GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, et. al, NIPS 2019

CSC2541: Large Models

Presented by:

Yi (Tom) Lu, Keyu (Roy) Bai

January 31st, 2025

UNIVERSITY OF TORONTO

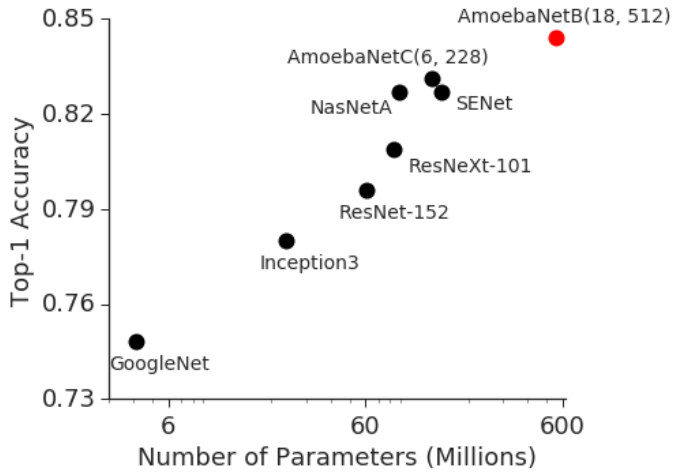
DEFY GRAVITY

Research Background

Increasing Model Size:

Larger and deeper networks provide better performance

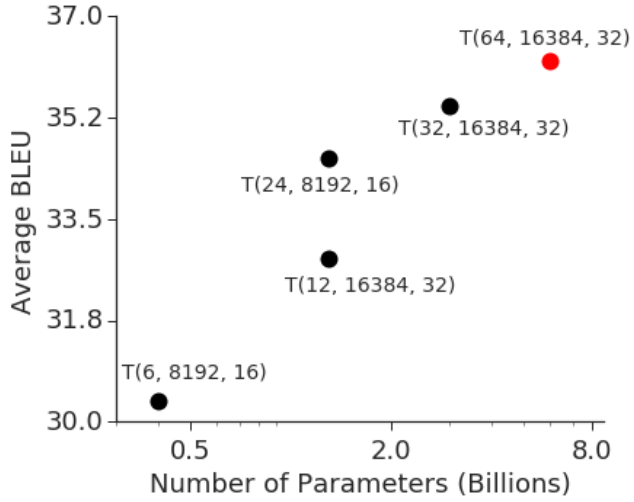
Demand for scaling neural networks are increasing



Difficulty in Training Large Models:

Hardware Constraints force users to explore model parallelism methods

Existing methods: model and architecture specific



Motivation:

Develop novel approach enabling training larger models

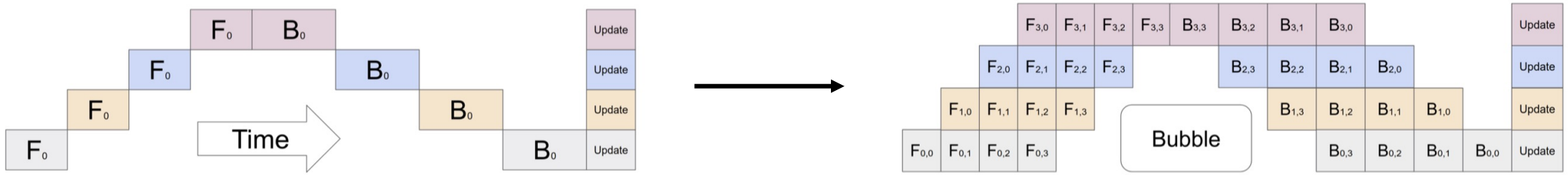
UNIVERSITY OF TORONTO

Figure Source: Huang, Yanping et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism."

TL;DR:

1. Scalable Model Parallel Library developed by Google, implemented in various major Deep Learning Libraries including Tensorflow and PyTorch

2. Optimize GPU resource usage via Data Parallelism



3. Optimize vRAM Usage via Gradient Checkpointing

$$O\left(N \cdot \frac{L}{K} \cdot d\right) \longrightarrow O\left(N + \frac{L}{K} \cdot \frac{N}{M}\right)$$ (Quadratic to nearly Linear)

4. High efficiency, Highly flexible and reliable model parallelism solution

Vanilla Model Parallelism

Motivation:

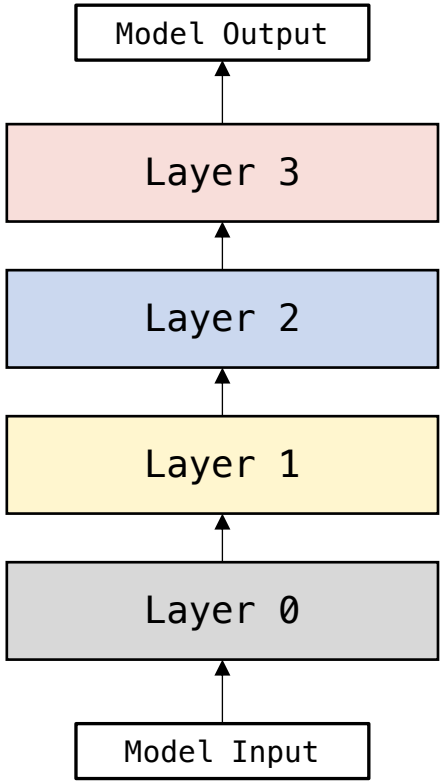
Separately a large model by layers then load them to multiple GPUs

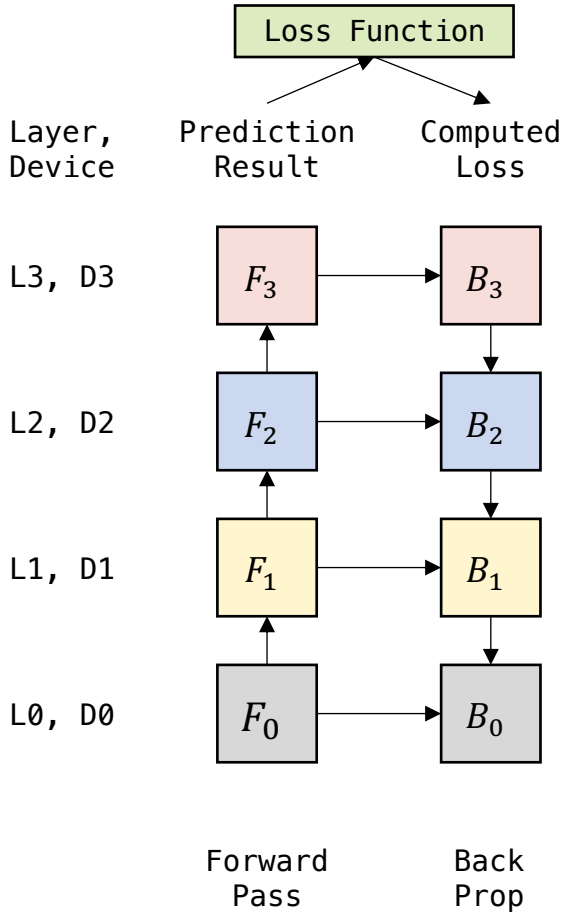
Computations:

Sequentially perform Forward Pass and Backpropagation

Model Parallelism:

