AI-Systems

Distributed Deep Learning (Part II)

(294-162)

Amir Gholami & Joseph E. Gonzalez

Acknowledgments

Many slides from Shigang Li, Prof. Kurt Keutzer, Pallas Group

Agenda for Today

- 1:10-1:40: Final Lecture on Parallel Training
- 1:40-1:45: Project Proposal Format
- 2:00-2:45: PC Meeting Discussions
- 2:45-3:00: Break
- 3:00-4:00: Guest Lecture by Michael Houston

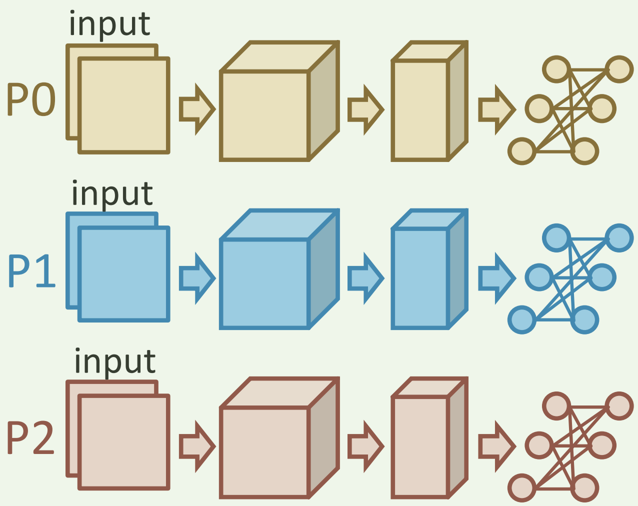
Objectives For Today

- Quick Review of Data & Pipeline Parallelism
- Spatial Parallelism
- Model Parallelism

Distributed Deep Learning: Summary So Far

Parallel and distributed training

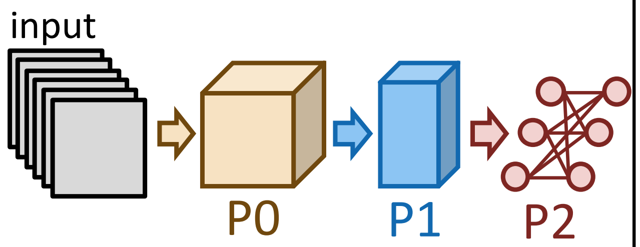
Data parallelism



Pros:
a. Easy to realize

Cons:
a. Not work for large models
b. High allreduce overhead

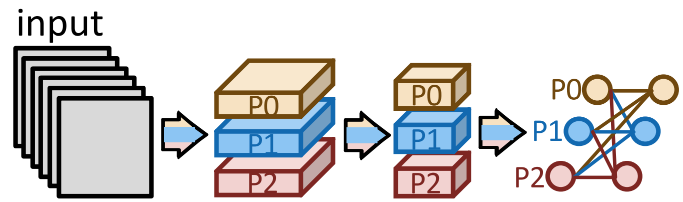
Pipeline parallelism



Pros:
a. Make large model training feasible
b. No collective, only P2P

Cons:
a. Bubbles in pipeline
b. Removing bubbles leads to stale weights

Model parallelism



Pros:
a. Make large model training feasible

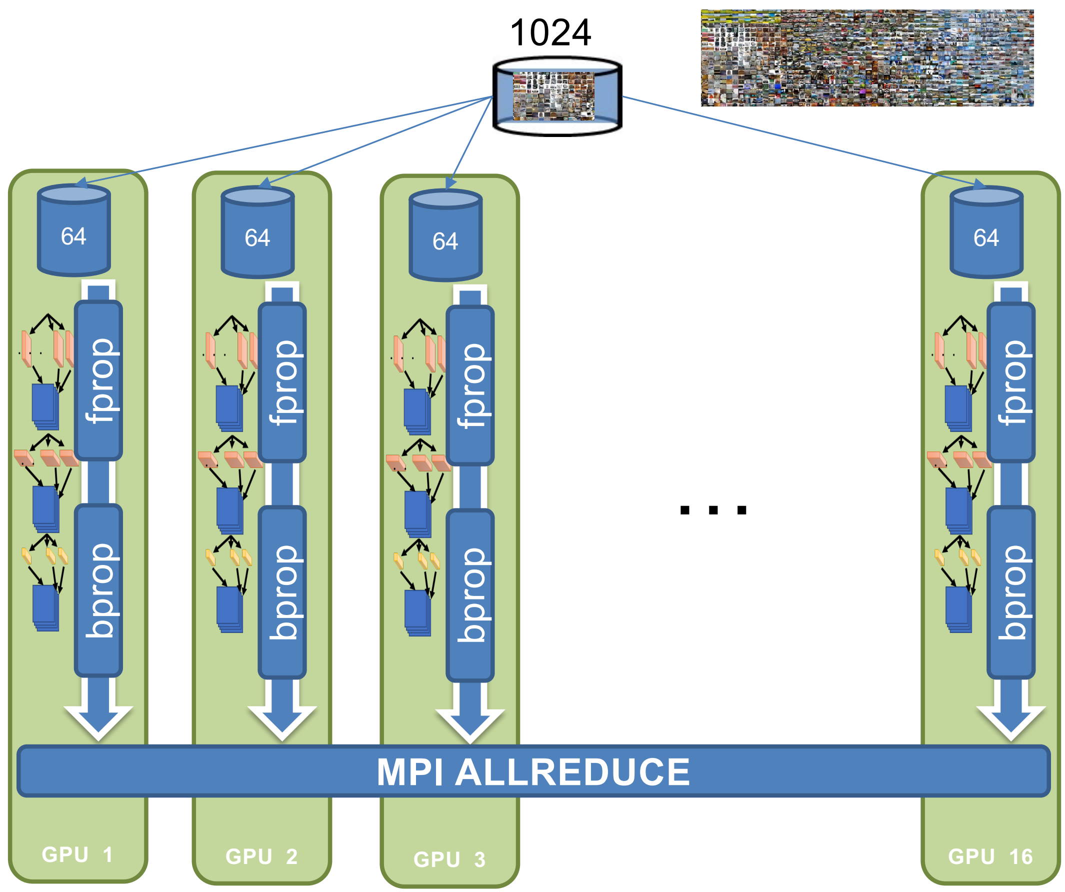
Cons:
b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

