*, pp. 32–39, 2001.

[14] J. Haskins and K. Skadron, "Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation," in *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.*, pp. 195–203, 2003.

[15] D. A. Wood, M. D. Hill, and R. E. Kessler, "A model for estimating trace-sample miss ratios," in *Proceedings of the 1991 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, SIGMETRICS91, ACM, Apr. 1991.

[16] NVIDIA Corporation, "NVIDIA nsight compute."

[17] NVIDIA Corporation, "NVIDIA ampere ga102 gpu architecture," whitepaper, 2021.

[18] M. Naderan-Tahan and L. Eeckhout, "Cactus: Top-down gpu-compute benchmarking using real-life applications," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, Nov. 2021.

[19] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs," *The Journal of Chemical Physics*, vol. 153, Oct. 2020.

[20] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[21] 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*, vol. 4, pp. 3:1–3:49, Aug. 2017.

[22] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *ArXiv*, vol. abs/1508.06576, 2015.

[23] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *ArXiv*, vol. abs/1506.02025, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[26] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2016.

[27] NVIDIA Corporation, "CUDA samples."

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