**Last 4 digits of ID:2687**

**A brief description of the main point of the paper.**

Deep Neural Networks have been widely used and have achieved remarkable achievements. However, DNNs with good performance often have extremely complex internal structures and are time-consuming for training. These shortcoming makes the training of the model so computationally expensive, which needs parallel execution on multiple accelerators. However, intra-batches parallelization can suffer from high communication overheads as workload grows. And also DNN training is bi-directional.

Based on these issues, the author first introduces a combination of intra-batch parallelism and inter-batch parallelization, named PipeDream,

Second , stress out  PipeDream uses an algorithm called 1F1B to keep hardware efficiency, And also uses different versions of model weights to maintain statistical efficiency. PipeDream limits the number of “in-pipeline” minibatches to the minimum needed to keep the pipeline full, reducing memory overhead.

Third, PipeDream extends 1F1B to incorporate round-robin scheduling across data-parallel stages.The combined scheduling algorithm produces a static schedule of operators that each worker runs repeatedly, keeping utilization high across all workers.

At last, some experiments are implemented to confirm the training time benefits and accuracy of PipeDream’s pipeline parallelism.

**A description of the State of the Art (SOTA), as described by the authors, prior to the writing of the paper.**

A DNN model is complex that need to be partition into different layers. When parallelaing DNN training, these layers need to partition into states to match the number of workers in our model. Based on this issue, the author review two partition parallelization approaches, intra-batch and inter batch to describe their essence.

Intra-batch parallelization means split single iteration of training across different available workers, and is widely used in parallelization.

Data parallelizaiong is a common one where each worker matintain a piece of model weights and input, then thra in the data independently, with the help of collective communication and parameter servers for communication. In contrast with the wait gradient being send as soon as avaliable, BSP requires every worker to wait for gradients from other, So there should be some communication stall in large scale input. Accurate experiment on state of art GPU platforms show some hints about the nature of Data parallelizaiong:

1)High overhead in still exist in latesd GPU platform in some ,Performance difference by models.

2)Slower interlink is the main limitation of application performance across servers. The same number bytes need to transmit on both high and low bandwidth channels.

3) communication overheads increase fo r all modes inspite of training with newest GPU , when number of worker increase.

4) GPU speed increase with the communication overhead.

ASP allowing each worker to proceed with the next input minibatch before receiving the gradients from the previous minibatch. It increases hardware efficiency, on the other hand reduce statistical efficiency. quantizing gradients works for specific model but lacking of generality. Recent works believes large minibatch is more effective by changing parameters less frequently.

Model Parallelism. is an approach where each worker evaluating and performing updates for only a subset of the model’s parameters for all inputs. vanilla model parallelism suffrers from two limitations,first, under-utilization of compute resources caused by MP, second burden of partitioning a model across multiple GPUs is left to the programmer, resulting in point solutions.

Hybrid Intra-batch Parallelism splits a single iteration of the optimization algorithm among multiple dimensions. OWT split the AlexNet model by hand, OWT does not include pipeling.

FlexFlow splits a single iteration along samples, and describes an algorithm of plitting automately.

Inter-batch Parallelism

GPipe reads partitioned model as input, then splits a minibatch into m microbatches, and performs forward passes followed by backward passes for these m microbatches. Gpipe have high memory efficiency but may suffer reduced hardware efficiency.

**A listing of the contributions made by this paper (according to the authors). How did they advance the SOTA?**

Aside from the many parallelision method such as data parallelism, Model parallelism and hybrid intra-batch parallelism, the authors propose pipeline parallelism to improve the efficiency parallel So training. It combines intra-batch parallelism with inter-batch parallelism. PipeDream first divided layers into stages, and map the stages to multiple workers(GPU). By injecting many minibatch into the input. PipeDream can perform the forward and backward pass automatically in a pipeline. When forward pass completes in every stage, it will send activation to the next stage, at the same time perform the next minibatch. PipeDream perform the Back ground communication and work training at simultaneously, which increase the efficiency. PipeDream transmits subset of activations and gradients to only one worker, which decrease the communication redudence.

However, this parallelision model also bring some challenge besides it advantages.