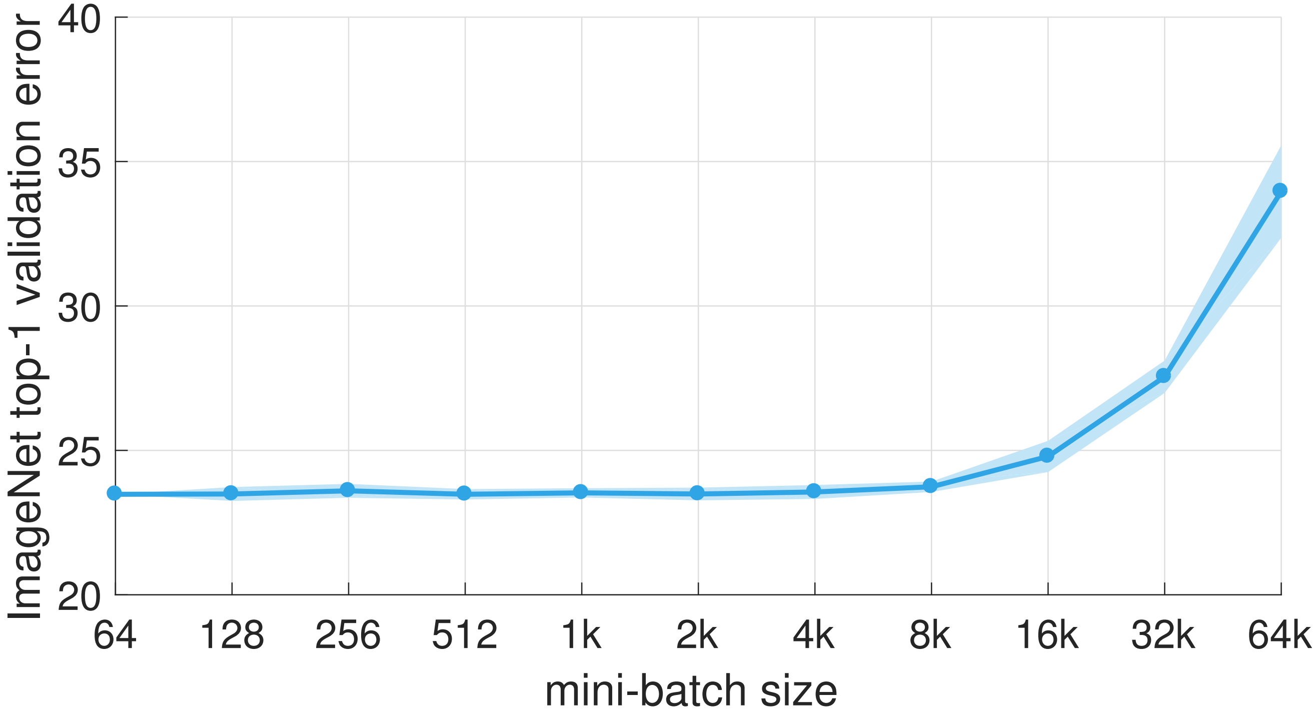
Synchronous Data Parallelism

- Compute the entire model on each processor
- Distribute the batch evenly across each processor:
  - 1024 batch distributed over 16 PEs: 64 images per GPU
- Communicate gradient updates through **allreduce**

$$w^1 = w^0 - \frac{\alpha}{B} \boxed{\sum_{i=1}^{B}} \frac{\partial \mathcal{J}(w^0)}{\partial w}$$



Generalization Gap Problem



Larger batch sizes harm generalization performance.

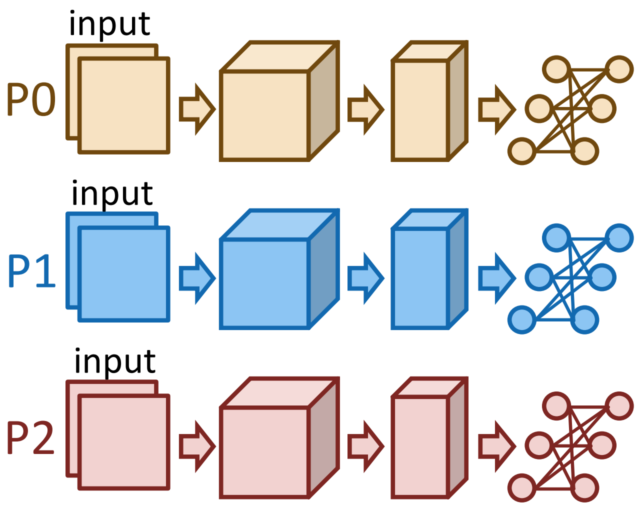
Goyal, Priya, et al. "Accurate, large minibatch SGD: Training imagenet in 1 hour." arXiv preprint arXiv:1706.02677 (2017).

