Pipeline Parallel DNN Training Techniques

References -

PipeDream: Fast and Efficient Pipeline Parallel DNN Training (2018) Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, Phil Gibbons

Memory-Efficient Pipeline-Parallel DNN Training (2021) Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, Matei Zaharia

Introduction

In today's world, the rate at which model sizes are increasing is much faster than the rate of increase of compute power. The limitations of underlying hardware often cause a bottleneck in the tasks these massive models could be capable of performing. Existing methods like Data Parallelism and Model Parallelism help in speeding up the training process to quite some extent, but each has their own drawbacks. Hence there is a need for efficiently training large scale models, in order to utilize the maximum capacities of hardware and benefit from more number of parameters.

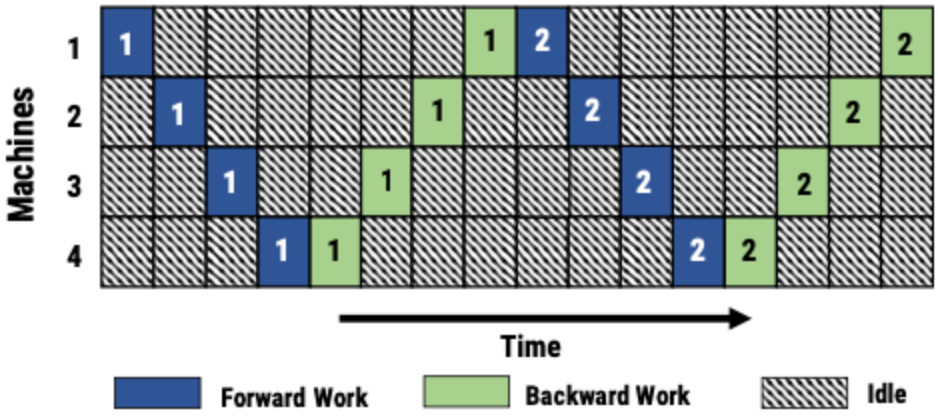
The above mentioned papers describe a pipeline parallel implementation of training large scale deep learning models. Lets discuss some key contributions from each of these techniques -

PipeDream

Primary Contributions of PipeDream paper -

1. Gives a parallelization approach for DNN training that addresses communication bottlenecks by combining model parallelism with pipelining along with data parallelism

The authors begin by explaining the drawbacks of data parallelism and model parallelism. As model size increases, per aggregation communication increases which is a threat to the efficient working of data parallel training. Whereas in model parallelism, there is underutilization of compute resources, as only one worker is active at a time and there is no overlap between compute and communication times (see Figure 3 in paper).



PipeDream implements pipeline parallelism scheme, abbreviated as PP. Similar to model paral- lelism, PP partitions the model and assigns consecutive set of layers (called stages in the paper) to each worker machine. But, unlike traditional model parallelism, it also aggressively pipelines mini-batch processing, with different workers processing different inputs at any instant of time, thus implementing data parallelism in pipeline with model parallelism

2. It defines challenges in this approach and describes its solutions for them -
    - Optimal partitioning of work across machines - using a profiler and partitioning algorithm

PipeDream uses a profiler, which computes the computation time of a forward pass and backward pass for each layer. This profiler runs on a single machine first using 1000

mini-batches and estimates the computation time for each layer requirements. This information is input to a novel partitioning algorithm that outputs the partitioning of each layer into stages, along with its replication factor and optimal number of mini-batches to keep the pipeline busy in steady state. The goal of this algorithm is to minimize training time and balance the work load between each stage. (Refer Figure 6 of the paper)

