PipeDream: Generalized Pipeline Parallelism for DNN Training

Deepak Narayanan[§], Aaron Harlap[†], Amar Phanishayee[★], Vivek Seshadri[★],
Nikhil R. Devanur[★], Gregory R. Ganger[†], Phillip B. Gibbons[†], Matei Zaharia[§]

[★] **Microsoft Research** [†] **Carnegie Mellon University** [§] **Stanford University**

Deep Neural Networks have empowered state of the art results across a range of applications…

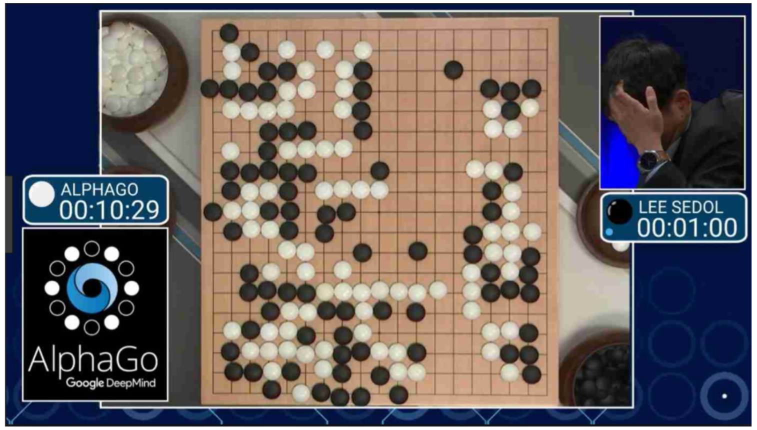
Image Classification

வணக்கம் என் பெயர் தீபக்

Hello, my name is Deepak

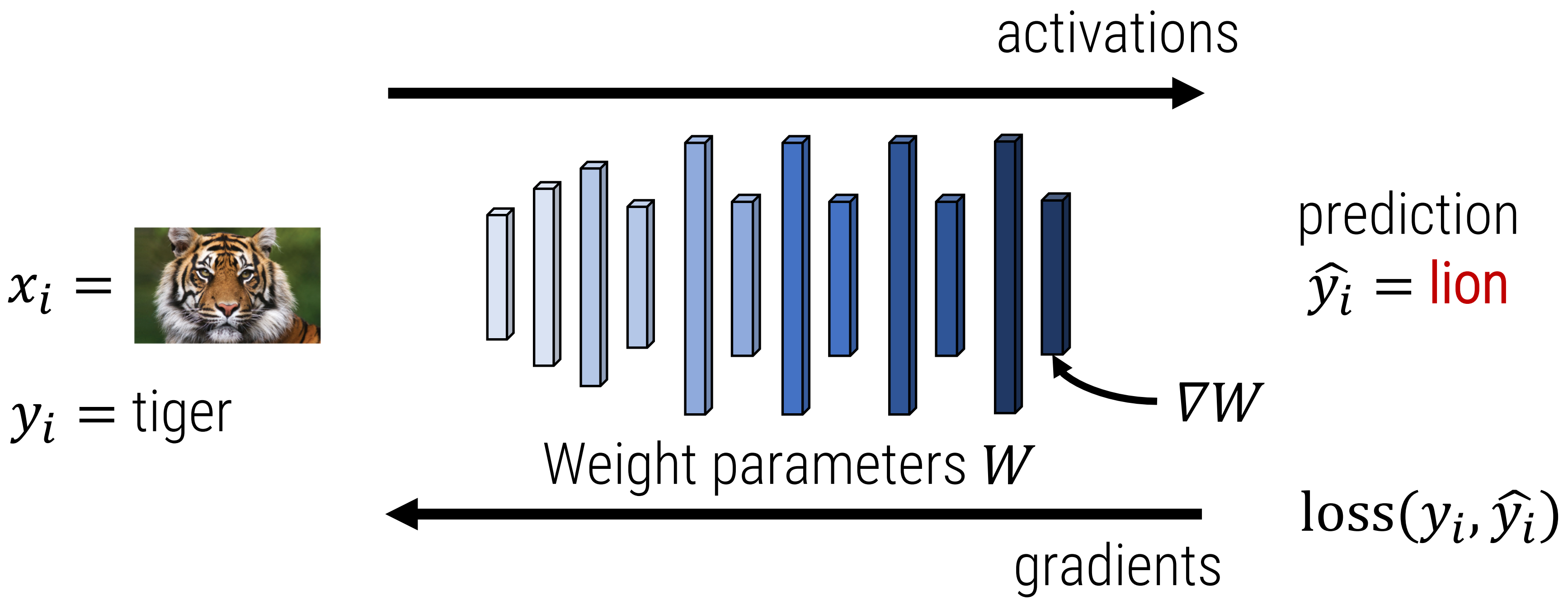
Machine Translation

Speech-to-Text



Game Playing

…but first need to be trained!