Distribute model layers to devices



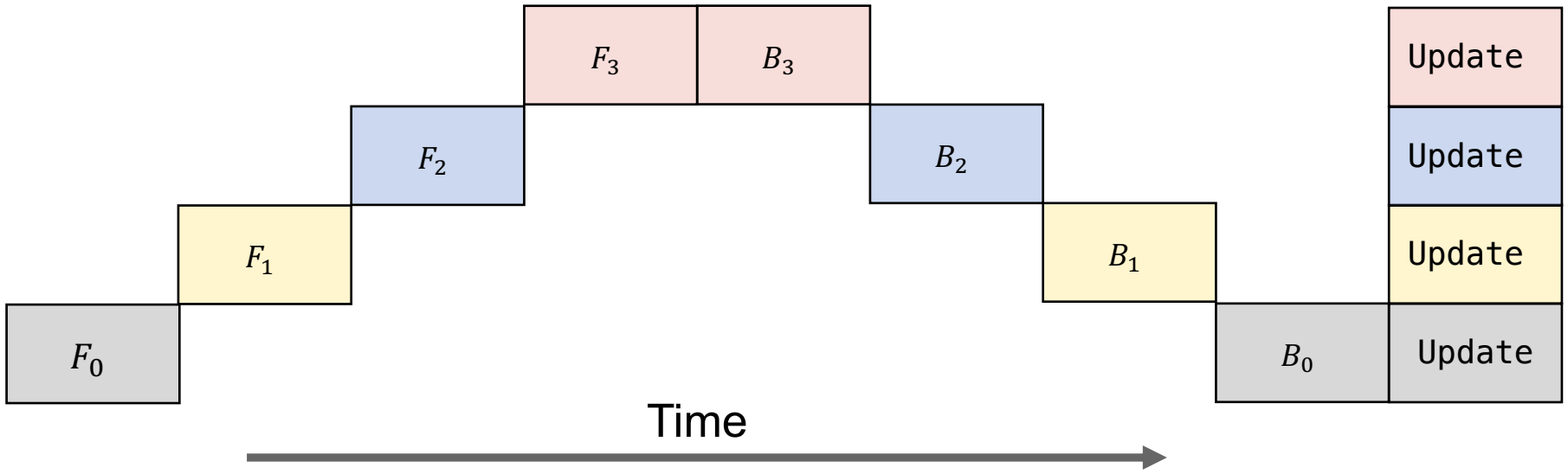


Vanilla Model Parallelism

Computation:

Perform forward and backward pass for each layer on each GPU sequentially,

Then update the gradients for each layer



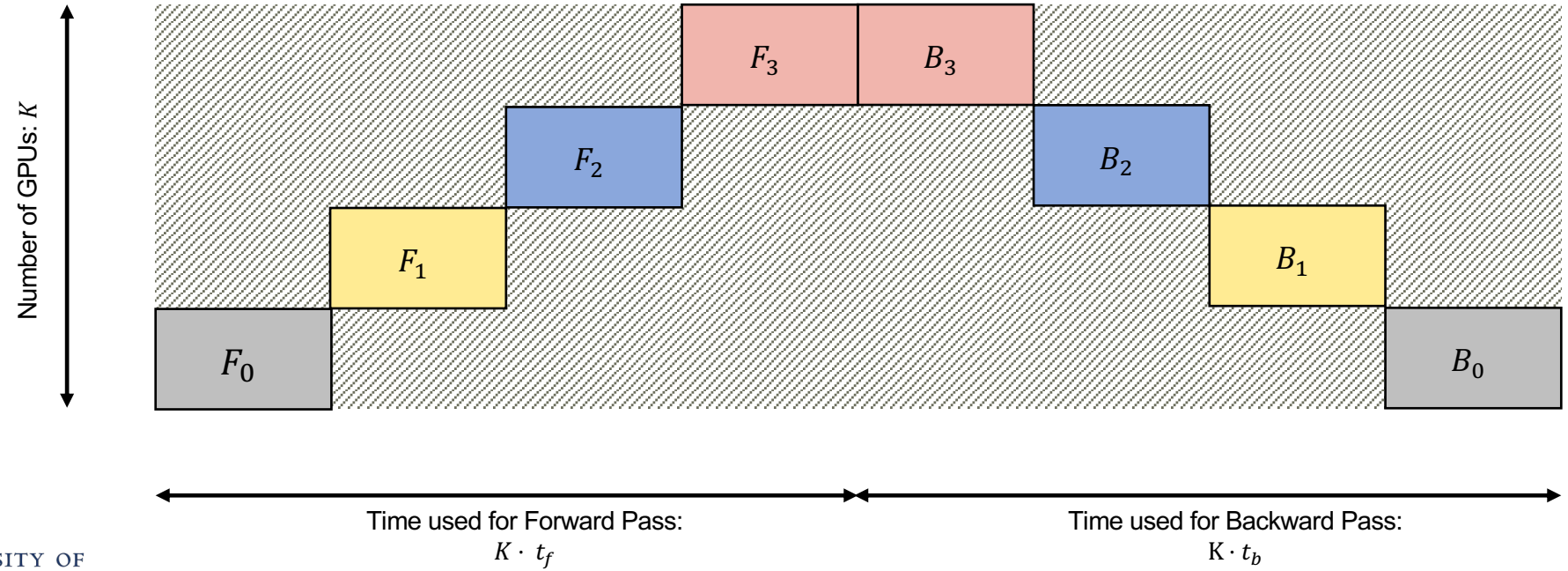
Vanilla Model Parallelism

Problems:

1. Low GPU Utilization:

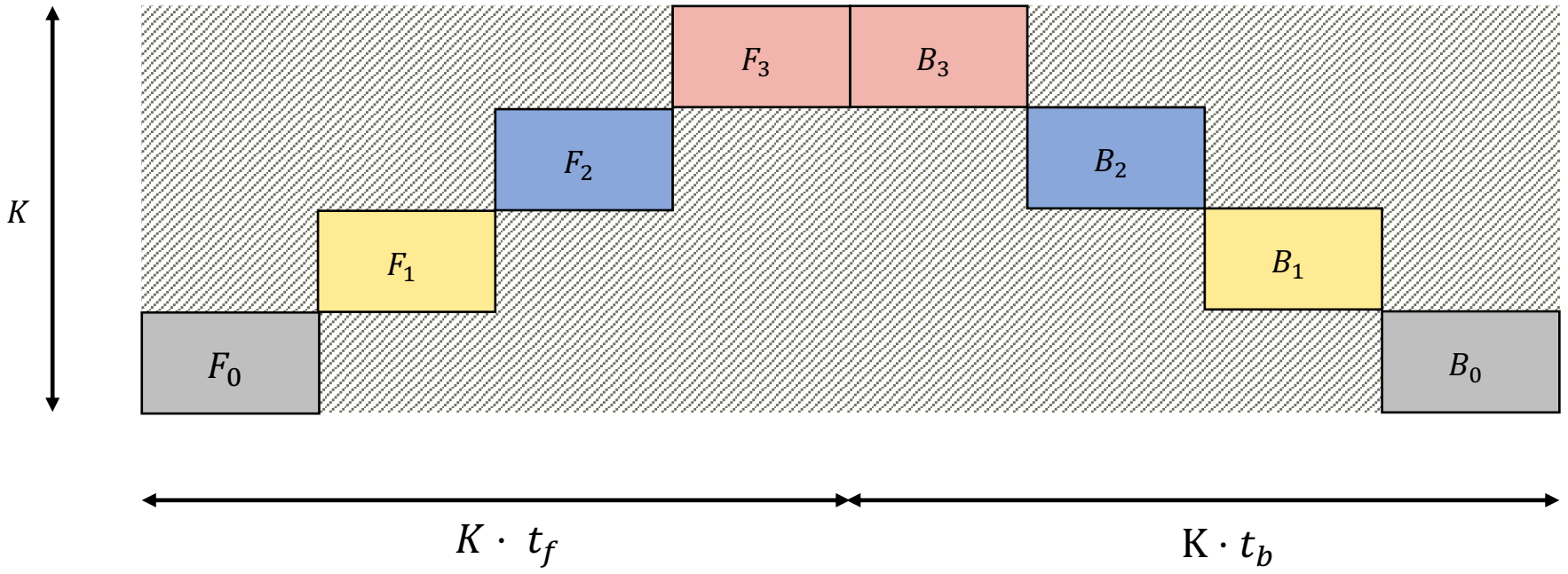
   All grayscale areas represent unused GPU times!