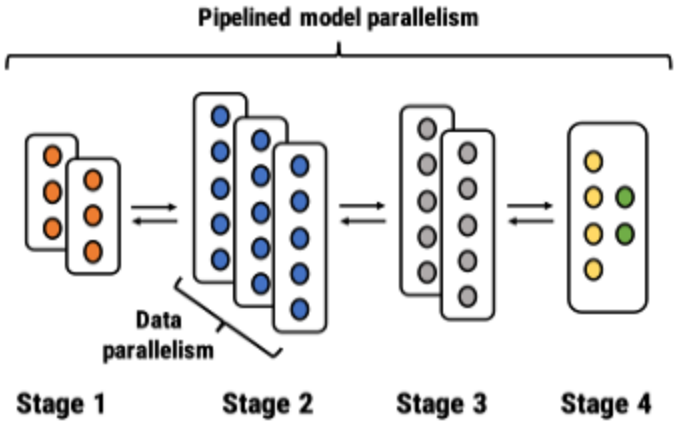
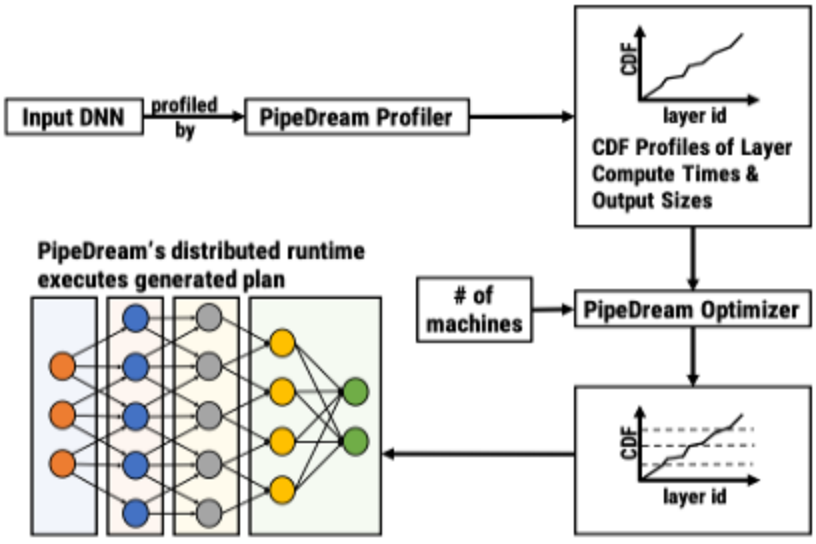


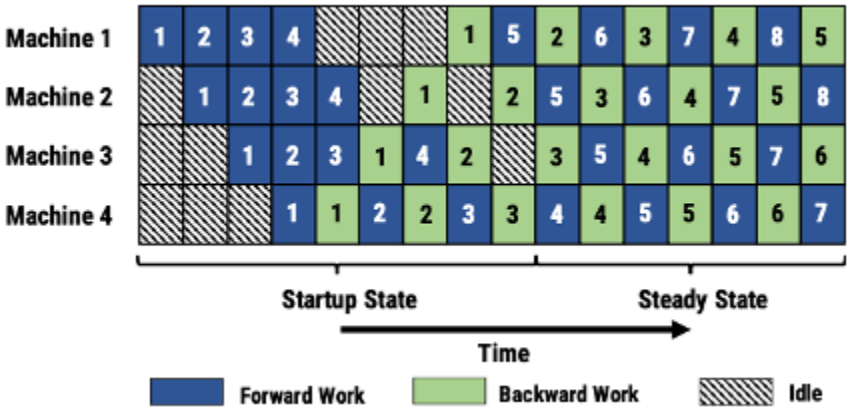
Figure 6: Pipeline Parallel training in PipeDream combines pipelining, model- and data-parallel training.



The benefit of having a stage is that you only need to communicate the parameters of one layer to another stage, instead of the parameters of all the layers as in data parallel training. During training, PipeDream asynchronously performs fwd and bckwd pass for each input minibatch, it overlaps computation and communication

- Scheduling of computation to maximize throughput while ensuring forward progress in the learning task - using 1-F-1-B mechanism and deterministic round robin load balancing

During the start of the training process, the optimal number of mini-batches (given by the partitioning algorithm) is fed into the pipeline. However in the steady state, each stage alternately performs a forward and backward pass for a mini-batch (called 1-F-1-B). This ensures no worker is idle during steady state. Round robin load balancing is used to spread the work across each replica of a stage.