W optimized using standard iterative optimization procedures

$$W = W - \eta \cdot \nabla W$$

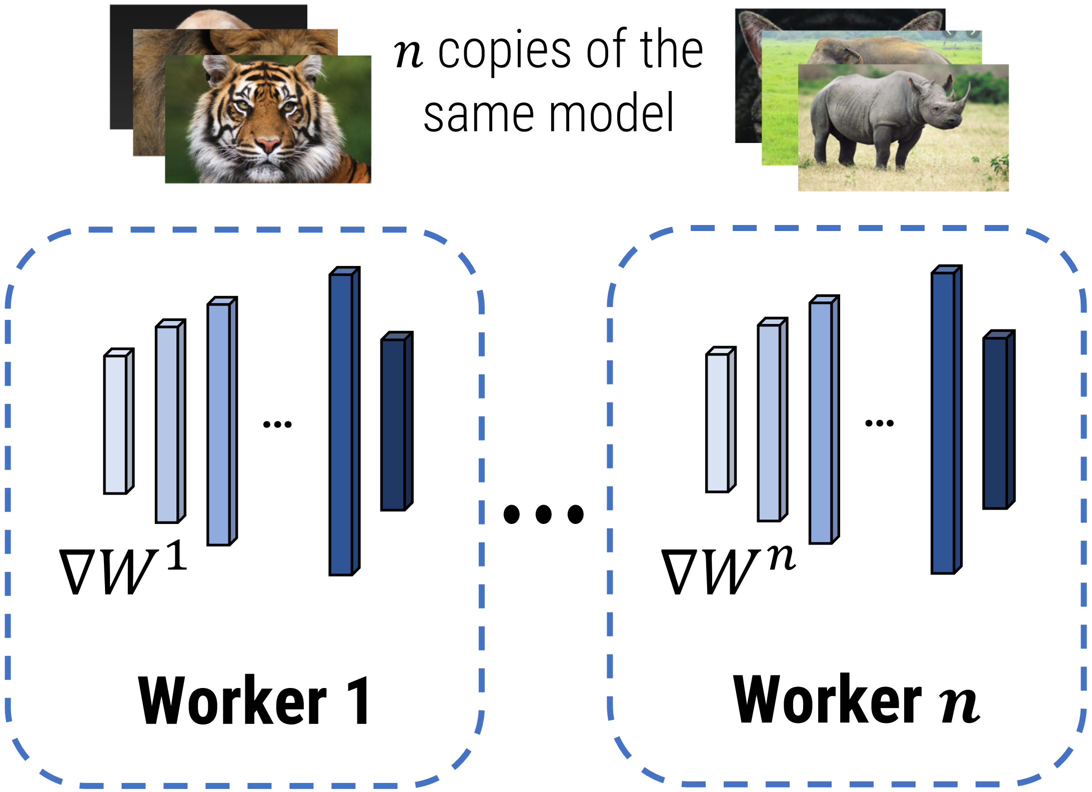
Background: DNN Training

Model training time- and compute- intensive!

W optimized using standard iterative optimization procedures

$$W = W - \eta \cdot \nabla W$$

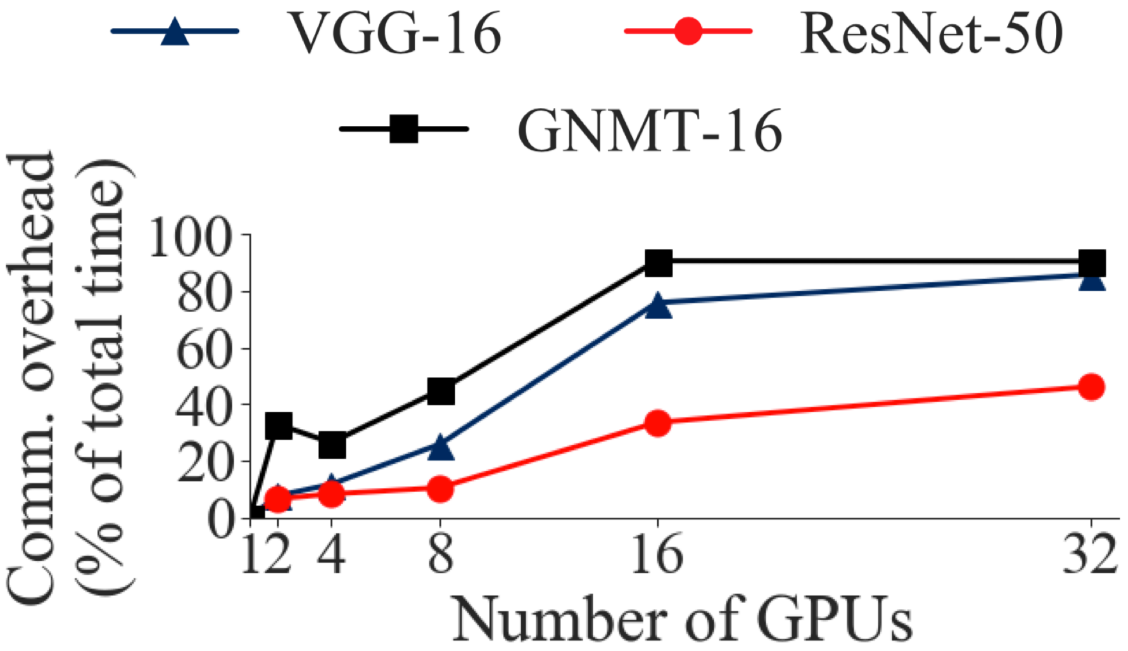
Parallelizing DNN Training: Data Parallelism