   How bad it is? $O\left(\frac{K-1}{K}\right)$ for number of GPUs: K



UNIVERSITY OF TORONTO

Vanilla Model Parallelism



Number of GPUs: K

Forward Pass per GPU: t_f

Backward Pass per GPU: t_b

Total Forward Pass Time: $K \cdot t_f$

Total Backward Pass Time: $K \cdot t_b$

Vanilla Model Parallelism

Total Time Spent:

$$K^2 \cdot (t_f + t_b)$$

Time used for computation:

$$K \cdot (t_f + t_b)$$

Proportion of Wasted GPU time:

$$1 - \frac{K \cdot (t_f + t_b)}{K^2 \cdot (t_f + t_b)} = \frac{K-1}{K}$$

Time complexity of Bubble Area:

$$O(\frac{K-1}{K})$$

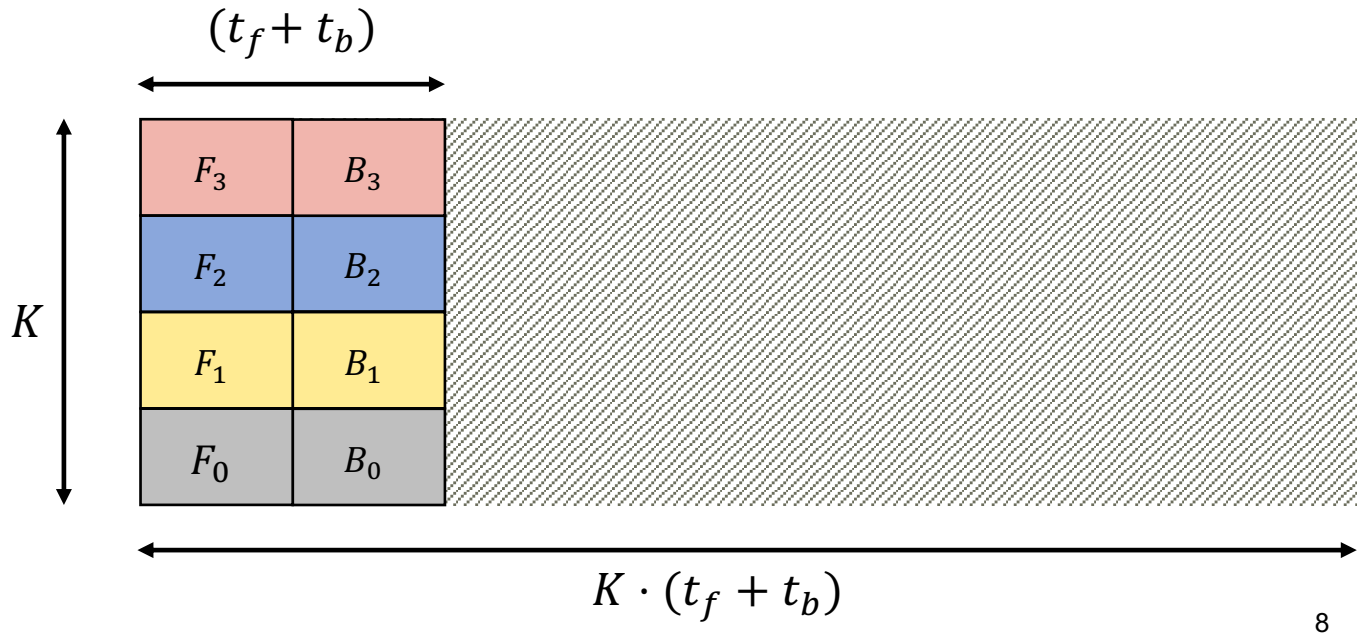
Number of GPUs: K

Forward Pass per GPU: t_f

Backward Pass per GPU: t_b

Total Forward Pass Time: $K \cdot t_f$

Total Backward Pass Time: $K \cdot t_b$



UNIVERSITY OF TORONTO

Vanilla Model Parallelism

Total Time Spent:

$$K^2 \cdot (t_f + t_b)$$

Time used for computation:

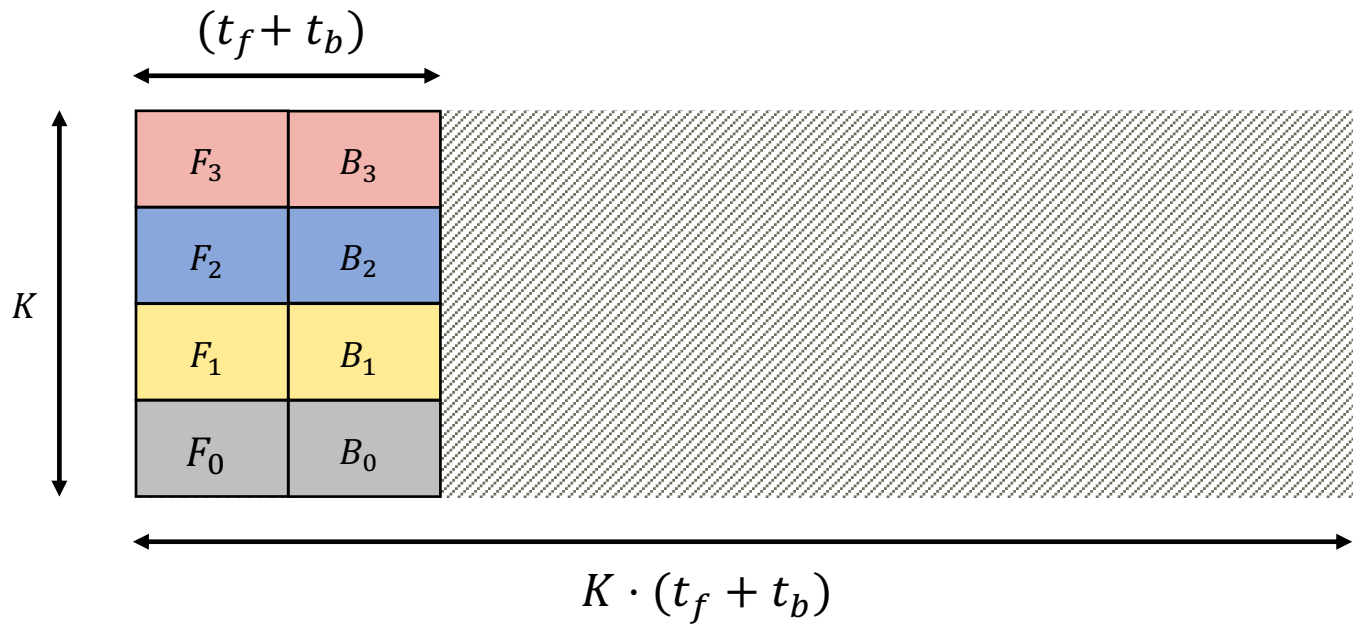
$$K \cdot (t_f + t_b)$$

Proportion of Wasted GPU time:

$$1 - \frac{K \cdot (t_f + t_b)}{K^2 \cdot (t_f + t_b)} = \frac{K-1}{K}$$

Time complexity of Bubble Area:

$$O(\frac{K-1}{K})$$



What can be observed:

$$\lim_{K \to \infty} \left(\frac{K-1}{K}\right) = 1$$

More GPUs are utilized, more resources will be wasted

UNIVERSITY OF TORONTO

Vanilla Model Parallelism

Problems:

2. Store excessive intermediate results.

   How bad it is?

$$O\left(N \cdot \frac{L}{K} \cdot d\right)$$

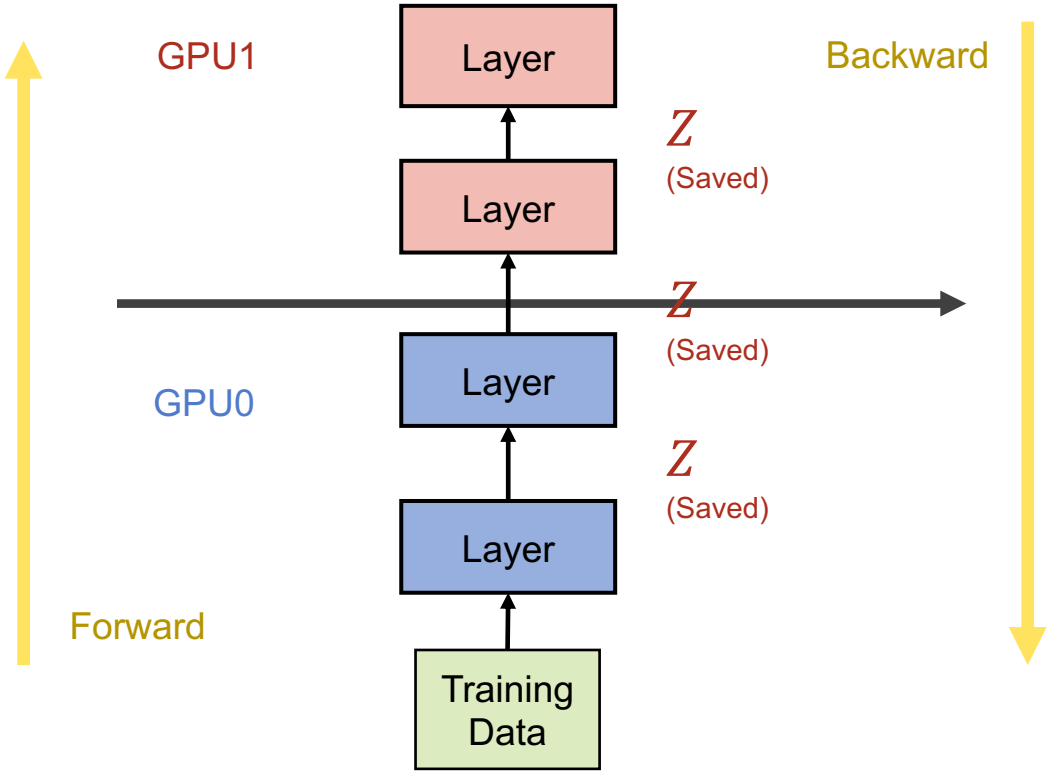
GPU Number: K

Batch Size: N

Intermediate Result: Z

Model Layer number: L

Width of each layer: d



With Model width or Depth increasing, the advantages brought by increasing K might be cancelled out

