Reducing Latency in CPU-GPU Interactions

URL: emmaloool.github.io/15400-Project

Project Description

Faculty/Collaborators

My faculty advisor will be Todd Mowry of SCS's Computer Science Department (specifically, Systems). I will be working in the scope of the Parallel Data Lab, a collaboration between CS and ECE departments at CMU. Primarily, I will be working with and reporting to Pratik Fegade, a PhD student in CSD advised by Professor Mowry.

Description

This project will study latency interactions between the GPU and the CPU for GPGPU (General-Purpose GPU) workloads. Specifically, we will be profiling canonical GPGPU workloads, analyzing them under the lense of artificial inference using the following neural networks.

After a couple of decades of re-architectures, the GPU now supports direct programming on a multi-core system; this enables users to leverage the GPU's inherent parallelism to perform faster computations. On a system-on-a-chip (SOC), the GPU (the dedicated data-parallel computation unit) and the CPU (the main system processor) interact together with other components. To run computations on the GPU, data, operations, and control flow must be designated by the CPU. The GPU receives and runs this program, called a kernel. GPUs batch together groups of individual threads, then execute them together in a single SIMD pipeline under a single "warp". With highly data-parallel inputs, GPUs can execute the same instructions across similar enough data, known as SIMD, to achieve high performance. In this way, the SOC can harness parallelism by splitting computations over multiple cores.

Because of this, researchers are interested in reducing the overhead to run programs on the GPU. One major source of overhead is *latency*, the amount of time needed for a transaction to complete. In parallel computing, data computations can be performed in tandem relatively quickly thanks to hardware support. The latency in communicating computed results back and forth in system memory becomes a bottleneck as a potentially sequential operation, dragging total program runtime. Specifically, in GPGPU computation, a major source of operations in these programs requires sending information back and forth between the processor. A typically huge amount of data needs to be transmitted on the system bus to the GPU, and the computed results need to be transmitted back to the program in main memory. The *CPU-GPU latency problem* asks how we can reduce communication latency between the CPU and GPU to provide better overall program performance.

Past research on this problem has encompassed methods to compress data in order to reduce the bulk of information needed to be communicated. However, in another direction, it has become increasingly desirable to equip the GPU similar control flow processing capabilities as that of the CPU so that it can effectively perform more decision-making on its own, thus reducing communication between the two processors. For my project, we will be investigating the underlying relationships in the program control flow of canonical GPGPU workloads, specifically for machine learning programs.

An interesting problem in migrating control flow to the GPU is that of *conditional branch*

divergence, which can occur when threads inside warps lead to different execution paths; additional overhead is introduced when dynamic prediction is used to intuit dividing computation at program runtime in order to alleviate this problem. Another control-flow related problem is *synchronization*, or syncing up data on the GPU with results in main memory; the control flow on the GPU needs to decide when and how to send data back to the main processor. Additional capabilities we would also like to study on the CPU side is to enable efficient *prefetching* (overlapping memory accesses to improve overall computation) and *speculation* (preparing for future data usage by offloading the data to the GPU sooner based on predictions).

In the span of my project, we will seek to create heuristics to handle branch divergence and synchronization concerns efficiently on-device; if time allows, we will also seek to study methods to enable prefetching and speculation in our models. To infer program control flow and create heuristics for migrating control to the GPU, we will be leveraging four types of neural networks applied to workload. These neural networks are TreeLSTMs, StackLSTMs, Sparsely-Gated Mixture-of-Experts, and Beam Search over RNN outputs. Three of them have working open-source implementations available (TreeLSTMs, StackLSTMs, and Sparsely-Gated Mixture-of-Experts). There are several satisfactory implementations for the Beam Search algorithm over RNN outputs available, but for now, we have enough source code available for the other three workloads to start analysis, so we will refine a reference implementation at a later point.