$$\nabla W = \nabla W^1 + \nabla W^2 + \cdots + \nabla W^n$$

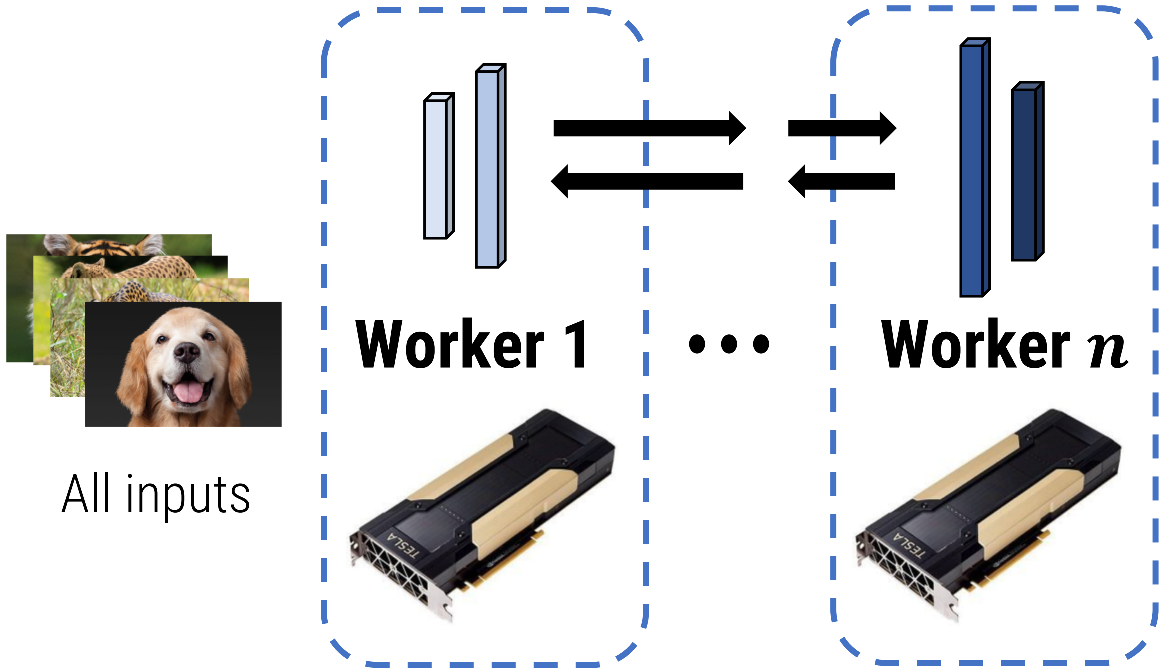
Gradient aggregation using AllReduce

Despite many performance optimizations, communication overhead high!



8xV100s with NVLink (AWS)
PyTorch + NCCL 2.4

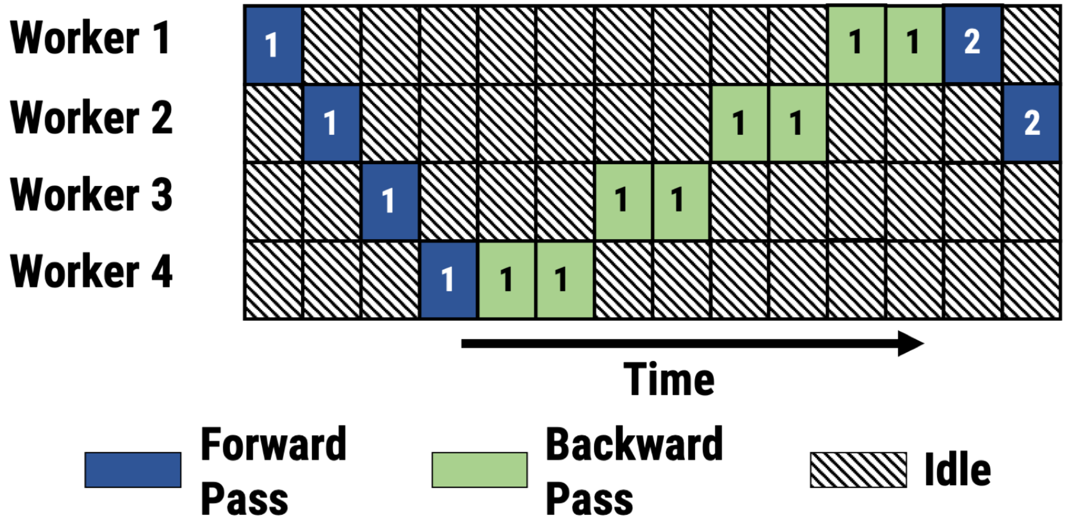
Parallelizing DNN training: Model Parallelism



