Scaling the training of large networks with limited hardware

Rushabh Musthyala
Charvi Gupta

rm6416@nyu.edu
cg4177@nyu.edu

Problem Motivation

- The rate at which model sizes are increasing is far greater than the rate at which compute power is increasing
  - The winner of ImageNet challenge in 2014 had 4M parameters, the winner in 2017 had 146M parameters - 36x increase there but GPU capacities increased only 3x in the same time period
- The types of tasks that can be performed become bottlenecked by hardware
- Current model/data sharing methods are task specific and do not scale well
- Drawbacks of Data Parallelism - Unable to perform well on large models (due to increased per-aggregation communication)
- Drawbacks of Model Parallelism - Severe underutilization of compute resources (because only 1 worker is actively used at a time)
- Maintain high throughput and low memory footprint while doing pipeline parallelism

Background Work

1) **GPipe:** Efficient Training of Giant Neural Networks using Pipeline Parallelism - introduces a technique that merges model and data parallelism in a pipelined approach to maximize GPU utilization
2) **Hydra:** A System for Large Multi-Model Deep Learning - builds upon a model and pipeline parallelism approach optimized for training multiple models simultaneously

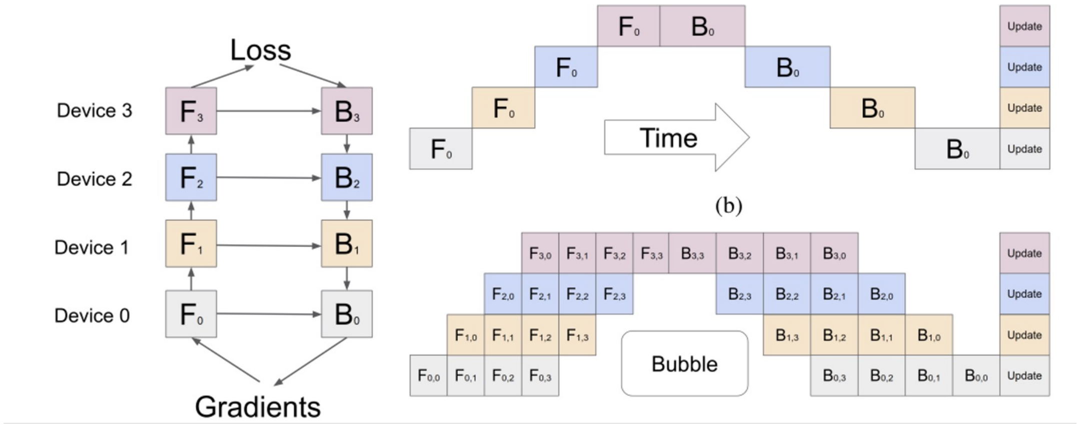
Background Work (cont.)

3) **PipeDream:** Fast and Efficient Pipeline Parallel DNN Training - 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

4) **PipeDream-2BW:** Memory-Efficient Pipeline-Parallel DNN Training - is a system for efficient pipeline-parallel DNN training that achieves high throughput and low memory consumption using an improved pipelining and weight gradient coalescing strategy with double-buffered weight updates

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

- GPipe is a pipeline parallelism library that **allows scaling any network** that can be expressed as a sequence of layers
- **Partitions a model across different accelerators**
  - Partitioning is done such that the variance in estimated cost between each partition is minimized
- **Splits mini batch into smaller micro-batches** (1 micro-batch per partition)
- **Pipelines the execution** of the partitions and the micro-batches they are working on

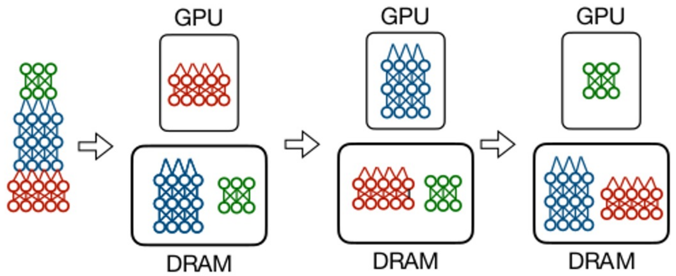

Loss
Device 3
Device 2
Device 1
Device 0
F_3 B_3
F_2 B_2
F_1 B_1
F_0 B_0
Gradients
Time
(b)
Update
Bubble
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$
$B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
$B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
$B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
$B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$

GPipe (cont.)

