Co-location of ML Jobs in GPU Clusters

by

Iman Tabrizian

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Co-location of ML Jobs in GPU Clusters

Iman Tabrizian

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2020

I would like to thank my parents and my dear brother for their amazing support. Without their support, I would not been able to do anything. I want to sincerely thank my advisor Professor Alberto Leon-Garcia for giving me the ability to work independently. His insightful comments have greatly impacted this project. I would like to thank Kaveh Mahdaviani and Professor Bianca Schroeder for their time and insightful discussions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Learning has transformed the world. Nowadays, there are various machine learning models used in production systems. Models that provide image classification, movie recommendations, and even photography. These advancements have been made possible thanks to the specialized hardwares such as GPUs, TPUs[4], and FPGAs. The process of training these models is a trial-and-error approach combined with intuition. It requires tuning a large number of hyperparameters to achieve the desired accuracy. To achieve the best accuracy in the shortest amount of time, usually many similar jobs are dispatched that only change a single hyperparameter. Research institutes and companies have access to compute clusters that provide 100s to 1000s of GPUs. It will require a scheduler to map the jobs to the resources. The users of the jobs need to specify the resource requirements of the jobs too. The scheduler will take in these specifications and map it to the the available resources. GPUs with their unique characteristics add a new dimension to this problem. The main problem with the GPUs is that they were not originally designed for multi-tasking. Unlike conventional resources like CPUs and main memory, that have hardware support for resource sharing, GPUs were originally designed for single process use. The assumption was that the application is able to effectively use all the resources on a GPU and adding another application to the GPU will hinder the performance. However, starting from the Volta[2] architecture GPUs now include hardware support for execution of multiple processes together.

In this dissertation, we explore the effect of co-locating multiple deep learning training jobs together. We devise metrics that guide the scheduler on which workloads can benefit from co-location and which workloads should not be placed together as it will decrease the training speed for these jobs. Additionally, we study the potential reasons behind the incompatibility of the workloads and how they can be used to predict the incompatible jobs.

The next chapter gives an overview of the scheduling algorithms in other domains and an introduction on how GPUs are used for general processing. Chapter 2 discusses the motivation for co-location of the DL jobs. Chapter 3 discusses the design of our proposed scheduler that employs co-location to improve the job completion time (JCT). Chapter 4 provides the experimental results for this design and how it compares to the schedulers that do not employ co-location. Finally, Chapter 5 concludes the thesis and provides future directions on how this work can be extended.

Chapter 2

Motivation and Related Work

2.1 GPUs

GPUs were among the first accelerators to be introduced alongside with CPUs. GPUs are throughput optimized in contrast to the CPUs which are latency optimized. They consist of many cheap cores that can perform large number of operations in parallel. They also have a large memory bandwidth that helps them bring data into these cores in efficiently.

Figure 2.1 shows a simple illustration of the GPU architecture. GPUs consist of several *Streaming Multiprocessors (SMs)*. SMs are the processing units in GPUs. Each

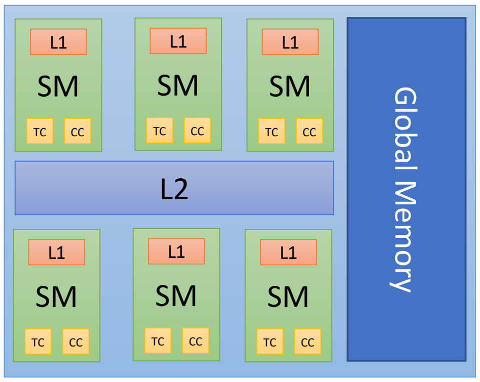

Figure 2.1: GPU Architecture

of the SMs contains various cores designed for operations on different data types. For example, V100 GPU contains 84 SMs where each of them has 64 FP32 cores, 64 INT32 cores, 32 FP64 cores, and 8 tensor cores[2]. In fig. 2.1, *CC* refers to the CUDA cores which consists of all the cores present in each SM except the tensor cores. Tensor Cores are abbreviated using *TC*. They provide accelerated performance for the reduced precision operations which are present in the Deep Learning workloads. They were introduced in the Volta[2] microarchitecture in 2017.

2.1.1 GPU Memory and Compute Capabilities

GPU memory and compute capabilities have increased by a tremendous amount over time.