Single version of weights split over workers

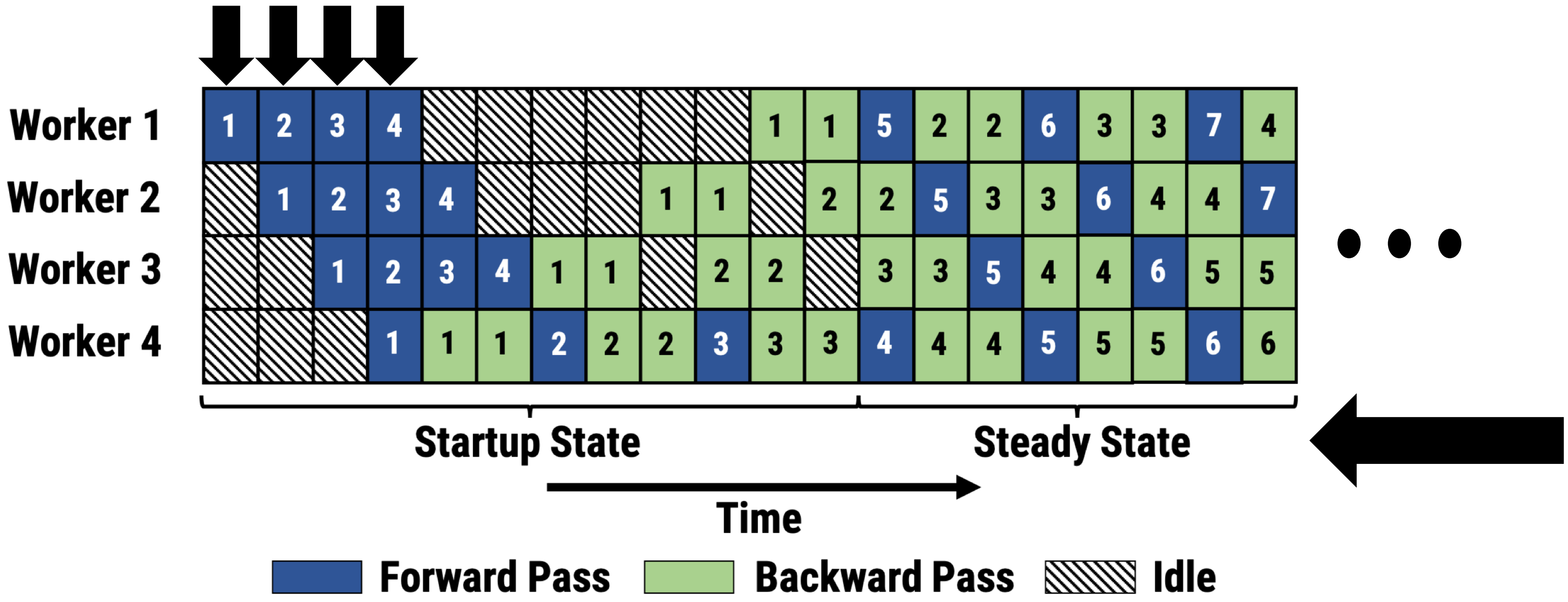
Activations and gradients sent between workers using peer-to-peer communication

Low hardware efficiency



PipeDream: Pipeline-Parallel Training

We propose **pipeline parallelism**, a combination of data and model parallelism with pipelining



Pipeline-parallel training up to **5.3x faster** than data parallelism without sacrificing on final accuracy of the model

Pipelining in DNN Training != Traditional Pipelining

- How should the operators in a DNN model be partitioned into pipeline stages?
  - Each operator has a **different computation time**
  - Activations and gradients need to be **communicated** across stages

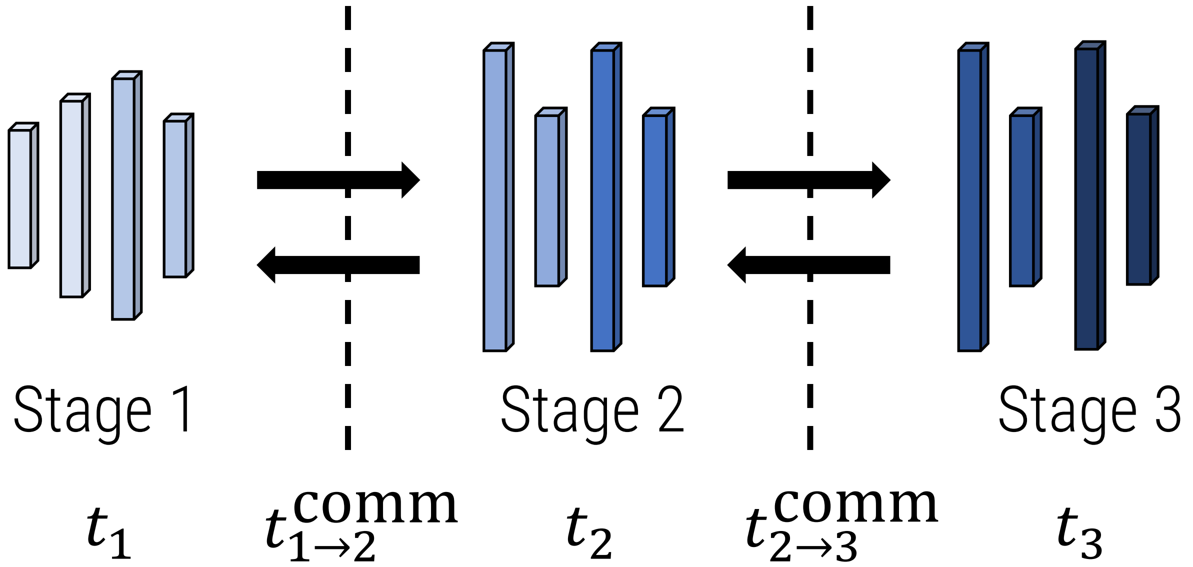
- How should forward and backward passes of different inputs be scheduled?
  - Training is **bidirectional**
  - Forward pass followed by backward pass to compute gradients

- How should weight and activation versions be managed?
  - Backward pass operators depend on **internal state** (W, activations)

Outline

- Background and Motivation
- **Challenges for effective pipeline-parallel training**
  - **Partitioning and load balancing operators across workers**
  - Scheduling of forward and backward passes of different inputs
  - Managing weights and activation versions for effective learning
- Evaluation

How do we assign operators to pipeline stages?



