Analyzing Machine Learning Workloads Using a Detailed GPU Simulator

Jonathan Lew*, Deval A. Shah*, Suchita Pati**, Shaylin Cattell*, Mengchi Zhang†, Amruth Sandhupatla*, Christopher Ng*, Negar Goli*, Matthew D. Sinclair**, Timothy G. Rogers†, Tor M. Aamodt*

*University of British Columbia, **University of Wisconsin-Madison, †Purdue University

***Abstract*—Machine learning (ML) has recently emerged as an important application driving future architecture design. Traditionally, architecture research has used detailed simulators to model and measure the impact of proposed changes. However, current open-source, publicly available simulators lack support for running a full ML stack like PyTorch. High-confidence, cycle-accurate simulations are crucial for architecture research and without them, it is difficult to rapidly prototype new ideas.**

In this paper, we describe changes we made to GPGPU-Sim, a popular, widely used GPU simulator, to run ML applications that use cuDNN and PyTorch, two widely used frameworks for running Deep Neural Networks (DNNs). This work has the potential to enable significant microarchitectural research into GPUs for DNNs. Our results show that the modified simulator, which has been made publicly available with this paper[1], provides execution time results within 18% of real hardware. We further use it to study other ML workloads and demonstrate how the simulator identifies opportunities for architectural optimization that prior tools are unable to provide.

***Index Terms*—GPGPU-Sim, Machine Learning, cuDNN, PyTorch**

I. Introduction

Machine learning is being employed to tackle a rapidly growing set of problems. In recent years DNNs have made striking advances in accuracy. Training DNNs requires massive amounts of computational power, which GPUs can provide. While the industry has rapidly introduced changes to GPU architectures to support ML training, such as Tensor Cores and NVLINK, academic researchers have largely focused on designing inference accelerators, partly due to the lack of support in current GPU architecture simulators for running these workloads. In this paper, we address this shortcoming by updating GPGPU-Sim [1], a widely used GPU simulator, to accurately simulate state-of-the-art GPUs running DNNs.

Popular ML frameworks, such as TensorFlow and PyTorch , typically provide a high-level Python application programming interface (API) to developers. Calls to this API invoke computation on a GPU via specialized libraries such as cuBLAS and cuDNN. To achieve the highest levels of performance, these closed-source, precompiled libraries are typically provided by hardware vendors and take advantage of the vendor's detailed knowledge of their product's microarchitecture, which is typically not fully described in publicly available documentation. As a result, popular open-source GPU architecture simulators such as GPGPU-Sim, gem5 [2], and Multi2Sim [3] are unable to run applications that make use of these precompiled libraries. In this paper, we focus on enabling support for cuDNN, given its wide popularity amongst several DNN frameworks such as Caffe, PyTorch, and TensorFlow.

Overall, we make the following contributions in this paper:

- We modify GPGPU-Sim to enable running cuDNN- and PyTorch-based applications.
- We introduce checkpointing support to GPGPU-Sim given the large runtime overhead of simulators.
- We analyze cuDNN-based workloads in our modified GPGPU-Sim and identify new opportunities for optimizations.

II. Background

A. Machine Learning Frameworks

GPUs are widely used to accelerate the execution times of ML workloads. A key enabler is that ML frameworks such as PyTorch build on top of optimized libraries such as NVIDIA's cuBLAS and cuDNN. These libraries employ fast algorithms (e.g., Winograd) for Matrix Multiplication, the key operation behind many neural network computations.

B. Measuring GPU performance

1) NVProf: Profilers like NVProf [4] provide high-level information about the architectural behavior of ML workloads at low overhead. Like GPGPU-Sim, NVProf gives many statistics, including IPC and the number of loads and stores instructions. Recent papers have used tools like NVProf to profile ML workloads [5], however, they are only able to provide high-level analysis on the application's behavior. GPGPU-Sim provides detailed information on memory usage, power efficiency, and can easily be extended to provide additional statistics. In addition, GPGPU-Sim can also be used to prototype architectural optimizations.

2) Simulation: Other prior work has simulated ML workloads, but they use private simulators [6], making comparison against our approach difficult. On the other hand, we simulate ML workloads at high fidelity in the widely-used and publicly-available GPGPU-Sim, accurately simulating state-of-the-art Pascal and Volta models [7], [8]. Moreover, the fact that other papers use disparate simulators for ML workloads makes it crucial to provide better, publicly available tools.

III. Implementation

In this section, we briefly describe the modifications that were required to simulate cuDNN-based applications in

[1]Source code available at https://github.com/gpgpu-sim/gpgpu-sim_distribution (dev branch)