Project Goals

75%: At a minimum, the first step will be to benchmark and profile the execution of the workloads (which may be provided as open-source materials, but may be modified by us). Then we will perform machine learning inference on each of them to create heuristics for inferring control flow behavior between the CPU and the GPU.

100%: For our final goal, we aim to perform additional static/dynamic analysis on the workloads to observe overlapping memory accesses, and then similarly, perform inference to intuit the mechanism of prefetching, or overlapping memory accesses with other computations and accesses, for these GPU workloads. Ultimately based on the heuristic created, we will strive to implement a rudimentary form of prefetching for GPU programs.

125%: The stretch goal will be to implement and benchmark implementations of each of the workloads with speculative execution. Potential challenges associated with this goal would be to performance machine learning inference on our workloads in order to predict future usage of data and subsequently servicing these data requests ahead of their computation needs.

Milestones

1st Technical Milestone for 15-300 (Fall)

By the end of the semester at minimum, I aim to gain familiarity in understanding and implementing the sequential version of the neural network workloads by reading the papers provided to me by Pratik, as well as others that may come along the way. Namely, my primary challenge will be to gain an understanding of general neural network concepts and workloads that we will be implementing and using for analysis. To gain familiarity with running these workloads, I will need to obtain the software

and platforms (mentioned below) and compute resources (if needed) from Professor Mowry and Pratik. This includes identifying the set of characteristic GPGPU workloads we'll use as reference benchmarks and to train the neural networks. The goal is that at the end of this semester, I will be equipped to start performance analysis and running the neural networks in the spring.

Bi-weekly Milestones for 15-400 (Spring)
Because this project's aims are largely investigative, these bi-weekly milestones may be modified as I continue refining the project's goals and tasks with Professor Mowry and Pratik.

January 27: Benchmark four workloads using GPGPU-Sim to obtain run-time/compile-time statistics, including control flow instructions and performance measurements. This baseline information will serve as a point of reference later.

February 10: Collaborate with Pratik to design a methodology to cluster similar control flow instructions (and other data)

February 24: Train the four neural networks implementations on the set of canonical GPGPU workloads, creating a heuristic for recognizing and handling branch prediction.

March 16: Apply neural networks to a different set of unique GPGPU workloads for comparison of performance and analyze results

March 30: Train the four neural networks implementations on the set of canonical GPGPU workloads, creating a heuristic for recognizing and handling prefetching of sub-workloads.

April 13: Apply this set of neural networks to a different set of unique GPGPU workloads for comparison of performance and analyze results

April 27: Complete implementation/tweaking of neural network models. In addition, commence and finish packaging and documenting source

May +: Wrap-up workload analysis, write reflections, and give presentation

Literature Search

Previous efforts contributed to by Todd Mowry carried along the same theme of reducing the latency in GPU/CPU communication via compression of some sort; formulating techniques to reduce extra bit operations and data redundancy [1], and employing idle on-device resources (units) to alleviate bottlenecks in code [2]. Although we will not be studying compression mechanisms in the scope of my project, I view these papers as good reference materials for how GPU architectural studies and implementations are structured and how analysis is typically performed. The papers utilize GPGPU-Sim, a tool I will also be using to simulate a collection of reference GPGPU workloads.

To better understand how neural networks perform inference, it's also relevant for me to read other papers that have implemented systems using them. InferLine is one such system which executes ML inference pipelines in order to optimize controlling performance configurations, in order

to reduce end-to-end latency constraints [3]. This implementation is in the domain of my project, as it employs machine learning inference and having the same end goal of reducing communication by optimizing controller configurations. As another form of of neural networks, recurrent neural networks are increasingly studied because of their efficacy in modeling sequences. The GRNN (GPU-based recurrent neural network model) inference library also lies in our project's domain, as it addresses improving upon existing RNN inference implementations but improves upon resource utilization, data reuse, and synchronization costs [4]. These are all concerns that my project seeks to address via inferring on program control flow. This, I believe it would be apt to study how this implementation achieves these improvements.