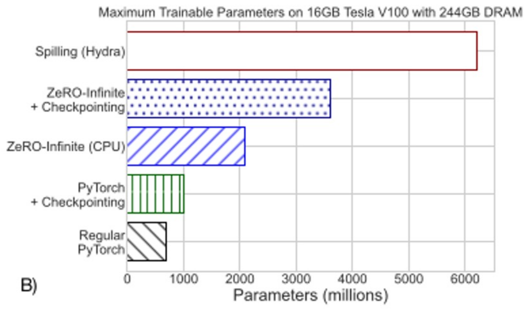
- Results and observations -
  - For AmoebaNet, was able to **train a model almost 4x bigger** using just 1 accelerator
  - When M >>> 4*K, **bubble overhead is almost negligible**
  - **Communication bandwidth is not as big a overhead** in this when compared to model parallelism
  - **Scaled up NMT model by 3x** on same hardware capacity
- Drawbacks -
  - Assumes a single layer can fit into the GPU
  - Micro-batch splitting makes things like batch norm harder
- Key takeaways -
  - **Efficiency**: Almost linear speedup with no. of devices
  - **Flexibility**: Can be used on any network that can be represented as a sequence of layers
  - **Reliability**: Since it uses synchronous SGD, it guarantees consistent training irrespective of number of partitions and micro batches

Hydra: A System for Large Multi-Model Deep Learning

- Most DL users don't just train one model in a vacuum but do it as part of a large multi-model execution scenario (choosing best hyperparameters) - task parallelism can help increase throughput
- SHARP is an approach that **combines model and task parallelism** which can run on only one GPU
- Model is **partitioned according to GPU memory**, breakpoints inserted whenever it goes OOM
- **Uses "spilled" execution scheme** which rewrites a model into shards and promotes/demotes them between GPU memory and DRAM



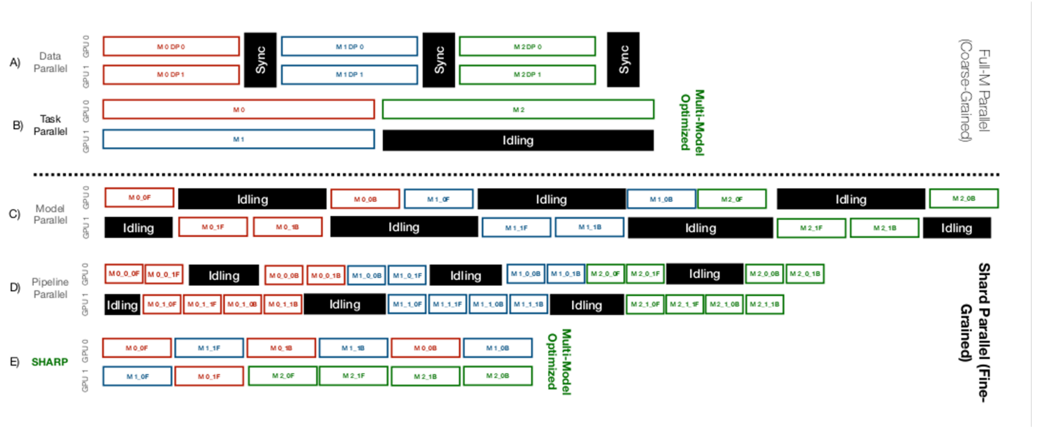
A)



B)

Hydra (cont.)