Conclusion of Vanilla Model Parallelism

Problems:

1. Low GPU utilization efficiency

   More GPUs are utilized, more resources will be wasted

2. Intermediate activations taken up large amount of vRAM

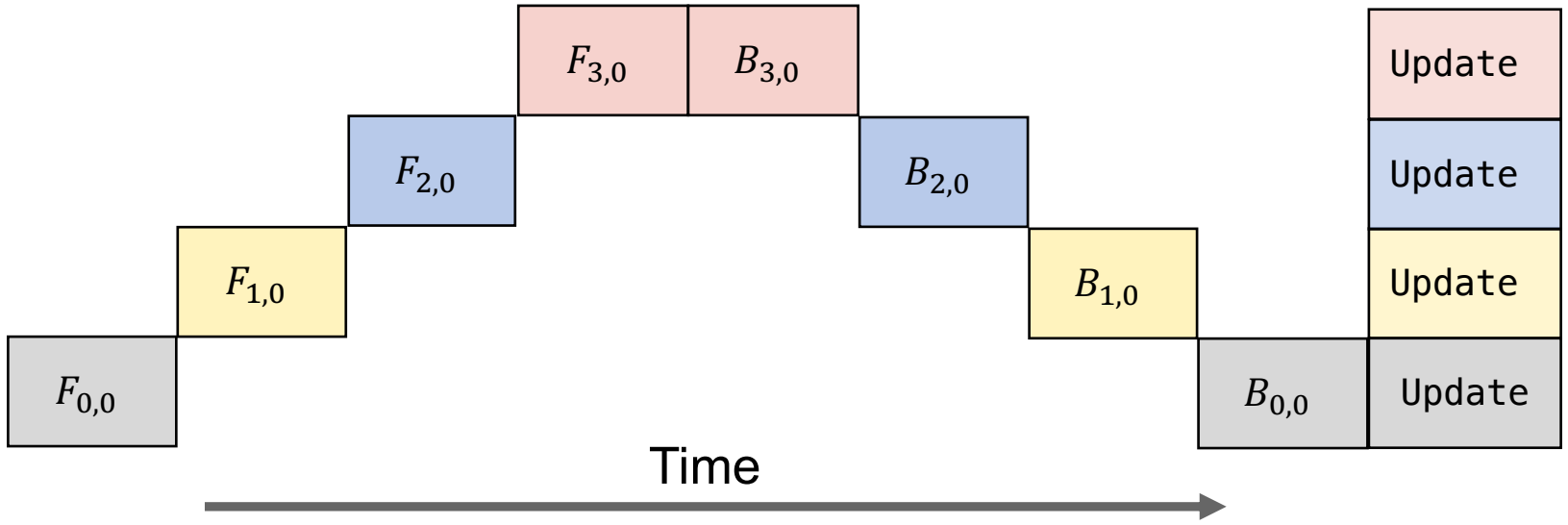
   With Model width or Depth increasing, the advantages brought by increasing K might be cancelled out

Solution: **GPipe**

GPipe

Observation:

If Divide a minibatch into micro-batches, we can observe…



GPipe

Observation:

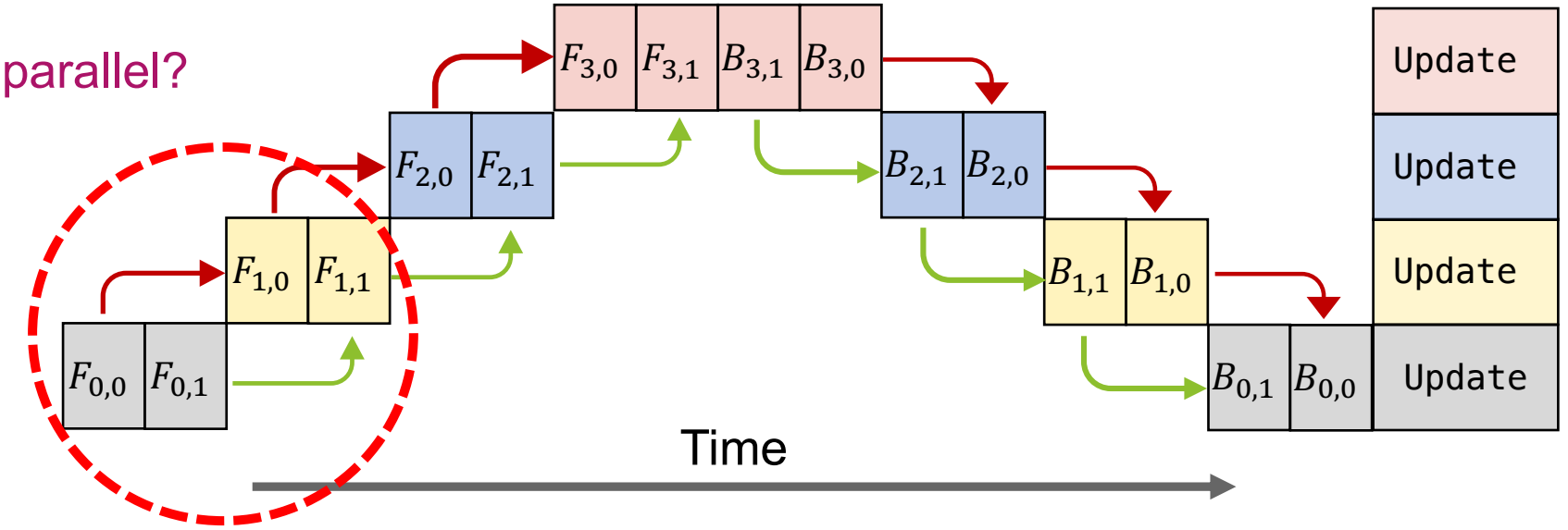
If Divide a minibatch into micro-batches, we can observe…

Computing $F_{0,1}$ has nothing to do with computing computing $F_{1,0}$

Computing $F_{1,1}$ has nothing to do with computing computing $F_{2,0}$

……

So can they be computed in parallel?



Notation of $F_{i,j}$:

i: Model Layer / GPU number

j: micro-batch number

GPipe

Observation:

If Divide a minibatch into micro-batches, we can observe…

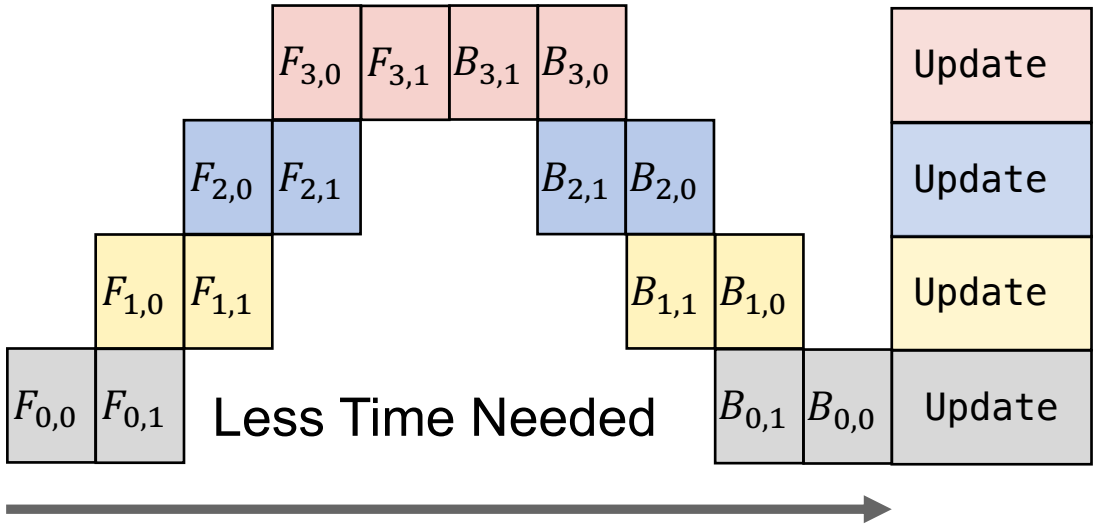
Computing Forward/Backward Pass $F_{i,j}$ $(B_{i,j})$ for i th layer on i th GPU

Has nothing to do with

Computing Forward/Backward Pass $F_{i+1,j-1}$ $(B_{i+1,j-1})$ for $i+1$ th layer on $i+1$ th GPU

So they can be parallelized!

