Improving GPU Utilization in ML Workloads Through Finer-Grained Synchronization

Reese Kuper
University of Wisconsin-Madison
rkuper@wisc.edu

Suchita Pati
University of Wisconsin-Madison
spati@cs.wisc.edu

Matthew D. Sinclair
University of Wisconsin-Madison
sinclair@cs.wisc.edu

Abstract

In recent years, machine learning has transformed society, including significant improvements in accuracy on a variety of tasks including image/text classification and video recognition. Moreover, on-going efforts are applying machine learning to new applications domains including autonomous agents, natural language processing, and speech translation. Although more specialized options also exist, general purpose GPUs are widely used in both industry and academia for machine learning, especially for machine learning training. However, unlike convolutional neural networks, current and next generation algorithms like recurrent neural networks, and Transformer (or attention) networks typically use narrower kernels and have numerous inter-kernel dependencies that limit parallelism. Thus, these algorithms often do not fully utilize the GPU. To overcome these inefficiencies, we propose to rethink the conservative kernel-level synchronization for these workloads. In particular, we build on the insight that often in these workloads, computation in subsequent kernels is only dependent on a small subset of the computation from prior kernels. By dynamically uncovering these dependencies, we can modify the GPU to issue work as soon as the work it depends on completes, and avoid unnecessarily waiting until the entire kernel finishes. This increases overall utilization of the GPUs while reducing idle time, thus improving performance and energy efficiency.

1 Introduction

In recent years, machine learning (ML) has emerged as an important application domain, driving the requirements for future systems. General-purpose GPUs (GPGPUs) in particular have emerged as the accelerator of choice for ML, especially for the time- and computationally-intense ML training phase, because GPGPUs offer a strong combination of programmability, performance, and energy efficiency. In recent years, GPGPUs have been especially useful for convolutional neural networks (CNNs). CNNs are often compute bound and utilize large, dense matrices, making them an ideal fit that fully utilize the GPU. Accordingly, companies such as NVIDIA have devoted significant resources to designing both highly tuned software (e.g., cuDNN) and adding new hardware features (e.g., Tensor Cores) that accelerate commonly used ML operations.

YArch '21, Feb 2021, USA

.

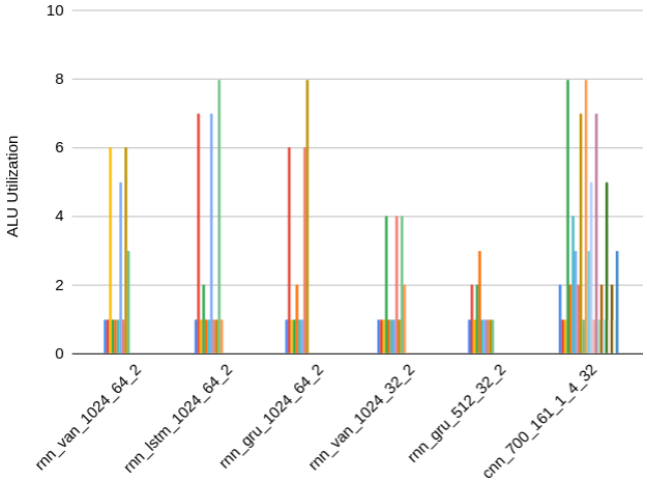
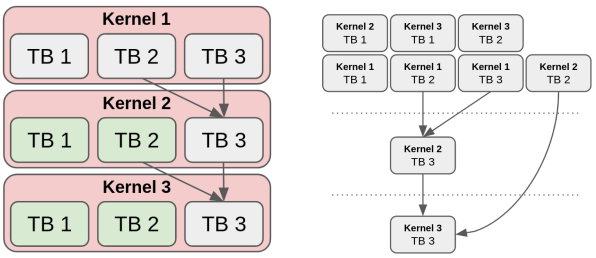


Figure 1. ALU utilization on an NVIDIA Volta Titan V. 1-3 is 'low', 4-6 is 'medium', and 7-10 is 'high' utilization. The runs are labeled as [hidden-size_batch-size_timestep_type] for RNNs and [width_height_channel_batch_filters] for CNN.

However, current and next generation ML workloads like recurrent neural networks (RNNs), and Transformer networks [1] (based on attention networks [2]) have different characteristics from CNNs. Unlike CNNs, these algorithms are often memory bound and have heterogeneous, input-dependent computation, often based on a job's sequence length, that does not fully utilize the GPU, as shown in Figure 1. Figure 1 shows that most kernels (including GEMMs, element-wise add and activation kernels) invoked during RNN (vanilla, LSTM and GRU) training for popular networks such as Google's Neural Machine Translation (GNMT) [3] and DeepSpeech2 [4] have very poor ALU utilization; only a few kernels achieve the GPU's peak utilization.