- Each shard is loaded into GPU twice (F&B) and in-shard intermediates are recomputed to save memory
- Communication is between GPU and CPU which has higher overhead -
  - **Double buffering is used** which overlaps communication with compute
- Results and Observations (fine tuning of GPT-2) -
  - Upto **90% lower runtime than GPipe**
  - Upto **2x less energy consumption**
  - **4x faster than naive model parallelism**



PipeDream: Fast and Efficient Pipeline Parallel DNN Training

- PipeDream uses *pipeline parallelism* (**PP**) - combines traditional data parallelism with model parallelism enhanced with pipelining
- Partition layers of DNN into *stages* - containing consecutive set of layers.
  - PP communicates only output data of one of the layers in each stage
  - Overlaps computation and communication
- **Automatic Partitioning** of Layers across Machines
  - using *Partitioning Algo* based on Dynamic Programming
  - minimizes training time
  - ensures each stage performs roughly same work
  - O(N^2M^2) where N is layers of DNN and M is machines
- Profile the model on a single machine using 1000 minibatches, and then run the Partitioning algorithm that groups layers into stages and determines replication factor across each stage

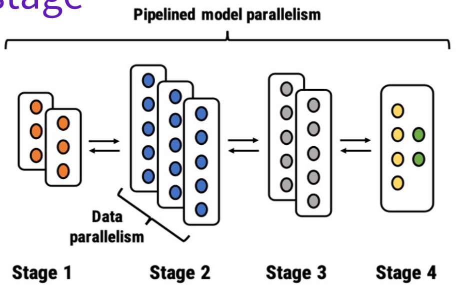
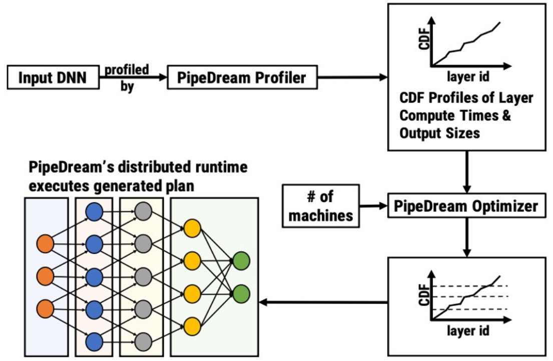
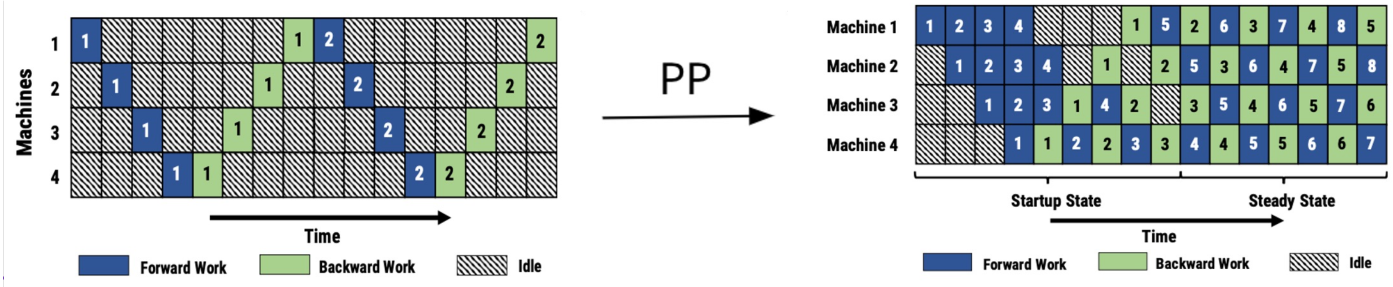


Figure 6: Pipeline Parallel training in PipeDream combines pipelining, model- and data-parallel training.


Input DNN
profiled by
PipeDream Profiler
CDF
layer id
CDF Profiles of Layer Compute Times & Output Sizes
# of machines
PipeDream Optimizer
PipeDream's distributed runtime executes generated plan
CDF
layer id

PipeDream (cont.)