Notation of $F_{i,j}$:
i: Model Layer / GPU number
j: micro-batch number

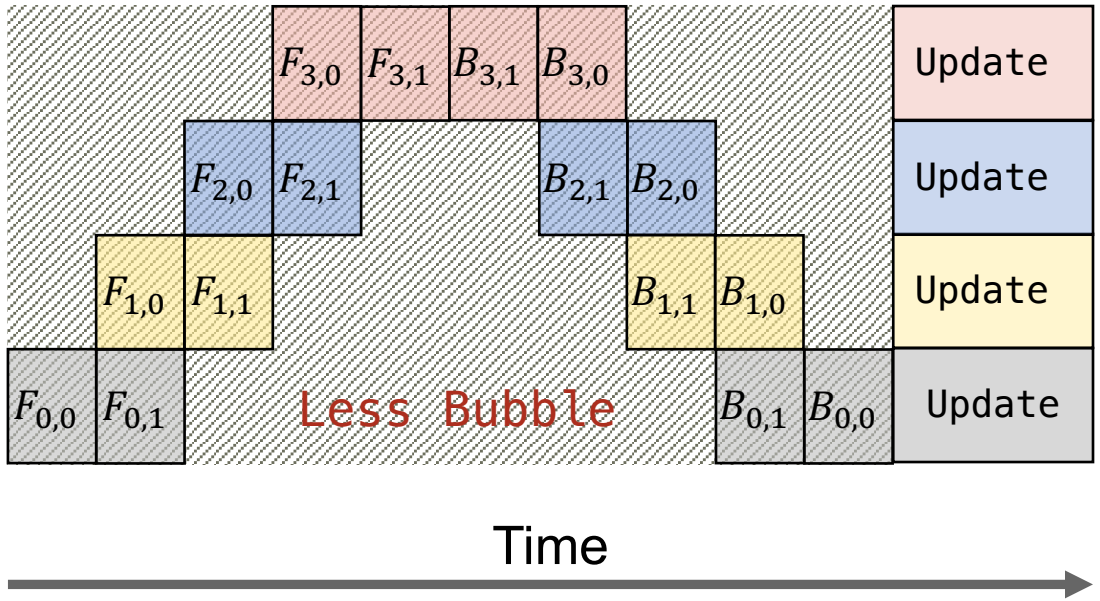


GPipe

Batch Parallelism:

Parallelize training data by dividing them further into micro-batches

Greatly reduce bubble size and increased GPU resource utilization



Number of GPUs (Pipeline Depth): K

Number of Micro batch: M

Forward Pass time per GPU: t_f

Backward Pass time per GPU: t_b

Total Forward Pass Time: $K \cdot (\mathrm{M} + K - 1) \cdot t_f$

Total Backward Pass Time: $K \cdot (M + K - 1) \cdot t_b$

GPipe:

Batch Parallelism:

Parallelize training data by dividing them further into micro-batches

Greatly reduce bubble size and increased GPU resource utilization

Total Time Spent:

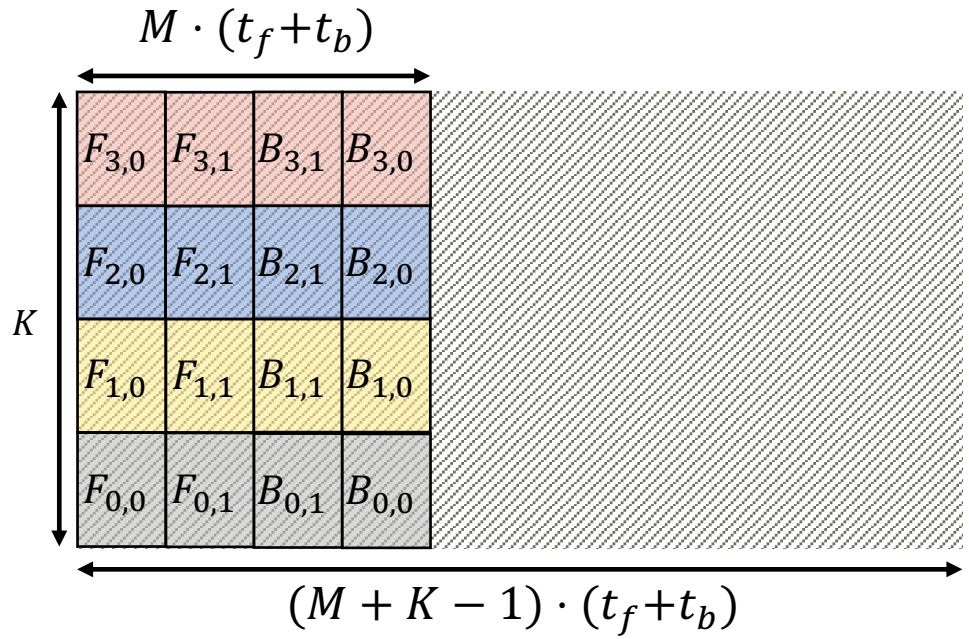
$$K \cdot (M + K - 1) \cdot (t_f + t_b)$$

Time used for computation:

$$K \cdot M \cdot (t_f + t_b)$$

Proportion of Wasted GPU time:

$$1 - \frac{K \cdot M \cdot (t_f + t_b)}{K \cdot (M + K - 1) \cdot (t_f + t_b)} = \frac{K - 1}{(M + K - 1)}$$



GPipe:

Batch Parallelism:

Parallelize training data by dividing them further into micro-batches

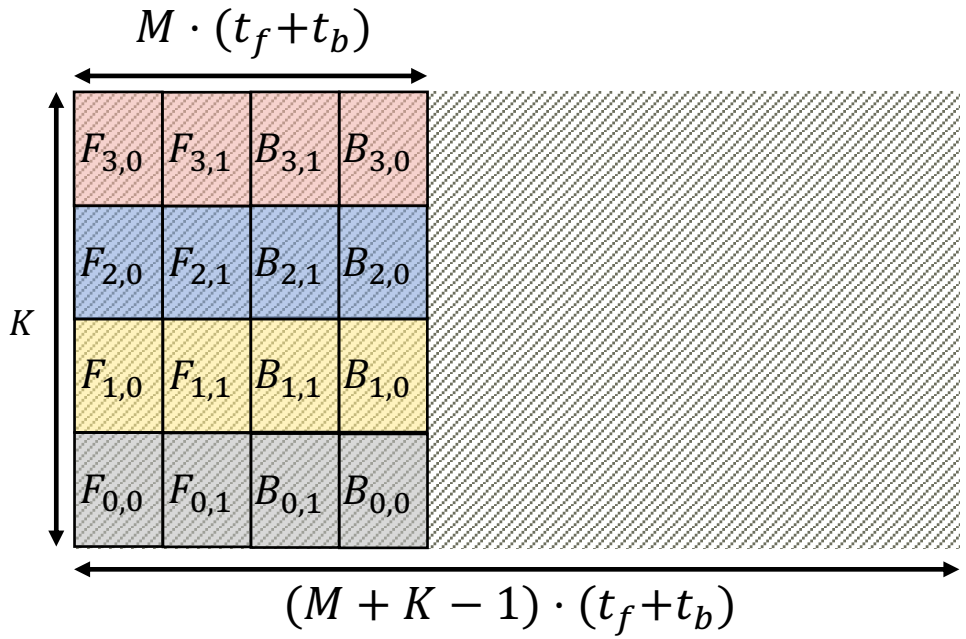
Greatly reduce bubble size and increased GPU resource utilization

Bubble Area Time Complexity:

$$O(\frac{K-1}{M+K-1})$$

More minibatches == Smaller bubble area

=> Higher GPU utilization efficiency!