How to partition the work to layers and stages is viatal, because the efficiency will be limited by the slowest throughput rate state(have bubble in fast stage, low efficiency), excessive communication between stages. Handle these problems and reach hardware efficiency and model efficiency.

A picture containing clock

Description automatically generated

PipeDream build a balanced pipeline to minimize the communication, and outputs an algorithm to partition layers into stages.

First , a profiler is carried out, within few minutes, it profles 1000 minibatches on a single worker, estimating the total communication time for weight synchronization which is the same with data parallelizm with n workers. As Fig 1 shows, output as well as the activation and parameter size.

Second, with the outputs the optimizer further uses partition algorithm to partition layers to stages, decide replicate workers for each stage, and get optimal number of minibatchs to keep pipeline busy.

But PipeDream need the topology of workers to be highly hirachey, where the sub-pipeline throughput compose the whole pipeline throughput. The algorithm consider a single state(pipelime) running time by compairing the shortest time and data-parallel communication between layer and devided by m, the result of the max on becomes the optimal time for training and communication. (also considered replica of stage across workers)Then break the pipeline into layers and calculate the sub-pipeline time using optimal sub-problem property. With some initial value of level 0. This mode will calculate out the running time for each partition case, and choose the best case of them as the optimal partition method.

Challenge 2 is how to schedule the work.

PipeDream training data in bi-direction, minibatch is either on forward or backward pipeline. So the worker needs to decide whether perform to forward to downstream stage or backward to upstream stage. And what to do in duplicated stages. The authors came up with a one-forward-one-backward 1F1B algorithm which each stage alternates between performing forward pass for a minibatch and then a backward pass for an earlier minibatch. This ensure the pipeline in a balanced manner. At first, the input state must accept enough minibatches to keep the pipeline full for efficiency, and calculate the stage replica per input(show stages replica upon many workers) to keep pipeline full. When confronted with state replica, data are running in a data parallelism mode, they use deterministic round-robin load balancing based on a minibatch identifier to spread work across the replicas. stages across the replicas. Such deterministic load- balancing ensures that the backward pass for a minibatch is performed on the machine responsible for the mini- batch’s forward pass[2]. Both the 1F1B scheduling policy for stages in a pipeline and the round-robin scheduling policy for load balancing across replicated stages are static policies. So it should work out and easy employed.

In previous parallelism each stage’s forward pass may not use the same weight and parameters with its backward pass as go forth and back the pipeline, this may occur the invalid gradients and prevent convergence of model. PipeDream use weight stashing to maintain different version of weights, within the same state . the same version of parameters and weights will be used for both forward and backward pass of a given minibatch. The authors also propose optional VerticalSync to help keep PipeDream minibatch parameters consistency across stage, instead use the latest weight , the forward pass for minibatch use stashed weight fly-in the pipelime. This method ensures the same parameters are used to compute gradients across the diffetent workers. In ML, staleness affect the conversionce. higher staleness lowers synchronization requirement and also slow down the progress. In PipeDream, everaged gradient across minibatches are used for learning with weight stashing. PipeDream has similar memory overhead with data parallelism, and can further reduce memory by efficient encoding or compression of intermediate data , gradient aggregation and trading computation time for activation-stash.

**A description of what the authors did to prove their result (simulation? implementation? Measurement? something else?)**

**Your analysis of the paper, including:**

* **Do you find the work to be valuable? Why or why not?**
* **Can you relate any part of this work to other papers that we have read for this class?**
* **Do you see issues with the paper, or confusions or questions that were not answered?**
* **What do you think researchers (including but not limited to the paper’s authors) should do next to further this line of inquiry?**

**Reference**

[1] PipeDream: Generalized Pipeline Parallelism for DNN Training.Deepak Narayanan, Aaron Harlap, et al.

[2] PipeDream: Fast and Efficient Pipeline Parallel DNN Training. Aaron Harlap, Deepak Narayanan, et al.

# [3] Pytorch tutorials .<https://pytorch.org/tutorials/>

# [4] Everything You Need to Know about Gradient Descent Applied to Neural Networks. <https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14>.

[5] Toward understanding the impact of staleness in distributed machine learning. Wei Dai, Yi Zhou, et al.