- Desiderata #1: t_1, t_2, t_3 as close to each other as possible
  - Compute resources seldom idle → better hardware efficiency

- Desiderata #2: $t^{\text{comm}}_{1\to2}$ and $t^{\text{comm}}_{2\to3}$ minimized
  - Less communication → better hardware efficiency

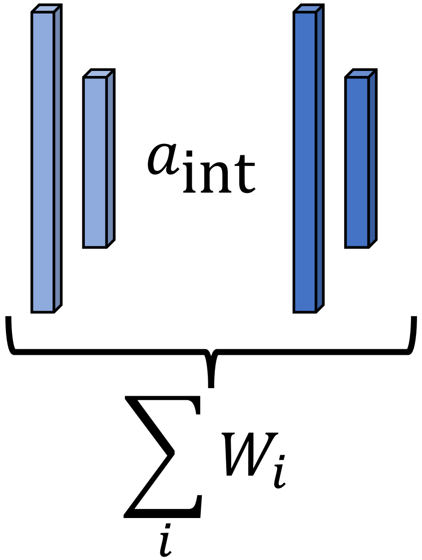
How do we assign operators to pipeline stages?

Compute time = 2

Throughput =
(1 / 2) × 2 = 1

Compute time = 1

Throughput = 1



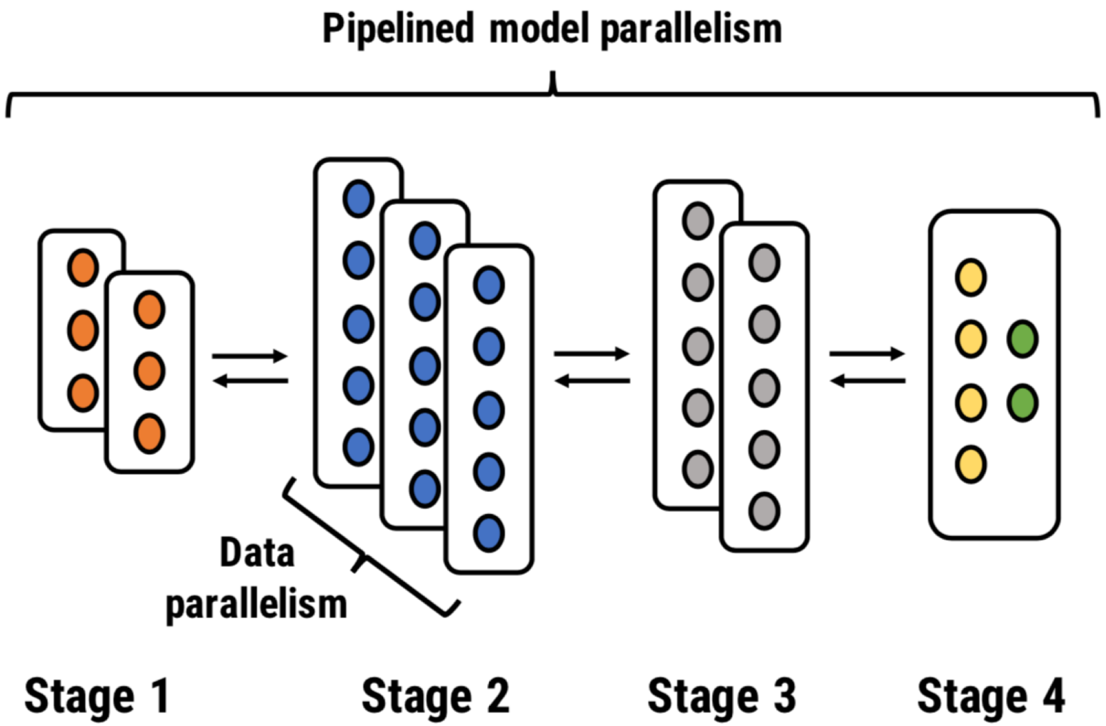
For **some** operators,
$\sum_i W_i < 2a_{\text{int}}$

Better load balancing across stages

Data-parallel communication small

Replication of stages helps load balance computation and reduce communication between workers

Example PipeDream configuration



Configuration: 2-3-2-1

Stages can have different replication factors

PipeDream Profiler and Optimizer

Input DNN → Profiler → Computational graph with profile

↓

Optimizer

↑

Deployment constraints such as number of accelerators, memory and interconnect characteristics

Determines a partitioning of operators amongst workers, while also deciding replication factors

Generalizes along many axes
- Hardware topologies
- Model structures
- Memory capacities of workers

See paper for details of algorithm!