GPipe:

Gradient Checkpointing

To compute backpropagations, each layer's intermediate activations need to be stored.

Without Gradient Checkpointing, peak total GPU vRAM storage complexity :

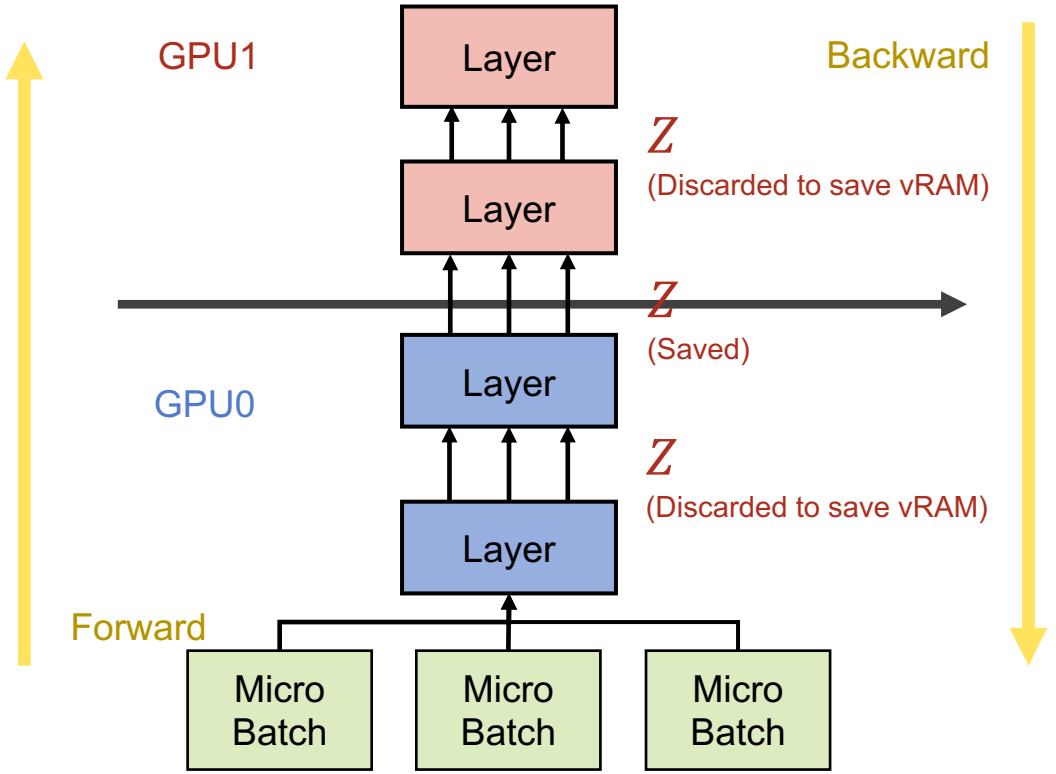
$$O(N \cdot L)$$

Where N: Batch Size, L: Total number of Layers, Assume each layer's size are similar.

Idea of Gradient Checkpointing:

Only store activations at final layers for each partition.

Recompute intermediate activations within partitions during backpropagation.



GPipe:

Gradient Checkpointing

For Gradient Checkpointing, peak total GPU vRAM storage complexity:

$$O(N + \frac{L}{K} \cdot \frac{N}{M})$$

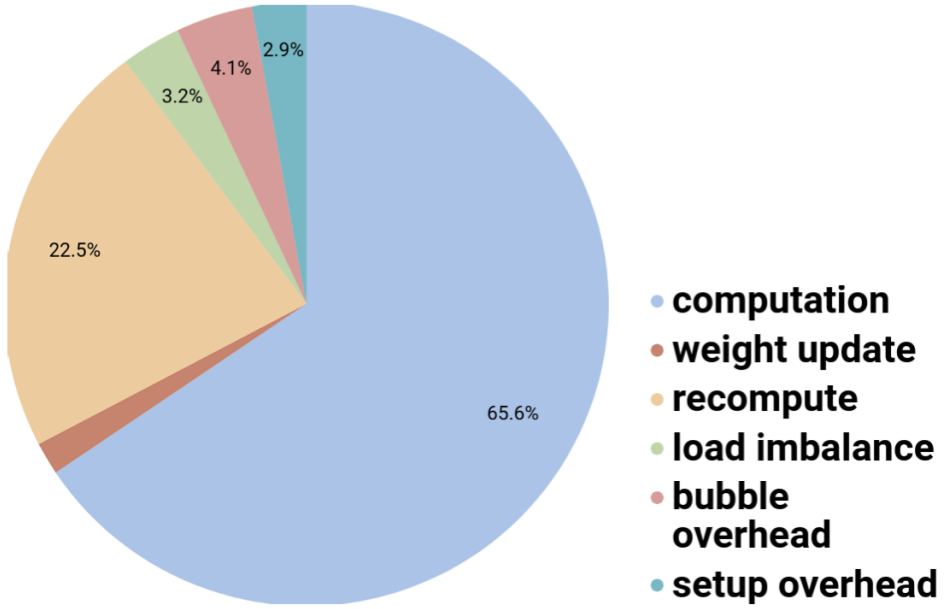
Where

N: Batch Size, L: Total number of Layers, K: Number of Partitions, M: Micro batch Size

Effect of Gradient Checkpointing:

Significantly reduce peak vRAM usage, trade off computation time with less vRAM consumption

Table 4: Time step breakdown



Tradeoff: took around 1/3 total running time to recompute activations

UNIVERSITY OF TORONTO

Figure Source: Huang, Yanping et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism."

Evaluations

Research Questions:

Does GPipe efficient in reducing vRAM usage?

Does GPipe efficient in accelerating model training?

Single Card

Pipeline-1,2,4,8 == GPipe + Gradient Checkpointing with 1, 2, 4, 8 cards

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
CNN: Image Classification					
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	**Naive-1**	**Pipeline-1**	**Pipeline-8**	**Pipeline-32**	**Pipeline-128**
Transformer: Language Translation					
Transformer-L	3	13	103	415	1663
# of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

UNIVERSITY OF TORONTO

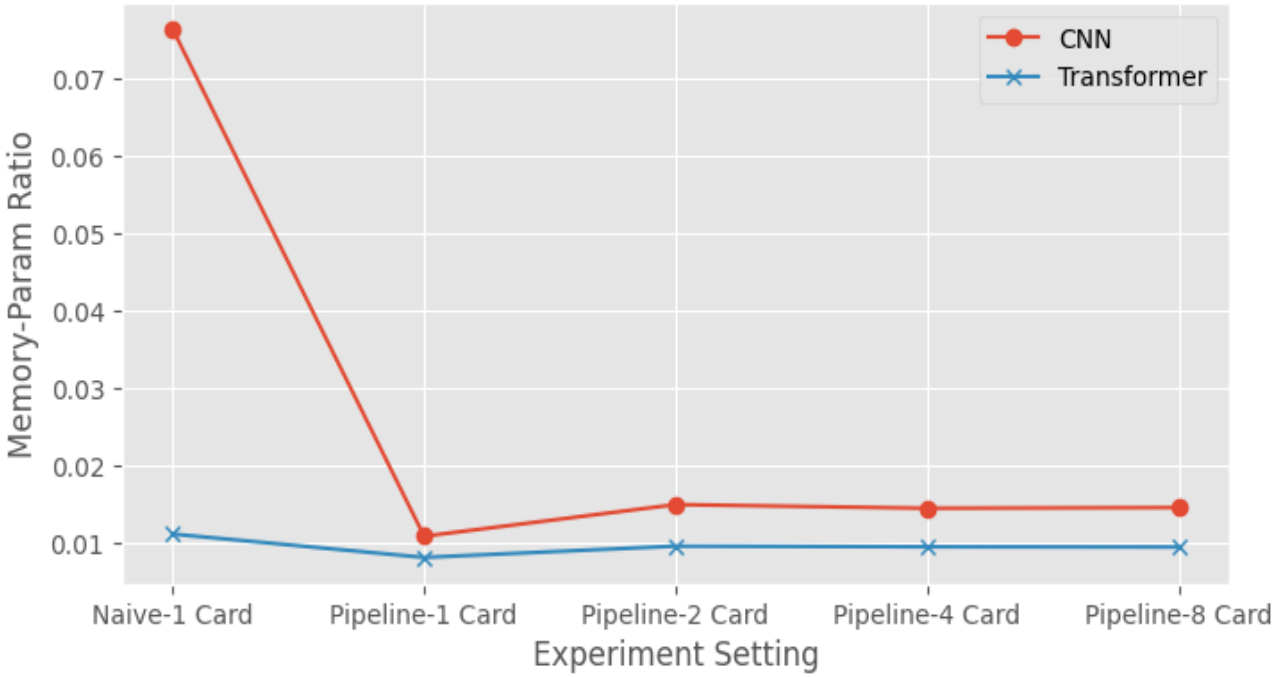
Table Source: Huang, Yanping et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism."