Normally, modern GPUs use a bulk-synchronous programming model for these workloads, dividing the computation into a number of layers (each of which contains one or more GPU kernels). This makes the algorithms easy to understand, design, and, often, easier to map to high-level frameworks like PyTorch and Tensorflow). Moreover, techniques such as fusion can reduce the number of layers and increase efficiency in some situations (at the cost of increased intra-kernel synchronization) [5–14].

Nevertheless, the increasing depth of ML workloads create distinct phases. At the end of each phase, processing units are often idle while waiting for other calculations to complete.



(a) Current kernel barriers **(b)** Proposed scheme

Figure 2. Independent TBs (in green) could execute once TBs they depend on in prior kernels complete, but current GPUs enforce kernel barriers. Our proposal uses dataflow information to schedule TBs once dependencies are met.

Moreover, expensive, inefficient global synchronization (e.g., returning to the host CPU) is required to ensure that all computations in the phase are complete before the next phase can begin. In CNNs, this often worked well because most or all computations in subsequent kernels depended on all of the computations in the previous layer. However, this is not the case for RNNs, and Transformers.

In order to analyze dependencies in ML workloads, we developed a tool which outputs a TB granularity dependency graph. Figure 2a shows a snippet (for brevity) highlighting these TB dependencies for three kernels in a GRU layer. In this figure, TB 1 and TB 2 of both kernel 2 and kernel 3 are independent of their respective preceding kernels. However, they cannot immediately execute; instead, they must wait for all TBs in the previous kernel to complete. Therefore, we propose to exploit dynamic dataflow ideas, in combination with recent advances in GPU stream schedulers [15], to avoid unnecessary, overly conservative global synchronization between kernels. By extending the stream scheduler to utilize these dependencies, we can schedule work across kernels as soon as it is ready, increasing utilization, reducing idle time, and improving performance and energy efficiency.

2 Proposal

Since TBs often depend on only a small subset of TBs in prior kernels, we propose to extend the stream scheduler to utilize a dynamic, per TB dependence graph instead of a per kernel dependence graph as in modern GPUs [15]. Thus, our stream scheduler effectively combines all kernels (and streams) from a given job into a single large pool, and uses the dependence information to determine when it is safe to schedule a given TB. This requires two major changes:

Finding and storing inter-TB dependencies: Initially, we will statically profile each application to extract its inter-TB dependencies. Here, we exploit the insight that ML training is repetitive (both in terms of layers as well as iterations), usually executing similar kernels in a loop. Thus, we can leverage this property to extract dependency information in the first invocation of a kernel and use it in their subsequent occurrences. However, this means that the initial launch of each kernel will still use the bulk-synchronous barriers. Thus, we will further extend this approach to dynamically identify dependencies through hardware-software co-design. We will use a combination of software hints (addresses or address range) and a Bloom filter (to be stored in the stream scheduler's memory) that tracks addresses being accessed by executing kernels. When a new kernel is added to a stream (or inspected for the first time by the stream scheduler), we will search the Bloom filter using the hints to see which, if any TBs in prior kernels are accessing the addresses, and use this information to construct the dependence graph.

Extending stream scheduler: We will modify the GPU stream scheduler to utilize the inter-TB dependence information and launch TBs as soon as their dependencies from prior kernels have completed. As part of this change, we will also augment the stream scheduler to determine an efficient priority when deciding between TBs from older kernels and newer kernels. This enables concurrent execution of TBs across multiple (formerly consecutive) kernels and reduces the overhead of going to the host.

3 Evaluation

To evaluate our proposed design, we will use two components. First, we will extend our tool that extracts directed dependency graphs to analytically model and provide an upper bound on the potential gains from overlapping independent computations for a GPU of given size. Next, we will extend Accel-Sim [16] to incorporate our dependence information into its stream scheduler, and modify the stream scheduler to schedule TBs as soon as its dependencies are complete and the GPU has available resources. We will initially use static dependence graphs generated from prior runs to test our stream scheduler changes, before moving to dynamic dependence graphs. To test our implementation, we will initially use microbenchmarks like DeepBench [17, 18], then progress to larger, more representative benchmarks like BERT [19], DeepSpeech2, GNMT, GPT-2 [20], and GPT-3 [21].