Data Parallelism Summary

- An efficient parallel training method where the comm time is independent of processors with ring allreduce
- Very easy to implement. Only requires allreduce operation before updating parameters
- Very challenging to scale. Using large batch training is not an option as it hurts generalization performance.
  - Existing solutions often require a lot of tuning (outside of ResNet-50 on ImageNet)
- Does not work for large models such as GPT-3 which are too large to fit in one GPU

Parallel and distributed training

Data parallelism



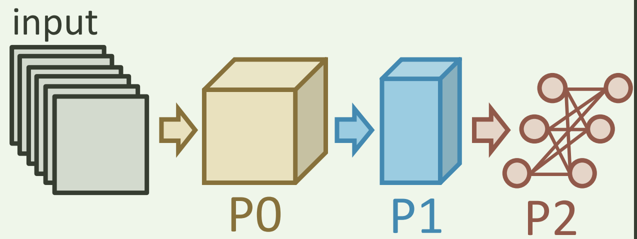
Pros:

a. Easy to realize

Cons:

a. Not work for large models
b. High allreduce overhead

Pipeline parallelism



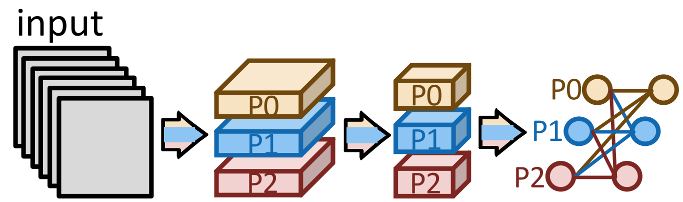
Pros:

a. Make large model training feasible
b. No collective, only P2P

Cons:

a. Bubbles in pipeline
b. Removing bubbles leads to stale weights

Model parallelism



Pros:

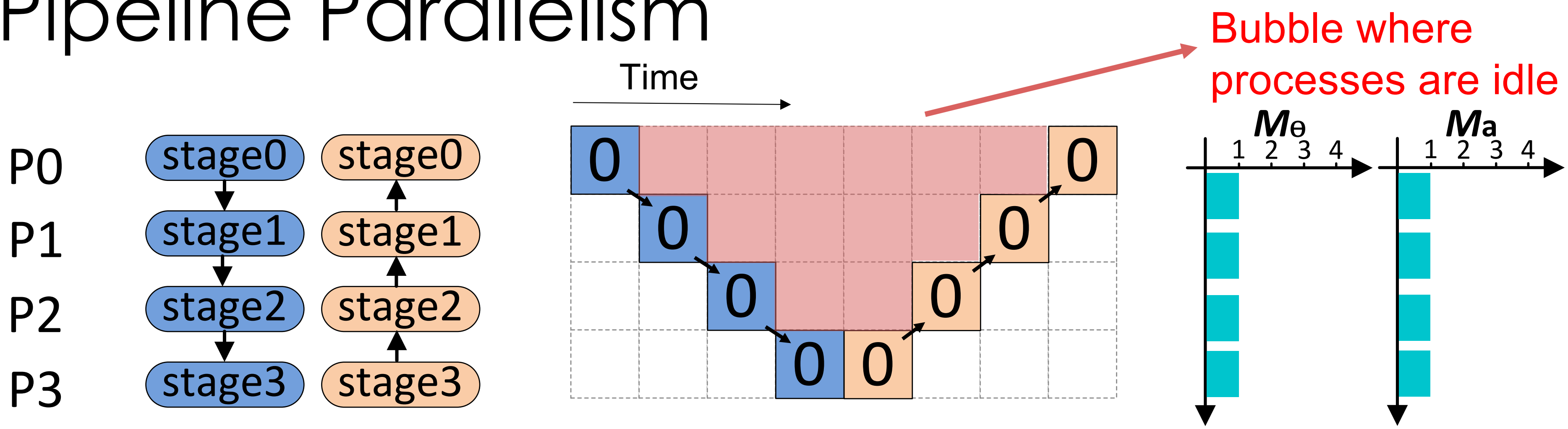
a. Make large model training feasible

Cons:

b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

Pipeline Parallelism



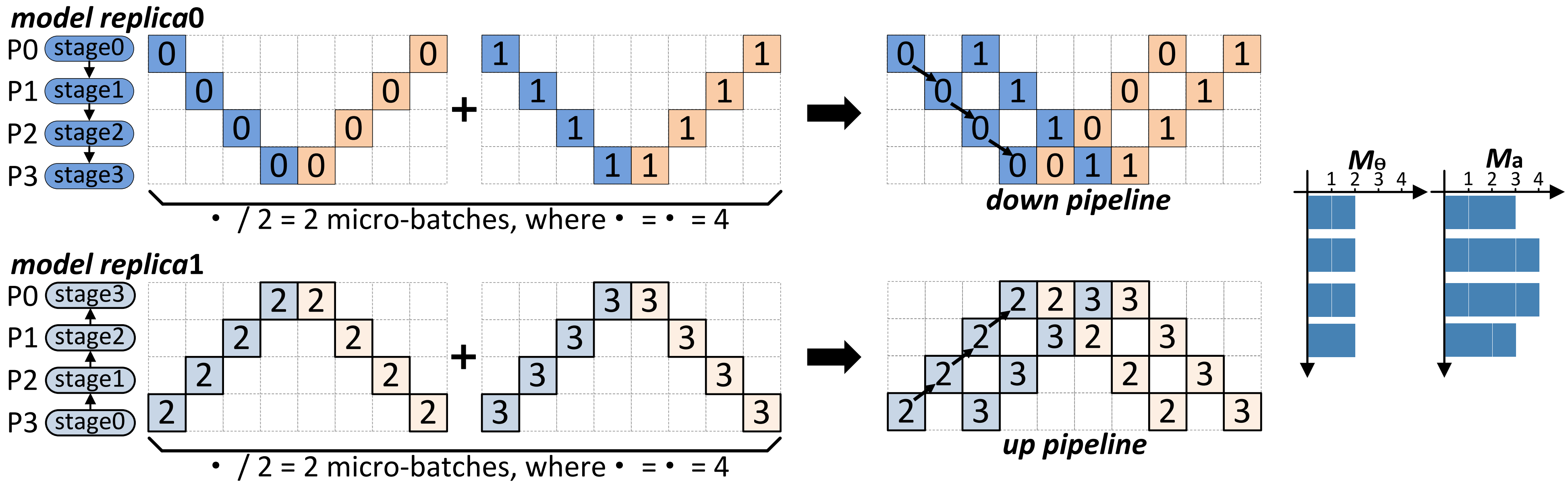
Bubble

x x Forward and backward passes of ***model replica*0** for micro-batch **x**

M_θ Memory consumption for the weights

M_a Memory consumption for the activations

Chimera: Bidirectional Pipeline



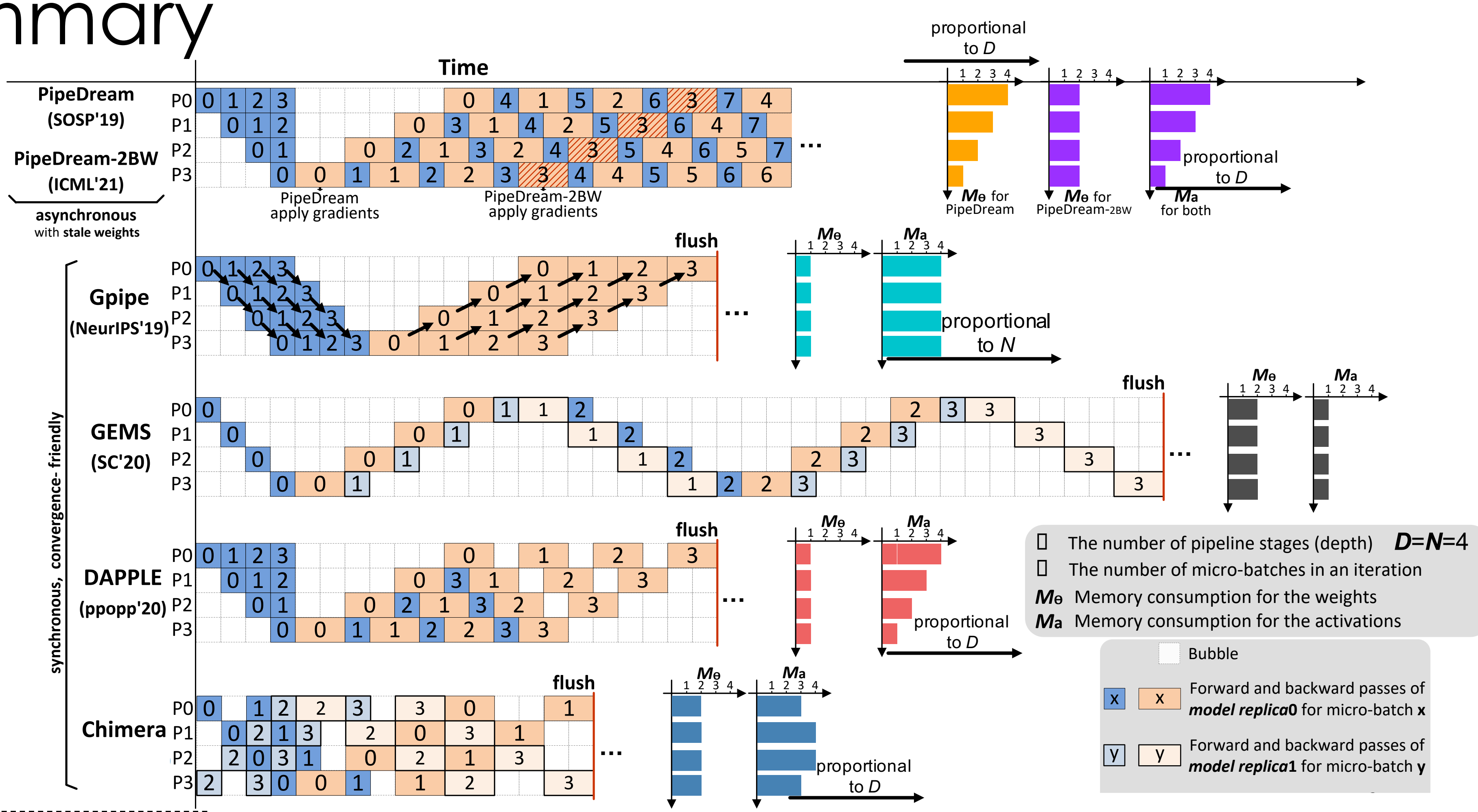
- ➢ Big idea: Replicate the model to the other processes so that we can do forward pass in two directions

x x Forward and backward passes of ***replica*0**

y y Forward and backward passes of ***replica*1**

Slide: Courtesy of Shigang Li

Summary



Slide: Courtesy of Shigang Li

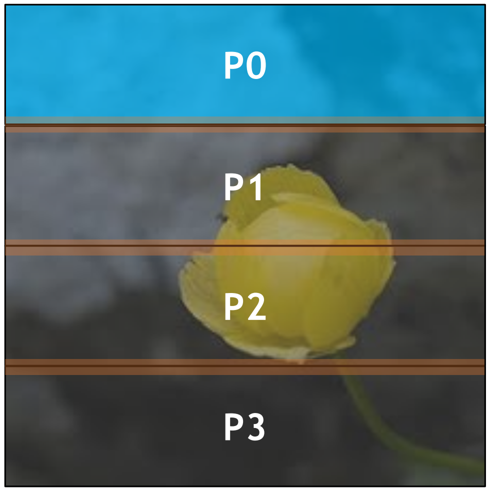
Pipeline Parallelism Summary

- More efficient for large scale training to thousands of processes where point-to-point communication is much cheaper than collective operations such as allreduce or all-gather
- Slightly more involved algorithm than data parallel method but with the advantage of only requiring point to point communication
- Requires special handling of bubble that results in idle processes