2.1.2 CUDA

CUDA is a set of extensions to C/C++ to enable easier application development in GPUs. CUDA also introduces a set of language abstractions that make it easier to think about GPU programs. GPU accelerated programs use many threads to perform the computation. CUDA groups the threads into *thread blocks* and *grid blocks*. A *thread block* is a group of threads which are guaranteed to be running on the same SM. A *grid block* is a group of thread blocks which contain all the processing necessary for the computation of a given *kernel*. A *kernel* is a function that runs on a GPU. Both thread blocks and grid blocks can be represented using three dimensions. Figure 2.3 shows the difference between a thread block, grid block, and a thread.

2.1.3 Thread Block Scheduling

There are various constraints that limit the scheduling of thread blocks into the SMs. Amount of the registers, shared memory, and number of threads inside a thread block

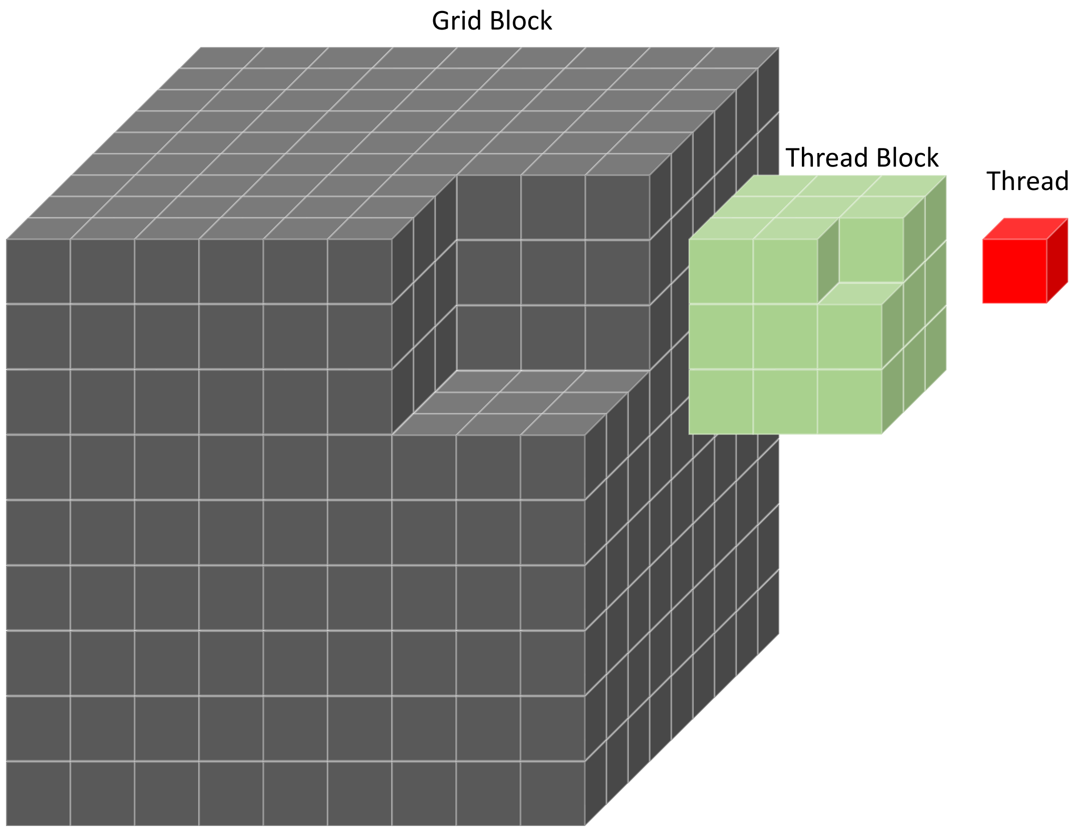

Figure 2.2: Grid Block vs Thread Block vs Thread

are among the factors that limit the number of blocks that can be scheduled into a single SM. Since there is a limited amount of these resources available in each SM, the number of thread blocks will be limited to the available resources. Apart from that, different GPU architectures have hard limits on the number of thread blocks and threads that can be scheduled on a given SM. All these factors lead to a metric called *Theoretical Occupancy*[1] of a kernel. Theoretical Occupancy is a metric in percent which determines the percentage of active warps in comparison with the total warps that could be scheduled on a given GPU. There is another concept called *Achieved Occupancy*. Achieved Occupancy measures the scheduled number of warps when the kernel is actually running on the GPU. This can be different from the Theoretical Occupancy because a given thread in warp might we stalled on a memory load and is not yet ready to be scheduled. If there is not enough warps in flight ready to be scheduled instead of the stalled thread block, the achieved occupancy will be lower than the Theoretical Occupancy. Theoretical Occupancy serves as the upper bound for the Achieved Occupancy.