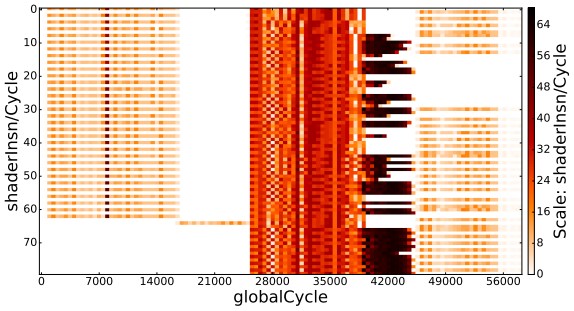

Fig. 1. Forward Convolution (Winograd Nonfused) Shader IPC Plot

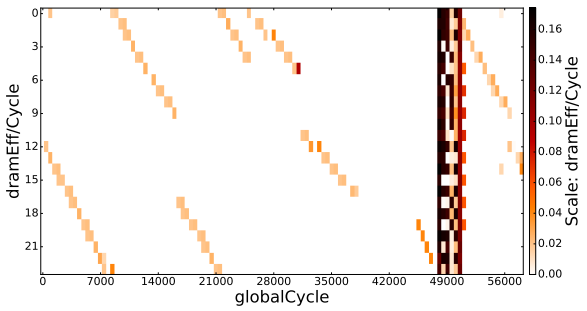

Fig. 2. Forward Convolution (Winograd Nonfused) DRAM Efficency Plot

GPGPU-Sim and evaluate it against hardware. Additional details of the implementation can be found elsewhere [9].

Changes in GPGPU-Sim: GPGPU-Sim extracts all PTX code embedded within an application using NVIDIA's `cuobjdump` tool and parses it using a program loader. Unfortunately, cuDNN-based applications are dynamically linked to the cuDNN library, which `cuobjdump` does not resolve. We resolved this by statically linking against the external library. Additionally, we add support for several CUDA API functions and additional PTX instructions used by cuDNN kernels.

Additional Support: We also added additional support for ML workloads including a new approach to debugging functional simulation errors resulting in incorrect outputs. At a high level, we compare the execution of every instruction in GPGPU-Sim to the result obtained from executing it on hardware. Moreover, we add checkpointing support to help users simulate only a certain region in an application. We execute the application in the functional simulation mode until the start of the region and save the state for the user to resume simulation in Performance mode any number of times.

Correlation: We use a 32-bit floating-point version of MNIST (LeNet trained with the MNIST dataset) to correlate GPGPU-Sim's execution time with a GeForce GTX 1050. We compared the number of GPU cycles as reported by GPGPU-Sim against NVProf while running MNIST on hardware. GPGPU-Sim achieves a correlation of 82%.

IV. Case Study

In this section, we use the modified GPGPU-Sim simulating a NVIDIA TITAN V to study the characteristics of a cuDNN-based program, conv_sample, which performs common ML operations such as forward, backward data, and backward filter convolutions. Due to space limitations, we only show results for shader IPC and DRAM efficiency for the Forward Convolution operation (using Winograd Nonfused algorithm) – a more detailed study can be found in [9]. This study is enabled by GPGPU-Sim and AerialVision [10] and uses the enhanced memory model described in [7]. Figure 1 (the y-axis is the shader number) shows that the forward convolution exhibits several distinct phases. For example, in some phases only a single core actively commits instructions while in other phases all cores commit many instructions (and achieve high per-shader IPCs). The hashing described in [7] is not enabled for this study, which might explain the partition camping observed in Figure 2. Figures 1 and 2 (the y-axis is the bank number) also show that when the IPC is highest, the memory efficiency is low. This highlights compute-bound phases in the program (for example, cycles 39000 to 46000). These results demonstrate how GPGPU-Sim can be used to identify regions of interest in applications. Furthermore, GPGPU-Sim can help identify (and prototype) optimization opportunities that profilers like NVProf cannot, such as turning off cores during the phases they are not used to reduce power.

V. Conclusion

In this paper, we describe changes made to GPGPU-Sim to enable it to run applications that use NVIDIA's cuDNN library (including PyTorch-based applications). We use the resulting modified simulator, which has been made publicly available with this paper, to study ML workloads and analyze their behavior. Our results demonstrate that GPGPU-Sim can help identify optimization opportunities that higher-level tools like NVProf cannot. Moreover, since most DNNs deployed today are trained using GPUs, this work enables significant microarchitectural research into DNNs.

References

[1] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads using a Detailed GPU Simulator," in *ISPASS*, 2009, pp. 163–174.

[2] A. Gutierrez, B. Beckmann, A. Dutu, J. Gross, J. Kalamatianos, O. Kayiran, M. Lebeane, 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 *HPCA*, 2018.

[3] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: a simulation framework for cpu-gpu computing," in *PACT*, 2012, pp. 335–344.

[4] NVIDIA, "Profiler's user guide," 2018.

[5] H. Zhu, A. Phanishayee, G. Pekhimenko, B. Schroeder, B. Zheng, M. Akrout, A. Pelegris, and A. Jayarajan, "Benchmarking and Analyzing Deep Neural Network Training," in *IISWC*, 2018.

[6] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *HPCA*, 2018, pp. 78–91.

[7] M. Khairy, A. Jain, T. M. Aamodt, and T. G. Rogers, "Exploring modern GPU memory system design challenges through accurate modeling," *CoRR*, vol. abs/1810.07269, 2018.

[8] A. Jain, M. Khairy, and T. G. Rogers, "A quantitative evaluation of contemporary gpu simulation methodology," *POMACS*, vol. 2, no. 2, p. 35, 2018.

[9] J. Lew, D. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers, and T. M. Aamodt, "Analyzing machine learning workloads using a detailed GPU simulator," *CoRR*, vol. abs/1811.08933, 2018.

[10] A. Ariel, W. W. Fung, A. E. Turner, and T. M. Aamodt, "Visualizing complex dynamics in many-core accelerator architectures," in *ISPASS*, 2010, pp. 164–174.