4 Related Work

Our proposed work builds on and leverages prior GPU work on dependence graphs and kernel fusion. In particular, Wireframe [22] and Puthoor et al. [15] propose significant advances in stream scheduling (or command processors) that schedule TBs based on dataflow graphs. The key difference between this work and ours is that Wireframe focuses specifically on intra-kernel dependencies using a hardware dependence graph, which ignores more challenging, memory intensive, and difficult task of identifying and scheduling independent work across kernels. Puthoor, et al. also utilize dependence graphs, but focus on scheduling independent work from different streams (command queues), which can be identified statically by the programmer.

Similarly, prior work intelligently fuses kernels together [6, 7, 23]. This removes the overhead of returning to the CPU between kernels and improves occupancy and performance, but adds expensive intra-kernel barriers. In contrast, we seek

to provide a middle point between this work and work that exploits dependence graphs: the benefits of kernel fusion, without expensive intra-kernel barriers, and exploiting dependence graph information across kernels, instead of only within kernels. Nevertheless, since our work builds on these works, we will compare our approach to them.

References

[1] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," 2019.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS, 2017.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *CoRR*, vol. abs/1609.08144, 2016.

[4] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, A. Y. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, C. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin," in *Proceedings of the 33nd International Conference on Machine Learning*, ICML, pp. 173–182, 2016.

[5] J. Appleyard, T. Kociský, and P. Blunsom, "Optimizing Performance of Recurrent Neural Networks on GPUs," *CoRR*, vol. abs/1604.01946, 2016.

[6] G. Diamos, S. Sengupta, B. Catanzaro, M. Chrzanowski, A. Coates, E. Elsen, J. Engel, A. Y. Hannun, and S. Satheesh, "Persistent RNNs: Stashing Recurrent Weights On-Chip," in *Proceedings of the 33nd International Conference on Machine Learning*, ICML, pp. 2024–2033, 2016.

[7] I. El Hajj, J. Gomez-Luna, C. Li, L.-W. Chang, D. Milojicic, and W.-m. Hwu, "KLAP: Kernel Launch Aggregation and Promotion for Optimizing Dynamic Parallelism," in *49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO, pp. 1–12, 2016.

[8] J. Filipovič, M. Madzin, J. Fousek, and L. Matyska, "Optimizing CUDA Code by Kernel Fusion: Application on BLAS," *The Journal of Supercomputing*, vol. 71, p. 3934–3957, Oct. 2015.

[9] J. Fousek, J. Filipovič, and M. Madzin, "Automatic fusions of cuda-gpu kernels for parallel map," *SIGARCH Comput. Archit. News*, vol. 39, p. 98–99, Dec. 2011.

[10] F. Khorasani, H. A. Esfeden, N. Abu-Ghazaleh, and V. Sarkar, "In-Register Parameter Caching for Dynamic Neural Nets with Virtual Persistent Processor Specialization," in *Proceedings of 51st IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2018.

[11] A. Li, B. Zheng, G. Pekhimenko, and F. Long, "Automatic Horizontal Fusion for GPU Kernels," 2020.

[12] M. Sivathanu, T. Chugh, S. S. Singapuram, and L. Zhou, "Astra: Exploiting Predictability to Optimize Deep Learning," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), p. 909–923, Association for Computing Machinery, 2019.

[13] M. Springer, P. Wauligmann, and H. Masuhara, "Modular Array-Based GPU Computing in a Dynamically-Typed Language," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY 2017, (New York, NY, USA), p. 48–55, Association for Computing Machinery, 2017.

[14] G. Wang, Y. Lin, and W. Yi, "Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, (USA), p. 344–350, IEEE Computer Society, 2010.

[15] S. Puthoor, X. Tang, J. Gross, and B. M. Beckmann, "Oversubscribed command queues in gpus," in *Proceedings of the 11th Workshop on General Purpose GPUs*, GPGPU-11, (New York, NY, USA), pp. 50–60, ACM, 2018.

[16] 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)*, pp. 473–486, 2020.

[17] S. Narang, "DeepBench." https://github.com/baidu-research/DeepBench, 2016.

[18] S. Narang and G. Diamos, "An update to DeepBench with a focus on deep learning inference." https://svail.github.io/DeepBench-update/, 2017.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," 2020.

[22] A. A. Abdolrashidi, D. Tripathy, M. E. Belviranli, L. N. Bhuyan, and D. Wong, "Wireframe: Supporting Data-Dependent Parallelism through Dependency Graph Execution in GPUs," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, (New York, NY, USA), p. 600–611, Association for Computing Machinery, 2017.

[23] F. Zhu, J. Pool, M. Andersch, J. Appleyard, and F. Xie, "Sparse persistent rnns: Squeezing large recurrent networks on-chip," in *Proceedings of 6th International Conference on Learning Representations*, ICLR, 2018.