Evaluations

Research Question 1:

Does GPipe efficient in reducing vRAM Usage?

Define vRAM Usage Ratio:
Peak Activation Memory divided by Model Parameter size



Yes. Application of GPipe yields to significantly better GPU memory utilization efficiency

Evaluations

Research Question 2:

Does GPipe efficient in accelerating model training?

Yes. Effectiveness depend on number of micro-batches and model architecture

Test GPipe's effect in accelerating model training with NVLink disabled:

GPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 32$	1	1.7	2.7	1	1.8	3.3

Acceleration rate (1.7 times, 2.7 times, etc.)

- K: Number of GPUs, M: Number of Micro-batches
- Even with NVLink disabled, training speed still increases as number of GPU increases.
- CNN performs worse than Transformer due to imbalance model splitting.

Test GPipe's effect in accelerating model training with NVLink enabled:

TPU	AmoebaNet å			Transformer		
$K =$	2	4	8	2	4	8
$M = 1$	1	1.13	1.38	1	1.07	1.3
$M = 4$	1.07	1.26	1.72	1.7	3.2	4.8
$M = 32$	1.21	1.84	3.48	1.8	3.4	6.3

Acceleration rate (1.07 times, 1.21 times, etc.)

Performance depend on MicroBatch number: M.

- M = 1: no linearity between number of GPUs and training speed
- M = 4: noticeable improvement
- M = 32: significantly better, observed linearity on Transformer.

UNIVERSITY OF TORONTO

Table Source: Huang, Yanping et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism."

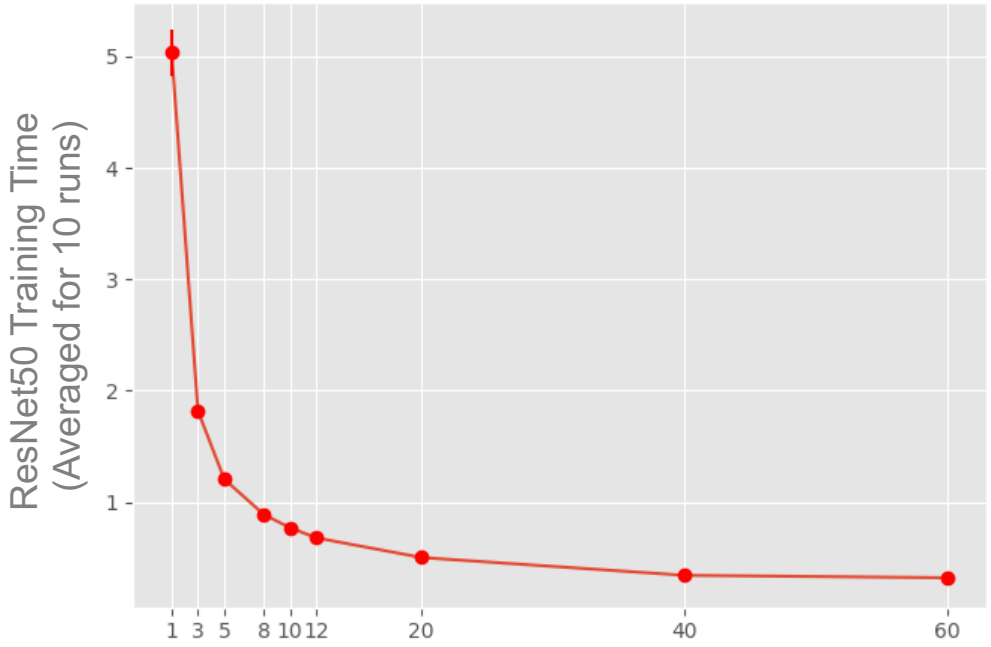
Jupyter Notebook Demo Result

Experiment Objective:

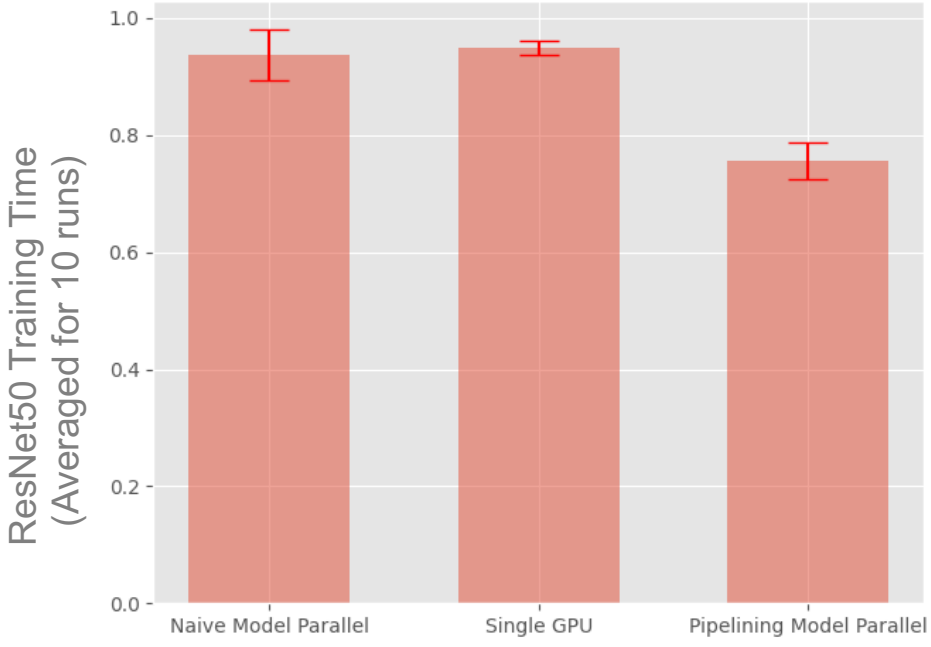
Demonstrate GPipe's effectiveness in accelerating CNN model training via toy examples

Experiment Environment: Kaggle Dev Notebook (2* Nvidia T4, batch size=128 only)

Experiment: Training Resnet 50 and evaluate training efficiency improvement brought by GPipe



Ablation on Micro-batch number
(More micro batches, faster training)



Comparing three strategies
(GPipe performs best)