- **Scheduling computation** - using *1-F-1-B* mechanism and deterministic round robin load balancing
- **Effective Learning** even in asynchrony -
  - *Weight Stashing* - maintain multiple weight versions
  - *Vertical Sync* - propagate input's associated weight across layers

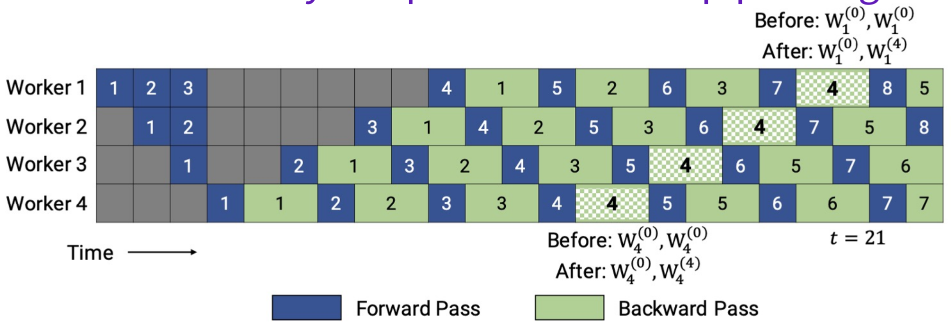


- Results
  - Faster training that Data Parallel approaches: **1.45x** for Inceptionv3, **5.12x** for VGG16, **1.21x** for Resnet-50 and **6.76x** for AlexNet on ILSVRC12 dataset

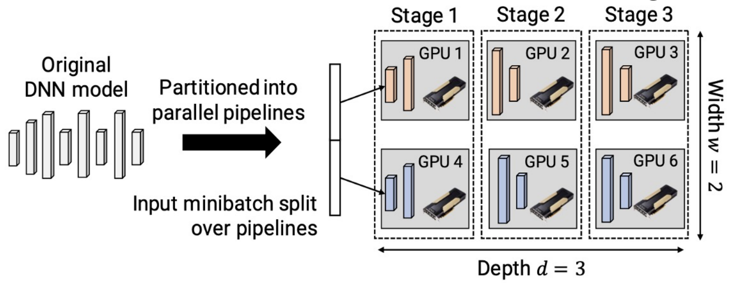
- Drawbacks - Can't use for bigger models (~ billions of parameters) as it uses higher memory footprint

PipeDream-2BW: Memory-Efficient Pipeline-Parallel DNN Training

- **2BW** - Double-Buffered Weight Updates - At most 2 weight versions needed than *d*
    - Gradient computed over microbatches then “coalesced”
    - Weight version a microbatch k uses = $max(\lfloor (k-1)/m \rfloor - 1, 0)$
- **PipeDream-Flush** - maintains single weight version and uses periodic pipeline flushes
- **Activation Recomputation** - only input activation on each stage are stashed and re-run forward pass to get activations needed in backward pass
- **Planner** - for every possible width and depth, compare throughput and memory for -
    - Model and data parallelism without pipelining
    - Hybrid parallelism with pipelining without activation recomputation
    - Hybrid parallelism with pipelining and recomputation

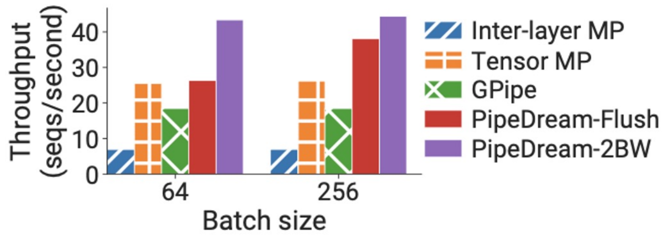

Before: $W_1^{(0)}, W_1^{(0)}$
After: $W_1^{(0)}, W_1^{(4)}$
Worker 1
Worker 2
Worker 3
Worker 4
Time
Before: $W_4^{(0)}, W_4^{(0)}$
After: $W_4^{(0)}, W_4^{(4)}$
$t = 21$
Forward Pass
Backward Pass