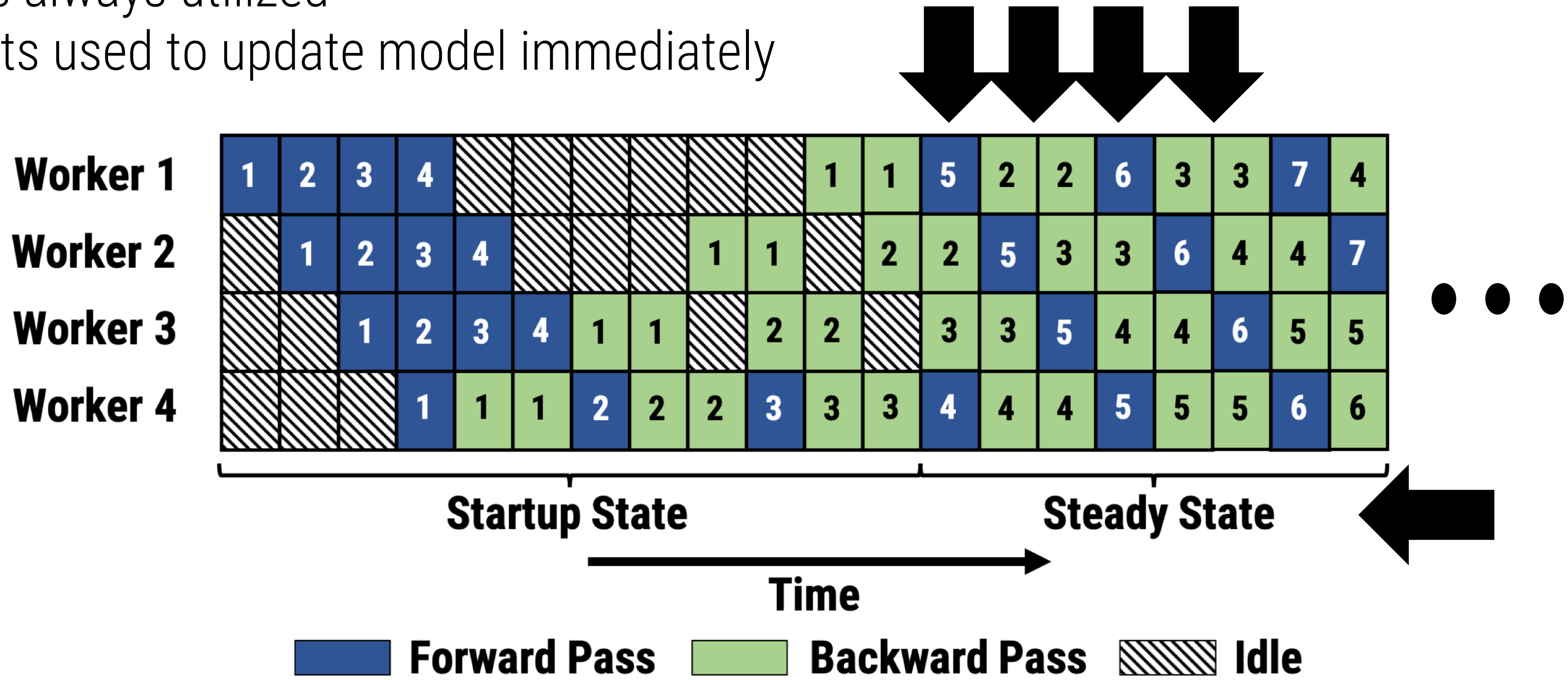
Outline

- Background and Motivation
- **Challenges for effective pipeline-parallel training**
  - Partitioning and load balancing operators across workers
  - **Scheduling of forward and backward passes of different inputs**
  - Managing weights and activation versions for effective learning
- Evaluation

1F1B Scheduling

Workers **alternate** between forward and backward passes

- Workers always utilized
- Gradients used to update model immediately



To support stage replication, need to modify this mechanism slightly – see paper for details!

Outline

- Background and Motivation

- **Challenges for effective pipeline-parallel training**
  - Partitioning and load balancing operators across workers
  - Scheduling of forward and backward passes of different inputs
  - **Managing weights and activation versions for effective learning**