Spatial Parallelism

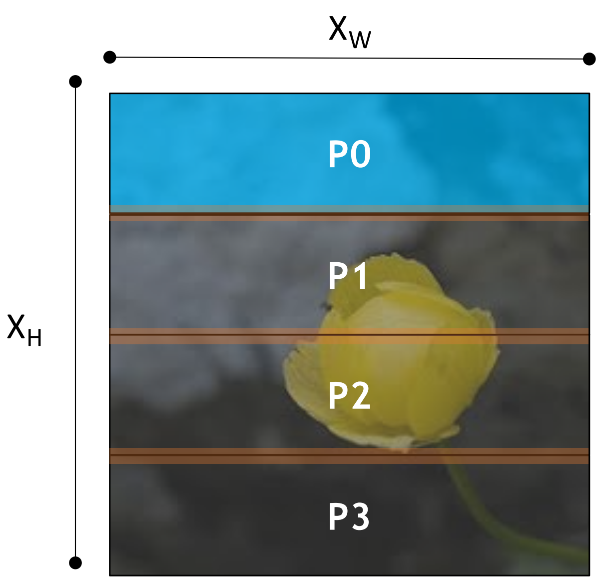
Spatial Parallel Training

- The general idea is to break the input into smaller pieces and distribute the work among different processors
  - Need to exchange boundary points for spatial convolutions



Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018

Communication Complexity



$$T_{comm}(domain) = \sum_{i=0}^{L} (\alpha + \beta B X_W^i X_C^i k_h^i / 2)$$

Exchanging horizontal pixels

$$+ \sum_{i=0}^{L} (\alpha + \beta B Y_W^i Y_C^i k_w^i / 2)$$

Exchanging vertical pixels

$$+2 \sum_{i=0}^{L} \left(\alpha \log(P) + \beta \frac{P-1}{P} |W_i| \right)$$

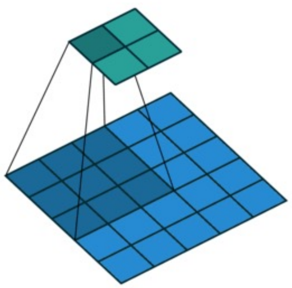
All reduce Cost
(same as before)

Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018.
Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

Useful for High Resolution Training

- Domain parallel scaling on V100 GPUs
  - 3x3 Conv, Batch=32, Channel=64

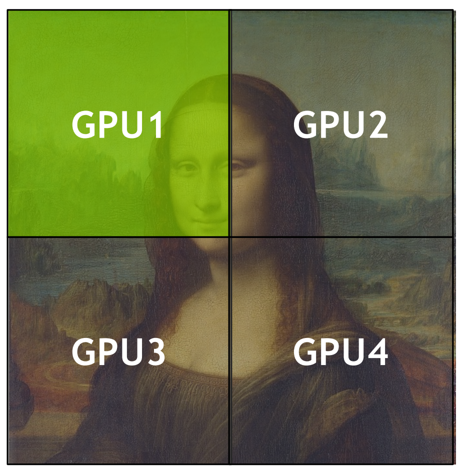
Resolution	GPUs	Fwd. wall-clock	Bwd. wall-clock
128×128	1	2.56 ms (1.0×)	6.63 ms (1.0×)
	2	1.52 ms (1.7×)	3.50 ms (1.9×)
	4	**1.23 ms (2.1×)**	**2.33 ms (2.8×)**
256×256	1	10.02 ms (1.0×)	26.81 ms (1.0×)
	2	5.34 ms (1.9×)	11.79 ms (2.3×)
	4	**3.11 ms (3.2×)**	**6.96 ms (3.9×)**
512×512	1	45.15 ms (1.0×)	126.11 ms (1.0×)
	2	20.18 ms (2.2×)	60.15 ms (2.1×)
	4	**10.65 ms (4.2×)**	**26.76 ms (4.7×)**



Peter Jin, Boris Ginsburg, and Kurt Keutzer. "Spatially Parallel Convolutions" ICLR Workshop Track, 2018
Figure from: Dumoulin, V., Visin, F.. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016.

Spatial Parallelism Summary

- A little harder to implement since you need to exchange the boundary points
- Only effective for high resolution input data
  - Limits the number of processors that can be effectively utilized



