Analysing Effect of Multiple Operations on a GPU for Deep Learning Training

GROUP - 01
Aditi Singh (singh273@wisc.edu)
Olesia Elfimova (oelfimova@wisc.edu)
Shubhankit Rathore (srathore2@wisc.edu)

1 Introduction

Owing to the common knowledge that the compute and communication cost of Deep Learning (DL) training comes pre-dominantly from auxiliary tasks, the majority of work around speeding-up and increasing throughout for DL training use the idea of overlapping compute operation with storage/network operations in these tasks. In many cases such optimizations might be memory and/or time efficient when applied individually. However, when trying to overlap multiple of these things, at some point the benefits from overlap are not apparent anymore. The goal of this project is to investigate at what point this happens and how is this limit for different hardware configurations.

Example of a library that speeds up DL training due to overlap of compute operation is PyTorch's Distribute Data Parallel (DDP) [3]. It overlaps gradient synchronization communications that take place during the backward pass with the backward computation. Even though DDP speeds up training, the process is still expensive and requires fault-tolerance mechanism. Common solution is to use checkpoints in order to resume the process where it was left off in case it was interrupted. With this idea in mind there was recently created CheckFreq [4] - a pluggable module for PyTorch. This framework solves the problem of checkpoint stalls - the time when GPU is idle, waiting for the checkpoint to complete - by introducing a DNN-aware two-phase checkpointing mechanism where each phase is pipelined with computation.

Another popular library for DL training is TensorFlow (TF). Like PyTorch, it also provides provides an API for distributed training called tf.distribute.Strategy [5]. Besides that, in order to reduce training time TF offers a powerful optimization - data prefetch. The prefetching function in tf.data overlaps the data pre-processing and the model training [1]. Data pre-processing runs one step ahead of the training. Moreover, TF's data API has a function experimental.dataset_prefetch_to_device() which can be used to specify that the parallel prefetch operation should stage the data directly into GPU memory.

A recent work in pipelining, PipeDream [2] speeds-up DNN training by combining intra-batch parallelism with inter-batch parallelism. Pipeline-parallel computation involves partitioning the layers of a DNN model into multiple stages. Each stage constitutes of a consecutive set of layers of the model. Stages are mapped one-to-one to GPUs that perform the forward pass (and backward pass) for all layers in that stage. PipeDream also automates the process of devising the optimal parallelism strategy for individual models. PipeDream is implemented in Python using PyTorch but is extendable to TF.

In this project we will investigate how applying distributed computation on a GPU, together with optimizations which use the idea of compute overlap, affects application. We will explore training efficiency by looking at multiple different combinations, e.g. DDP and CheckFreq, TF distributed mode and prefetch, and PipeDream with TF's and PyTorch's optimizations.

2 Related Work

There are multiple efforts on optimizing each of auxiliary ML tasks such as data loading, distributed computation, or checkpointing but few of these studies have attempted to integrate several optimizations into a single training system. For instance, authors of CheckFreq in their paper devote a small segment to discuss how framework works with the DDP mode [4]. While they mention that CheckFreq works with DDP mode where only one GPU per node is responsible for checkpointing and show results for single- and multi-GPU training, they never investigate the performance of a system in multi-node settings. Similarly, the performance of TF's distributed library together with data prefetch is largely unexplored. There has been a research [1] showing that the TensorFlow prefetcher is a key factor for the input pipeline performance in TensorFlow. In their mini-application, investigators found a complete overlap of per-batch compu-

tation and I/O pipelines leading to a minimal impact of I/O performance on the overall performance of the mini-application. However, it is also mentioned in the paper that the focus was on single-node I/O performance. The performance of TensorFlow I/O using distributed systems is listed as a topic for future work.

3 Our Planned Approach

Our extended goal would be to devise a strategy wherein DL training can reap the benefits of several optimizations simultaneously and achieve superior performance. We plan on incorporating optimizations at different levels of training in one large customizable system. The system will have knobs for modulating how aggressively each optimization is applied. The user can decide this depending on their requirements. For example, if fault-tolerance is a more important goal than fast data pre-processing, CheckFreq can be tuned to checkpoint parameters more frequently and data pre-processing can be done at coarse granularity. Finding an optimal balance between frameworks for a general training scenario would be one of the outcomes. We plan to take a heuristic based approach to analyze the impact of each optimization framework in conjunction with others.

4 Anticipated Results

Our hypothesis is that a naïve application of all the optimizations to DL training simultaneously will give worse performance than using one of those optimizations individually, especially on small models and datasets. Communication links will be oversubscribed and overhead will dominate over any benefits these optimizations could provide. As we modify the granularity of frameworks, we anticipate a change in congestion and therefore a diminishing return. We expect our approach to show better results than an all-included solution mainly because there should be less contention and fight for communication and computation resources.

5 Timeline and Evaluation Plan

For evaluating our project we plan to do the following:

- Run DL training applications for different combinations, for example:
    1. CheckFreq and PyTorch's DDP
    2. TF's distributed mode and TF's prefetch
    3. PipeDream and CheckFreq
    4. PipeDream and TF's prefetch

    Observe performance on a distributed cluster for different datasets, for example:

    1. CIFAR on ResNet18
    2. CIFAR on ResNet50
    3. ImageNet on ResNet18
    4. ImageNet on ResNet50
    5. BERT on IMDB dataset

    The combinations will be chosen to include datasets and models of different sizes.

- We will conduct these experiments with different hardware configurations of RAM, GPU, network bandwidth etc. We plan on using NVIDIA V100 GPUs accessible through the Google Cloud Platform (GCP). For VMs that have V100 GPUs attached, GCP can provide a maximum bandwidth of up to 100 Gbps while the memory can vary between 1 GB and 624 GB.

Timeline:

- Nov. 9 - Have multiple versions of DL applications with combined optimizations ready.
- Nov. 23 - Conduct preliminary analysis and record performance of applications.
- Nov. 30 - Have a rough report prepared with analyses of different scenarios.
- Dec. 14 - Finalize the poster.
- Dec. 20 - Finalize the report.

References

[1] S. W. D. Chien, S. Markidis, C. P. Sishtla, L. Santos, P. Herman, S. Narasimhamurthy, and E. Laure. Characterizing deep-learning i/o workloads in tensorflow. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage Data Intensive Scalable Computing Systems (PDSW-DISCS)*, pages 54–63, 2018.

[2] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training, 2018.

[3] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[4] J. Mohan, A. Phanishayee, and V. Chidambaram. Checkfreq: Frequent, fine-grained DNN checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 203–216. USENIX Association, Feb. 2021.

[5] B. Pang, E. Nijkamp, and Y. N. Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020.