My project utilizes four distinct neural networks in order to evaluate different benefits to creating heuristics for program control flow. To understand how to use the four neural networks, I will need to read up on their design and implementation, as well as understand their tradeoffs.

LSTMs, or Long Short-Term Memory networks, are suitable for classifying control flow because its suitability to classifying and predicting time-based computations; control flow represents time-ordered sequences of operations. Two of the neural networks that we will be using are TreeLSTMs [5] and StackLSTMs [6]. The former describes a generalization of LSTMs in the form of a tree-structured network topology, as opposed to traditional linear chains. The latter describes an implementation of a variable-length stack control structure for sequence-to-sequence neural net works; this allows us to utilize an efficient model to perform backpropagation and training, leading to better parsing performance.

The Sparsely-Gated Mixture of Experts implementation describes a trainable gating network consisting of thousands of feed-forward sub-networks [7]. This type of neural network is advantageous in performing conditional computation, which activates regions of the neural network gate (i.e., a subset of the sub-networks) to increase model capacity dramatically while keeping the proportion of computation increase low. This is attractive to our project to study conditional inference while maintaining low overhead.

Finally, we will also be using the Beam Search implementation, a heuristic search algorithm based on graph searching for promising nodes in data sets. The implementation we are pursuing is a seq2seq (sequence to sequence) model that generates output sequences greedily but maintained via beam search [8]. We would like to pursue this type of heuristic algorithm because of its memory requirement-reducing optimizations.

Resources Needed

The main software resources needed for this project consists of several machine learning tools to create neural networks. In the foreseeable future, we'll leverage PyTorch, a open-source machine learning framework, to construct and train our neural networks on our benchmarks. PyTorch will allow us to directly run neural network workloads on the GPU relatively seamlessly. As for other environments and software, at some point we will likely be using GPGPU-Sim, a cycle-accurate simulator for GPGPU workloads, to run our workloads for performance analysis. We will also need a set of GPGPU workloads to evaluate the performance of each of the neural network implementations.

If needed, Professor Mowry and Pratik will grant me access to compute resources on AFS/resources specifically to the Parallel Data Lab. Unless otherwise granted special hardware/compute resources (i.e., dedicated GPUs) at a later point this semester, I will be using my MacBook Pro to work on the aforementioned neural network worloads.

References

[1] Pekhimenko, Gennady et al. “A case for toggle-aware compression for GPU systems.” 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA) (2016): 188-200.

[2] Nandita Vijaykumar, Gennady Pekhimenko, Adwait Jog, Abhishek Bhowmick, Rachata Ausavarungnirun, Chita Das, Mahmut Kandemir, Todd C. Mowry, and Onur Mutlu. 2015. A case for core-assisted bottleneck acceleration in GPUs: enabling flexible data compression with assist warps. SIGARCH Comput. Archit. News 43, 3 (June 2015), 41-53.

[3] Crankshaw, Daniel, et al. "InferLine: ML Inference Pipeline Composition Framework." arXiv preprint arXiv:1812.01776 (2018).

[4] Connor Holmes, Daniel Mawhirter, Yuxiong He, Feng Yan, and Bo Wu. 2019. GRNN: Low-Latency and Scalable RNN Inference on GPUs. In Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19). ACM, New York, NY, USA, Article 41, 16 pages.

[5] Tai, Kai Sheng et al. “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks.” ACL (2015).

[6] Dyer, Chris et al. “Transition-Based Dependency Parsing with Stack Long Short-Term Memory.” ACL (2015).

[7] Shazeer, Noam et al. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.” ArXiv abs/1701.06538 (2017): n. Pag.

[8] Lee, Ceshine. “Implementing Beam Search - Part 1.” Medium, The Artificial Impostor, 17 July 2019, https://medium.com/the-artificial-impostor/implementing-beam-search-part-1-4f53482daabe.