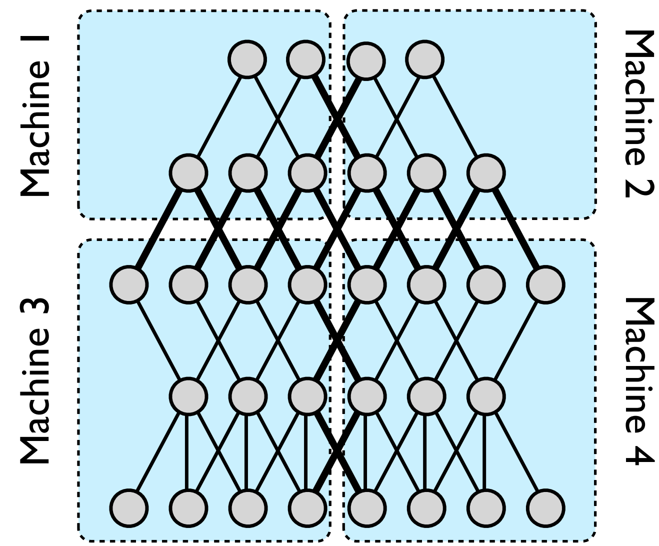
Model Parallelism

AKA Operator Parallelism

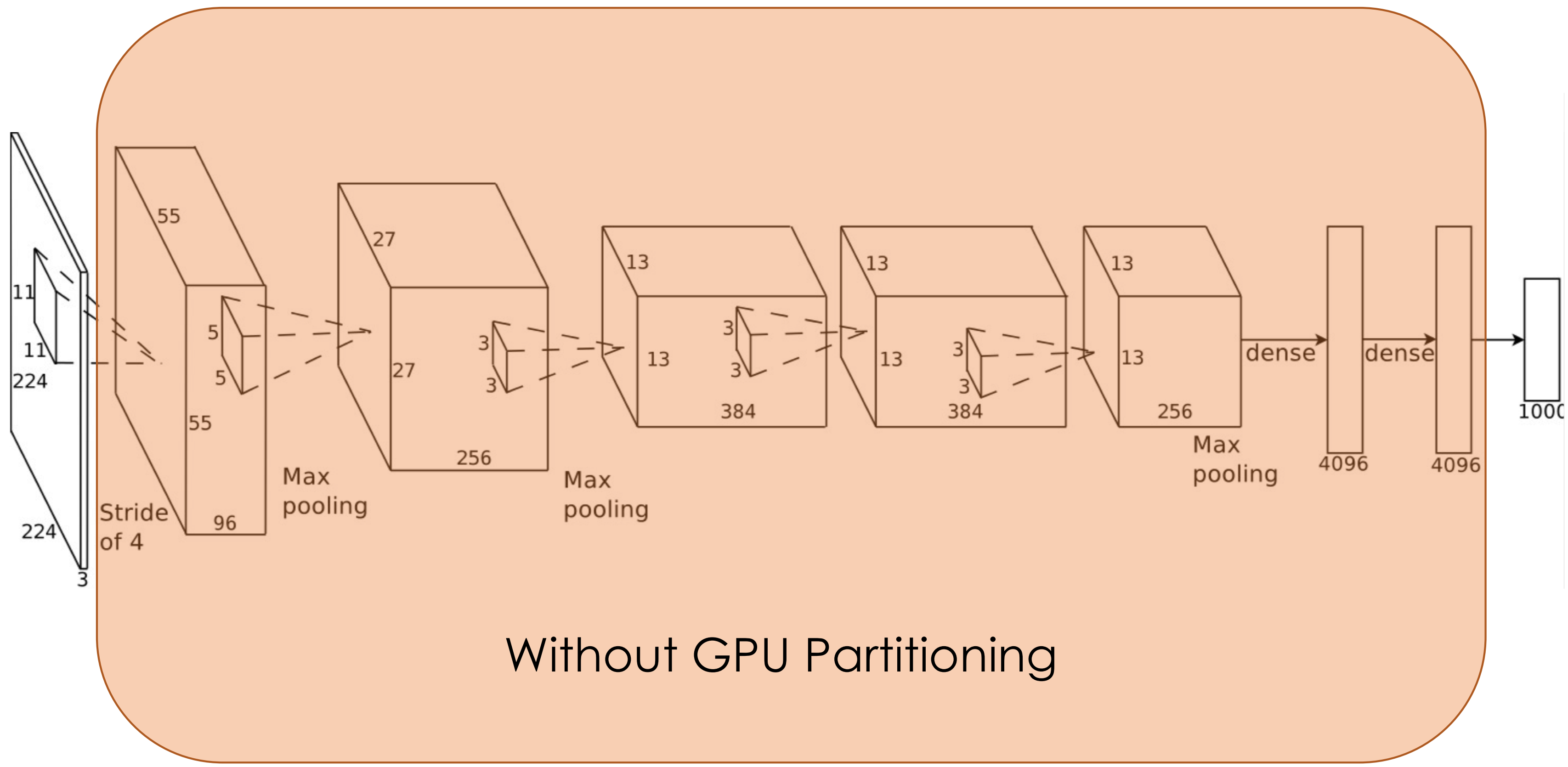
Model Parallelism

Divide the model across machines and replicate the data.

- Supports large models and activations
- Requires communication within single evaluation
- How to best divide a model?
  - Split across layers
    - Only one set of layers active a time → poor work balance
    - This is basically pipeline parallelism
  - Split individual layers
    - which dimension?
      - Weights or spatial → depends on operation