In the figure showing steady state, if we observe stage 1 (machine 1), the forward pass for minibatch 5 is performed after the updates from minibatch 1 are computed, whereas the backward pass for minibatch 5 is performed after updates from minibatches 2, 3, and 4 are computed.

- Ensuring that learning is effective in the face of asynchrony introduced by pipelining - using weight stashing and vertical syncing

To prevent discrepancy in weight versions and hence model convergence, PipeDream maintains different versions of weights for each active mini-batch. So it is able to use the same version to compute weight gradient in the backward pass. It ensures that same weight version is used for the forward and backward pass against a mini-batch. The stashed weights are propagated across all stages, thus ensuring vertical syncing of weight versions. This leads to efficient training even in asynchronous computation.

- Implements the pipeline parallel training system PipeDream

It is implemented as a C++ library, which can also be extended to Tensorflow and other ML frameworks. The library manages the parameters and intermediate data. PipeDream provides the ML worker thread (Caffe) pointers to GPU memory containing layer input data, parameters, and buffers for recording layer outputs and parameter updates, manages buffer pools, and handles communication between and within workers.

- Measures performance of PipeDream for cases when communication bottleneck limits data parallel training

PipeDream is able to achieve faster training that data parallel approaches for popular DNN models trained on on ILSVRC12 dataset -

1.45x faster for Inceptionv3

5.12x faster for VGG16

1.21x faster for Resnet-50

6.76x faster for AlexNet

Drawbacks of PipeDream

Since it maintains multiple weight versions for each mini-batch, it has a high memory consumption, which makes it infeasible to train larger DNN models, having parameters of the order of billions

PipeDream-2BW

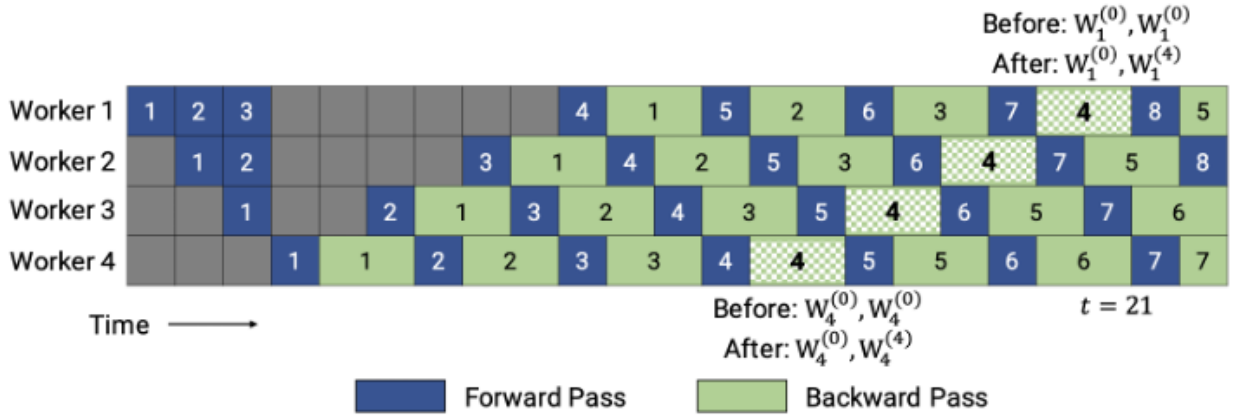
Primary Contributions of PipeDream-2BW paper -

1. Double-Buffered Weight Updates 2BW technique and flush mechanisms (PipeDream-Flush)

The 2BW along with the 1-F-1-B technique uses less memory footprint than PipeDream and GPipe (another pipeline parallel training architecture that periodically flushes inputs to update weights). PipeDream-2BW generates a new weight version after every m

micro-batches are processed. Gradients are coalesced over micro-batches and then applied over a batch. This limits the number of weight versions required for a batch to be atmost 2 for all the mini-batches active in the pipeline. (Refer Figure 2 of paper)

A micro-batch can use only max(⌊(k − 1)/m⌋ − 1, 0) versions of weights during the entire forward and backward pass, where m is the number of micro-batches in a batch. It also uses activation recomputation, meaning only the activations of inputs are stored in memory as opposed to all intermediate activations as in PipeDream, which enables large per-GPU micro-batch size and training of larger models.