- Evaluation

Naïve pipelining leads to weight version mismatches

Naïve pipelining leads to **mismatch in weight versions**

$$x_n \longrightarrow W_n \longrightarrow y_n$$ Forward pass

$$W_{n+1}$$

$$\vdots$$

$$\nabla x_n \longleftarrow W_{n+p} \longleftarrow \nabla y_n$$ Backward pass

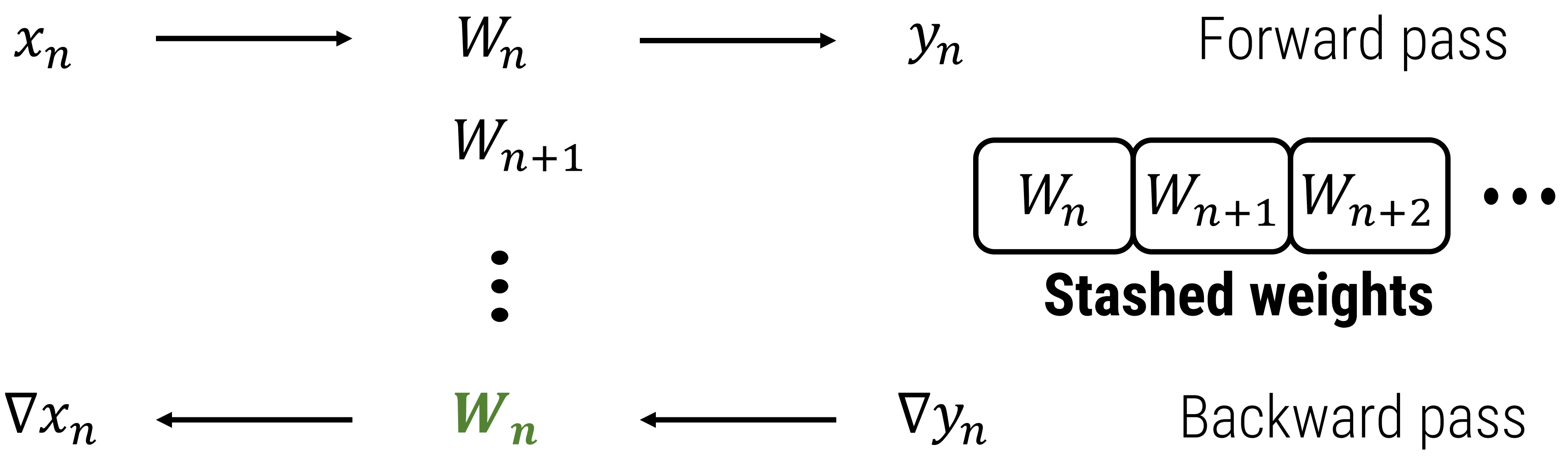
Input n sees updates in backward pass not seen in the forward pass, leading to incorrect gradients

1F1B Scheduling + Weight Stashing

Naïve pipelining leads to **mismatch in weight versions**

Store **multiple <weight, activation> versions**

- Ensures same weight versions used in both forward and backward pass



- Worst case memory footprint similar to data parallelism ($= n \cdot {(|W|+|A|)}/{n}$)

Outline

- Background and Motivation

- Challenges for effective pipeline-parallel training

- **Evaluation**
  - **Setup**
  - **Comparison to Data Parallelism on Time-to-Accuracy**
  - **Communication Overhead of Pipeline Parallelism**
  - **Comparison to Model Parallelism and Hybrid Parallelism on Throughput**
  - **PipeDream's Memory Footprint**

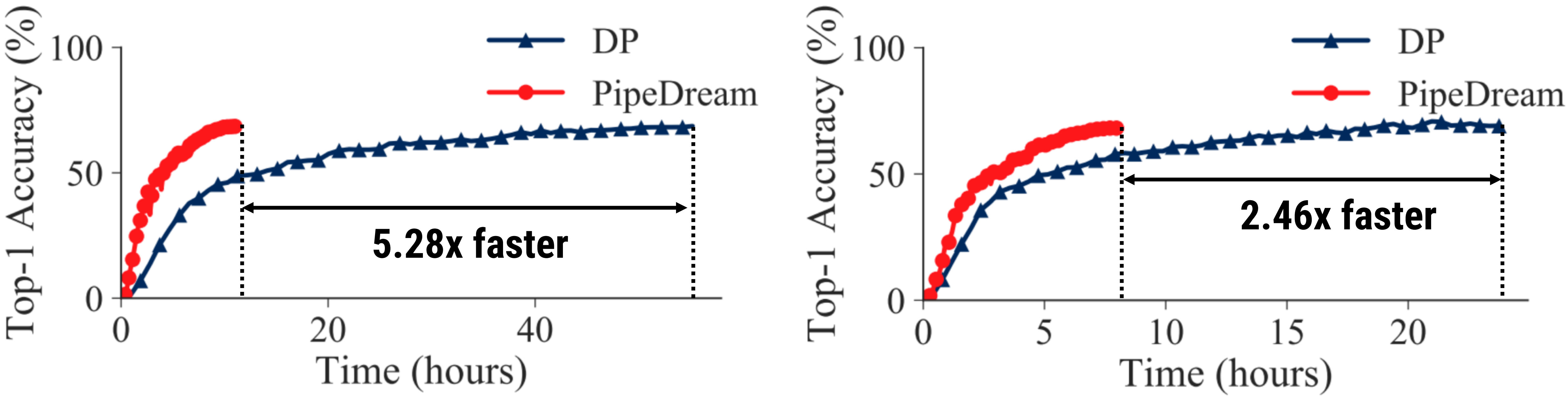
Evaluation Setup

- Integrated PipeDream with PyTorch in ~3000 lines of Python code

- Integrated with PyTorch's communication library
  - NCCL backend for Data Parallelism baselines
  - Gloo backend for PipeDream

- Experiments run on three different server types
  - Cluster A: 4xV100 GPUs, PCIe intra-server, and 10 Gbps inter-server (Azure)
  - Cluster B: 8xV100 GPUs, NVLink intra-server, and 25 Gbps inter-server (AWS)
  - Cluster C: 1xTitan X, and 40 Gbps inter-server (private)

PipeDream > Data Parallelism (DP) end-to-end



(a) Cluster-A. **(b) Cluster-B.**

PipeDream vs. Data Parallelism on Time-to-Accuracy

Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP: Epoch time	Speedup over DP: TTA
Image Classification	VGG-16 [48]	ImageNet [44]	68% top-1	4x4 (A)	15-1	5.28×	5.28×
				2x8 (B)	15-1	2.98×	2.46×
	ResNet-50 [26]	ImageNet [44]	75.9% top-1	4x4 (A)	16	1×	1×
				2x8 (B)	16	1×	1×
	AlexNet [37]	Synthetic Data	N/A	4x4 (A)	15-1	4.92×	N/A
				2x8 (B)	15-1	2.04×	N/A
Translation	GNMT-16 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.46×	2.2×
				4x4 (A)	Straight	2.34×	2.92×
				2x8 (B)	Straight	3.14×	3.14×
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