UNIVERSITY OF TORONTO

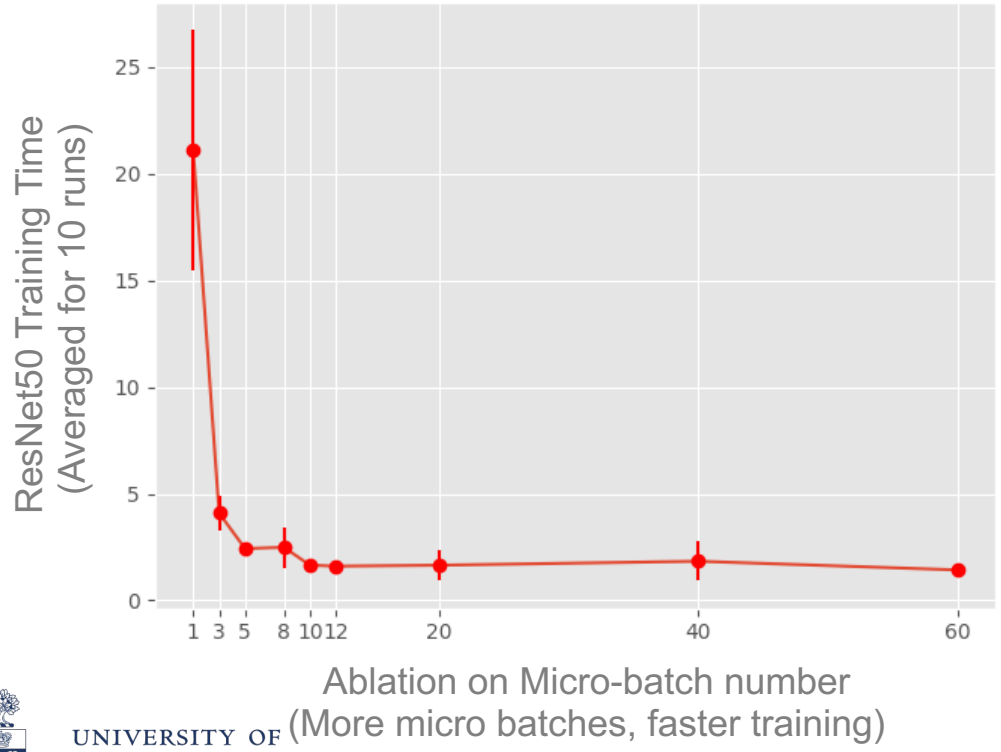
Jupyter Notebook Demo Result

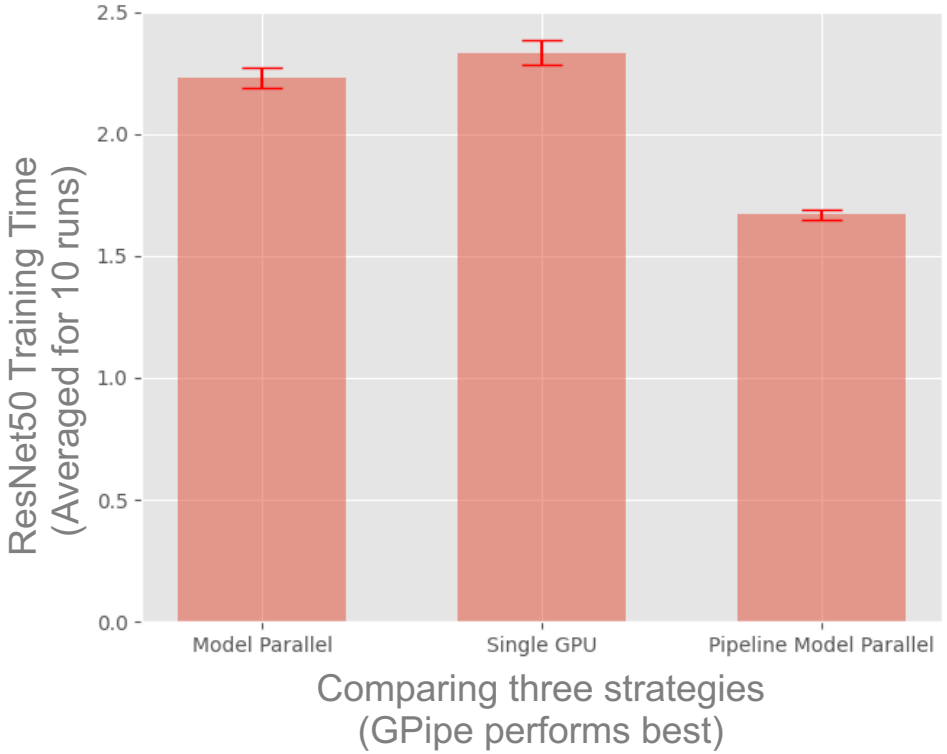
Experiment Objective:

Demonstrate GPipe's effectiveness in accelerating Transformer model training via toy examples

Experiment Environment: Kaggle Dev Notebook (2* Nvidia T4, batch size=128 only)

Experiment: Training Transformer and evaluate training efficiency improvement brought by GPipe





UNIVERSITY OF TORONTO

Scope and Limitations

Scope is Limited to Pipeline Parallelism

Higher-level model parallelism, including data parallelism and tensor parallelism can be implemented to achieve better efficiency (Megatron LM, ZeRO, etc.)

GPU vRAM usage could be further optimized

PipeDream (Microsoft, SOSP 2019):

Immediately perform backpropagation once a micro batch has completed forward propagation, can abandon micro-batches' activations earlier

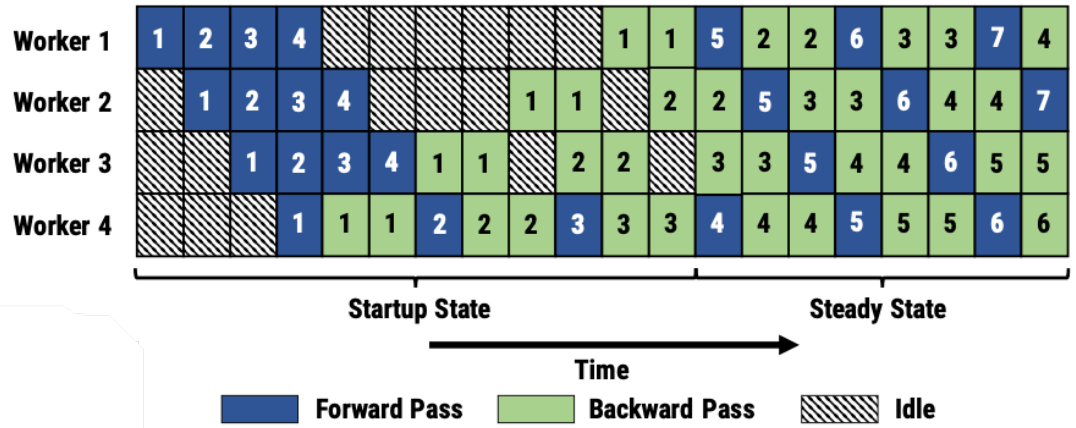


Figure Source: Harlap, Aaron et al. "PipeDream: Fast and Efficient Pipeline Parallel DNN Training."

UNIVERSITY OF TORONTO

Scope and Limitations

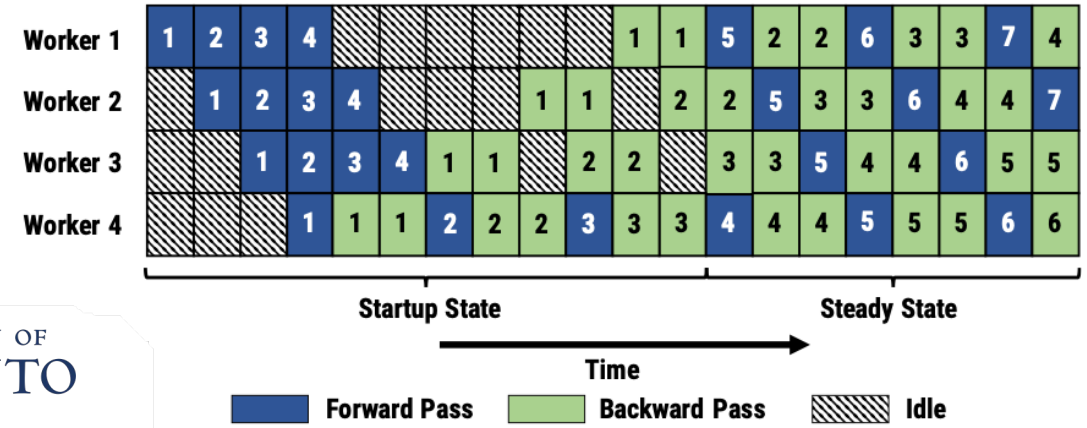
Scope is Limited to Pipeline Parallelism

Higher-level model parallelism, including data parallelism and tensor parallelism can be implemented to achieve better efficiency (Megatron LM, ZeRO, etc.)

GPU vRAM usage could be further optimized

PipeDream (Microsoft, SOSP 2019):

Immediately perform backpropagation once a micro batch has completed forward propagation, can abandon micro-batches' activations earlier



Why Google decide to made the trade-off?

1. Cannot reduce bubble size
2. Require maintaining multiple model weight versions for consistent gradient update
3. Further reduce memory usage but require complex implementation

UNIVERSITY OF TORONTO

Figure Source: Harlap, Aaron et al. "PipeDream: Fast and Efficient Pipeline Parallel DNN Training."

Summary

1. Proposed novel pipeline-parallelism:

   GPipe is achieved through batch-splitting mechanism

2. Performed intensive experiment:

   Demonstrated strong empirical results and proved the efficiency of GPipe

3. Introduced GPipe:

   Implemented as a scalable model-parallelism library enabling the training of giant neural networks on Tensorflow

Thanks