2.2 Related Work

In the area of GPU schedulers for deep learning workloads there are various related works to the work presented in this thesis. Optimus [6] is one of the earlier works in this area. The main goal of this work is which job should be given more workers so that the total job completion time for a set of jobs is minimized. It is assumed that the jobs use the Parameter Server [5] architecture. They learn the convergence curve for a given job to predict the number of required iterations until the completion. Using this information, they can measure how long the job is going to last. Then, they create a heuristic algorithm that increases the number of workers or parameter servers for a job that gains more speed up compared to the other jobs. This work is complementary to our work. We can augment their techniques with our co-location algorithms to better

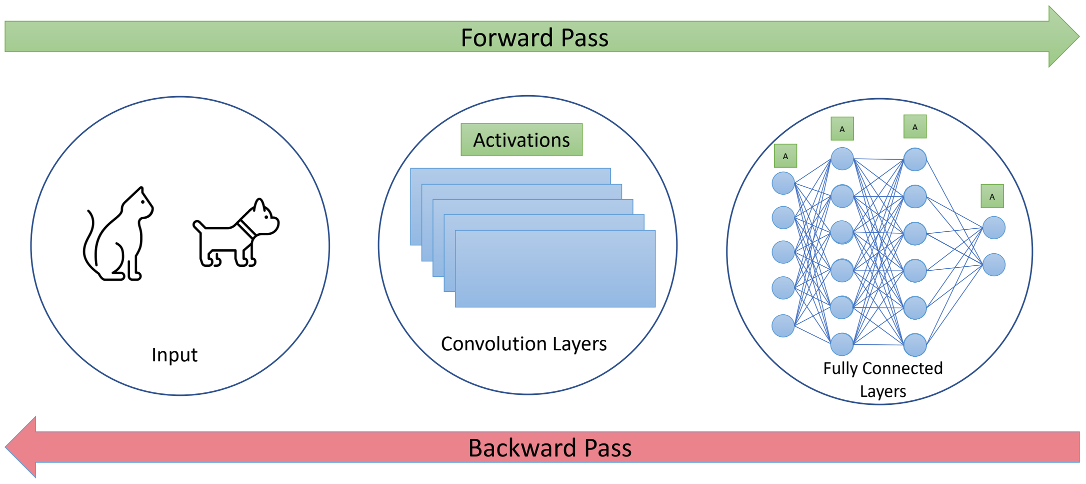

Figure 2.3: Memory Allocation in Forward and Backward Pass

utilize the GPUs.

Tiresias [3] presented a more realistic scheduling algorithm. Tiresias is able to use the historical data of the jobs to minimize the job completion time. They do not adaptively change the number of workers for a given job. This is a more realistic approach since increasing the number of workers affects the accuracy and may require retuning all the hyperparameters. They do not consider the packing of multiple jobs on the same GPU.

Gandiva [7] introduced a fast context switch mechanism to avoid starvation of the jobs in deep learning clusters. They observed that the GPU memory usage of a job is not constant during the training and is minimum between the iterations. While this is not true for PyTorch and Tensorflow frameworks, some frameworks like MXNet deallocate the memory when it is no longer needed.

Chapter 3

Scheduler Design

We designed SC (Scheduling using Co-location) to utilize the excess amount of compute available in GPUs to reduce the queuing time and job completion time for the deep learning training jobs in GPU clusters. In Section 3.1, we show empirical results on how various jobs respond to co-location. In Section 3.2, we discuss the underlying reason on why some jobs are not compatible with each other. In Section 3.3 we provide the general design and architecture for our scheduling algorithm.

3.1 Co-location Effect

3.2 Why Some Jobs are not Compatible with each other?

3.3 Scheduling algorithm

Chapter 4

Evaluation

Chapter 5

Future Steps

Bibliography

[1] Cuda warps and occupancy. `https://on-demand.gputechconf.com/gtc-express/2011/presentations/cuda_webinars_WarpsAndOccupancy.pdf`. (Accessed on 11/07/2020).

[2] Volta architecutre. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`. (Accessed on 10/20/2020).

[3] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, Boston, MA, February 2019. USENIX Association.

[4] Norm Jouppi. Google supercharges machine learning tasks with tpu custom chip. *Google Blog, May*, 18:1, 2016.

[5] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[6] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceed-*

ings of the Thirteenth EuroSys Conference, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, Carlsbad, CA, October 2018. USENIX Association.