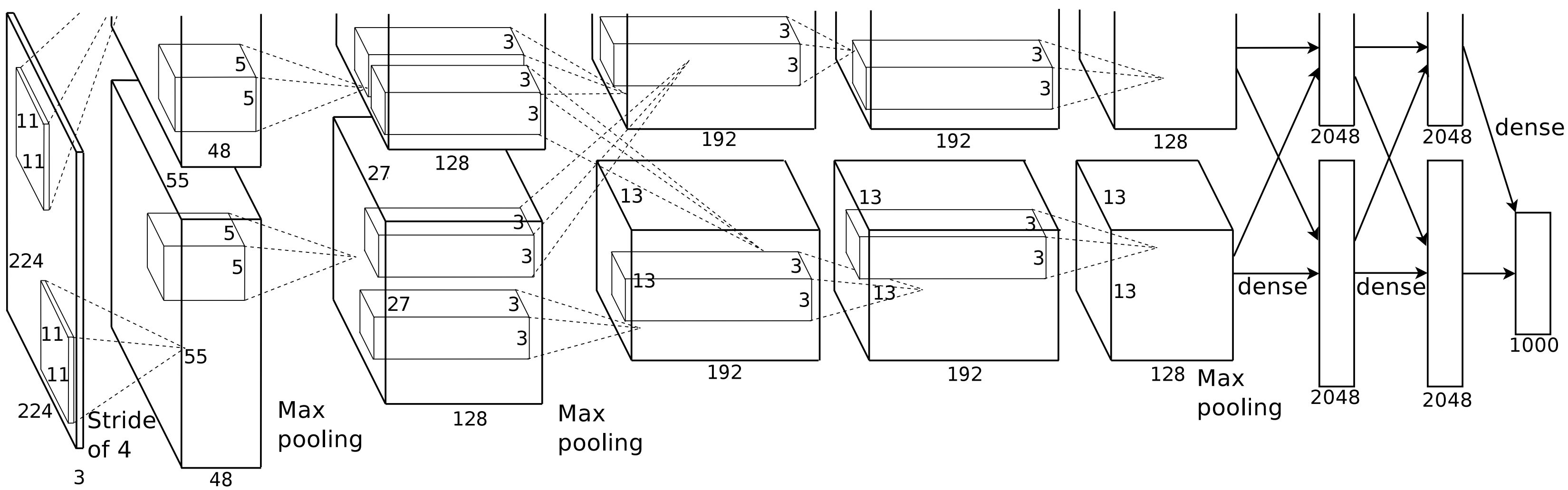


The AlexNet Architecture



The **Actual** AlexNet Architecture

from the paper



Training on Multiple GPUs

- Limited by GPU **memory** using Nvidia GTX 580 (3GB RAM)
  - 60M Parameters ~ **240 MB**
  - Need to cache activation maps for backpropagation
    - Batch size = 128
    - 128 * (227*227*3 + 55*55*96*2 + 96*27*27*2 + 256*27*27*2 + 256*13*13*2 + 13*13*384*2 + 256*13*13 + 6*6*256 + 4096 + 4096 + 1000) *4 Bytes ~ **782MB Activations**
    - That is assuming no overhead and single precision values
- Tuned splitting across GPUS to balance communication and computation

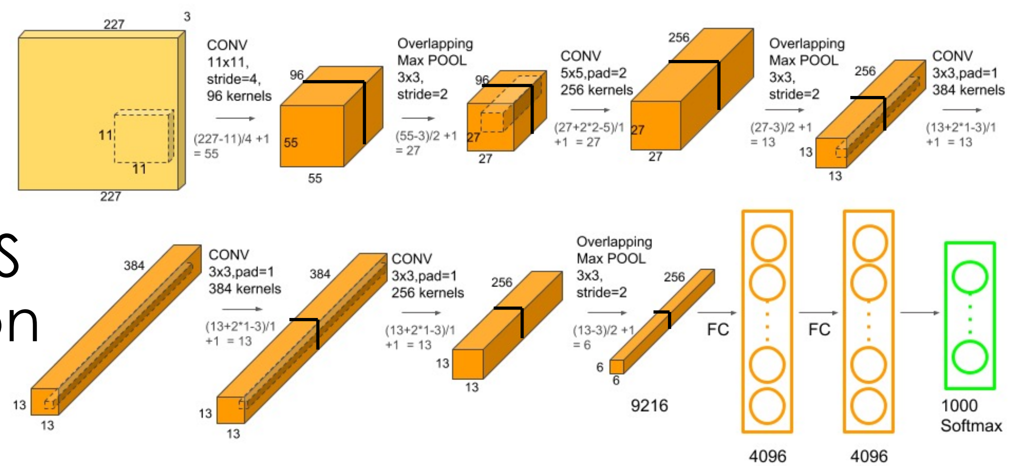


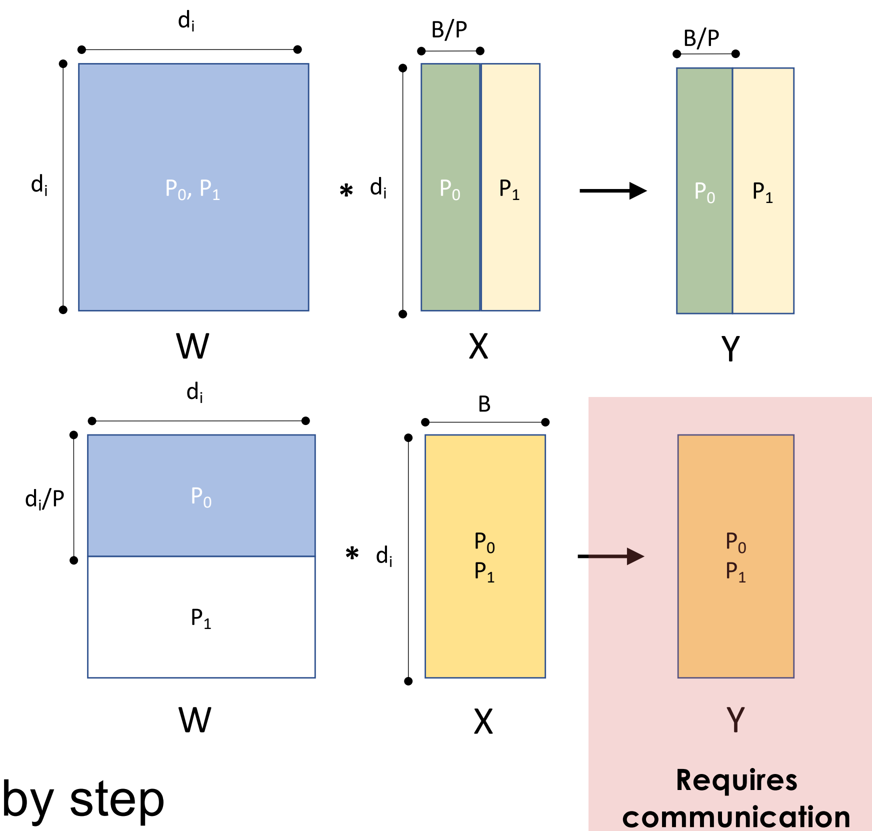
Image from https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