PipeDream-2BW (2 width, 3 stages)

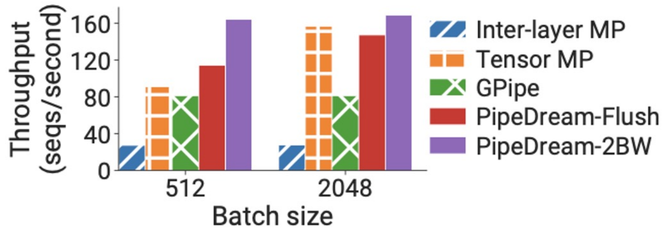


PipeDream-2BW (cont.)

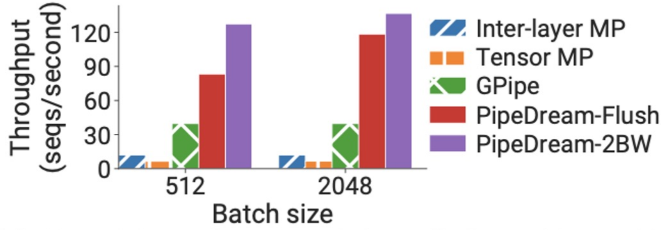
- Key Takeaways -
    - Global batch size with gradient accumulation B ↑ ⇒ Throughput ↑
    - Pipeline depth *d* ↑ ⇒ Throughput ↓
    - Pipeline depth *d* ↑ ⇒ maximum microbatch size for each GPU ↑
    - Larger and larger models can be trained as Pipeline depth *d* ↑



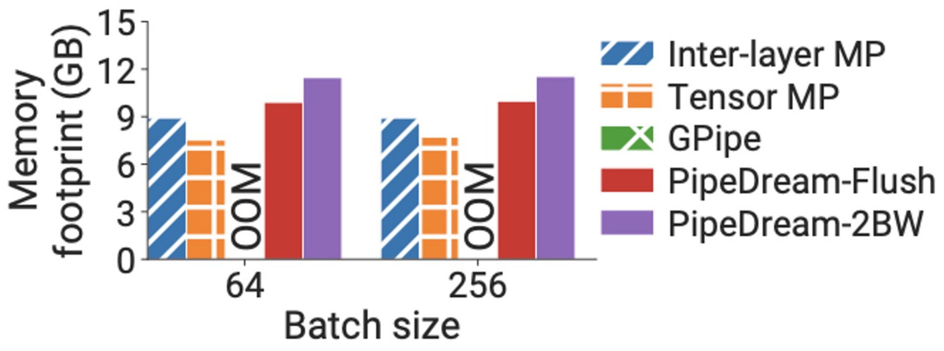
(a) GPT, 2.2B, 8-way model parallelism (8×V100s).



(b) GPT, 2.2B, 8-way model parallelism (64×V100s).



(c) GPT, 3.8B, 16-way model parallelism (64×V100s).



Observations and Insights

- Increasing the number of machines causes the communication overhead to increase for DNN models like AlexNet, VGG16, S2VT, ResNet50 and Inceptionv3
- PipeDream - using faster GPUs result in faster end-to-end training time for VGG training (from 220 hours on 8-machine cluster with NVIDIA Titan X GPU to less than 100 hours on 8-machine cluster with NVIDIA V100 GPU)
- PipeDream-2BW - 20X faster training of GPT (~3.8 billion params) vs MP approach Megatron
- PipeDream-2BW - 3.2X faster training of GPT vs PP approach GPipe

Conclusions

- Pipeline Parallel training methods like GPipe, PipeDream and PipeDream-2BW improve on the drawbacks of data parallelism and model parallelism -
    - Better GPU utilization as compared to Model Parallelism methods
    - Able to train large DNN faster as compared to Data Parallelism methods

- Using Hydra, DL researchers to save time when training multiple models in parallel