PipeDream vs. Data Parallelism on Time-to-Accuracy

Experiments on 4 different tasks: image classification, translation, language modeling, video captioning

Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP Epoch time	Speedup over DP TTA
Image Classification	VGG-16 [48]	ImageNet [44]	68% top-1	4x4 (A)	15-1	5.28×	5.28×
				2x8 (B)	15-1	2.98×	2.46×
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Translation	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

PipeDream vs. Data Parallelism on Time-to-Accuracy

With the same number of GPUs, PipeDream up to 5.3x faster than Data Parallelism

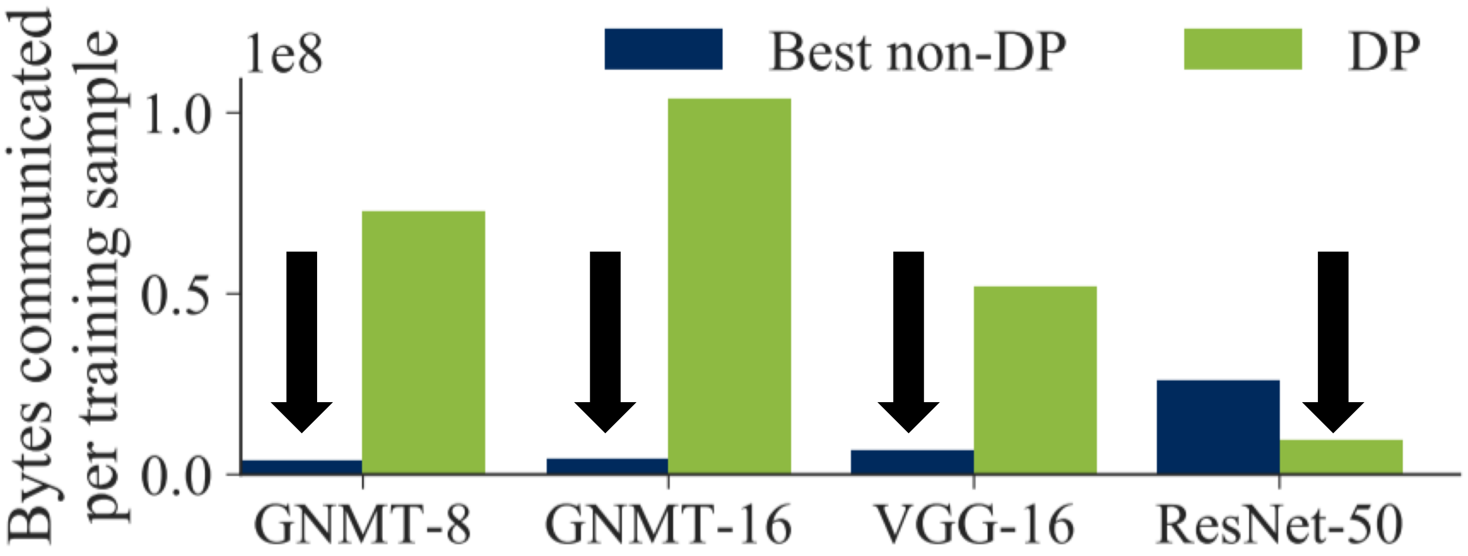
Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP: Epoch time	Speedup over DP: TTA
	VGG-16 [48]	ImageNet [44]	68% top-1	4x4 (A)	15-1	5.28×	5.28×
				2x8 (B)	15-1	2.98×	2.46×
						1×	1×
						1×	1×
						4.92×	N/A
						2.04×	N/A
						1.46×	2.2×
	GNMT-16 [55]	WMT16 EN-De	21.8 BLEU	4x4 (A)	Straight	2.34×	2.92×
Translation				2x8 (B)	Straight	3.14×	3.14×
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

PipeDream vs. Data Parallelism on Time-to-Accuracy

Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP: Epoch time	Speedup over DP: TTA
	VGG-16 [48]	ImageNet [44]	68% top-1	4x4 (A)	15-1	5.28×	5.28×
				2x8 (B)	15-1	2.98×	2.46×
					16	1×	1×
					16	1×	1×
					15-1	4.92×	N/A
					15-1	2.04×	N/A
					Straight	1.46×	2.2×
					Straight	2.34×	2.92×
					Straight	3.14×	3.14×
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

Optimizer recommends a number of different configurations like 15-1, Straight, and a fully data-parallel setup

PipeDream reduces communication overhead



For many models, intermediate activations and gradients order of magnitude smaller than communication with Data Parallelism (DP)

Conclusion

- Model and data parallelism often suffer from **high communication overhead** and **low resource utilization** for certain models and deployments
- PipeDream shows **pipelining** can be used to accelerate DNN training
- Pipelining, when combined with data and model parallelism in a principled way, achieves end-to-end speedups of up to **5.3x**

Code available at
https://github.com/msr-fiddle/pipedream

https://cs.stanford.edu/~deepakn/