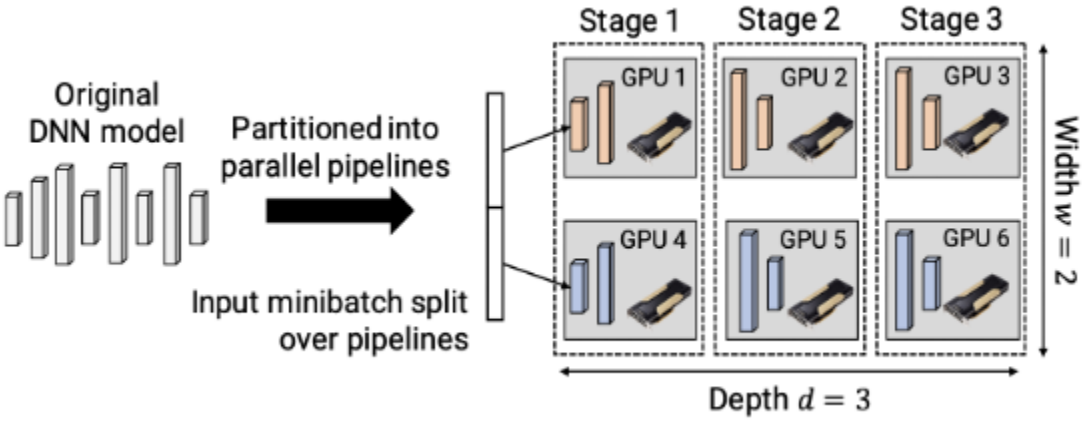


In this example, you only need version 0 until backward pass of batch 4 is complete, Once you have batch 8 in worker 1, you can discard version 0. Hence each worker can discard W (0) once they are done processing the backward pass for input 8 since all subsequent inputs use a higher and later weight version for both their forward and backward passes.

2. Planner

It implements a planner algorithm which tries to find the optimal width and depth of layer partitioning that achieves highest throughput and largest per-GPU micro-batch size that fits into memory. It determines the size of each model partition, batch size, and whether to use memory- saving optimizations like activation re-computation by optimizing for a cost function for these 3 executions -

- ☐ Model and data parallelism without pipelining
- ☐ Hybrid parallelism with pipelining without activation recomputation
- ☐ Hybrid parallelism with pipelining and recomputation



Partitioning using planner algorithm

Results and Observations-

- PipeDream-2BW results is 20X faster training of GPT (~3.8 billion params) vs the model parallel approach of Megatron
- PipeDream-2BW results in 3.2X faster training of GPT vs pipeline parallel approach of GPipe
- As the global batch size with gradient accumulation B increases, the throughput of model increases; due to less communication across the stage replicas
- As the pipeline depth d increases, since the number of bytes communicated remains constant, the throughput decreases
- As pipeline depth d increases, it increases maximum micro-batch size for each GPU. This results in more computation and throughput
- Larger and larger models can be trained as the pipeline depth d is increased

To summarize -

These papers introduce pipeline parallel DNN training architectures that try to overcome the drawback of data parallelism and model parallelism scenarios.

PipeDream is a training system that automatically partitions DNN layers between workers to optimize work and minimize communication, keep updated versions of model parameters and enables scheduling of forward and backward passes of different inputs to minimize training time.

PipeDream-2BW is a system for efficient pipeline-parallel DNN training that achieves high throughput and low memory consumption on the PipeDream architecture by using an improved pipelining and weight gradient coalescing strategy with double-buffered weight updates.