Model Parallelism: Comm Analysis

It helps to think of the operations in matrix form. Consider an FC layer

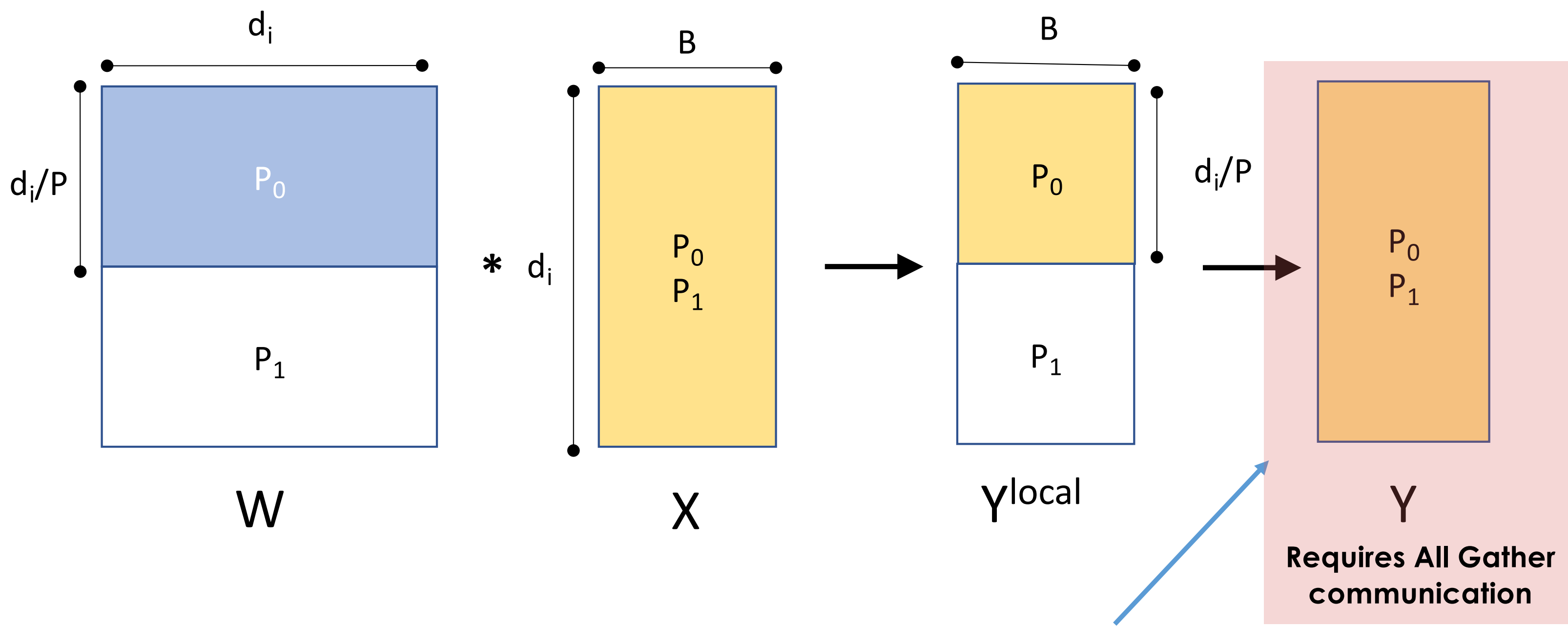
Data Parallelism: Partition input across different Processors (batch dimension)

Model Parallelism: Partition weights across different Processes (W dimension)



Let’s discuss the communication details, step by step

Comm Analysis: Forward Pass

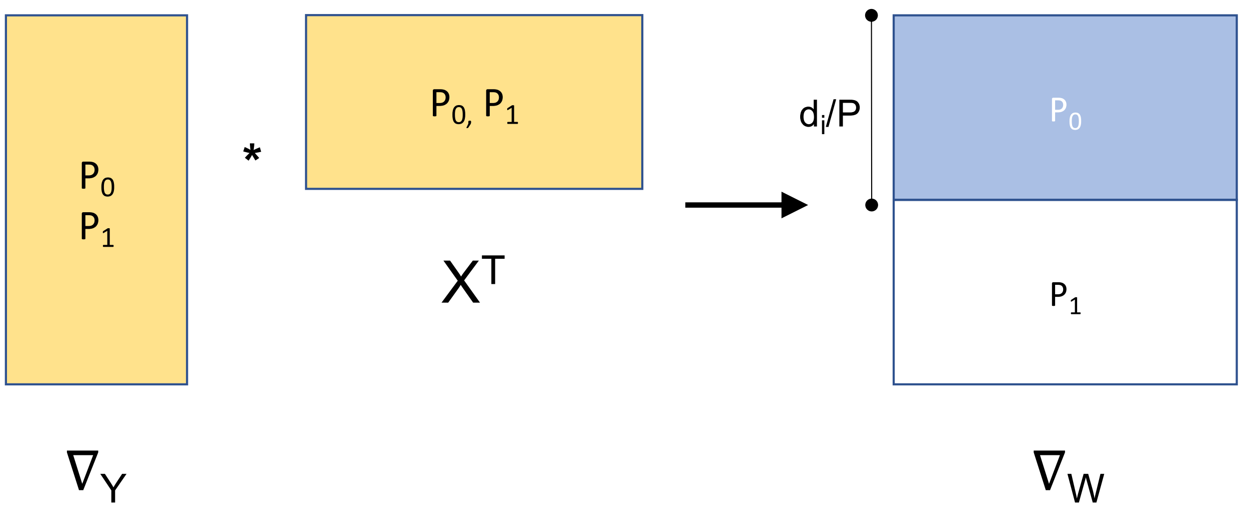


$$\sum_{i=1}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

- Requires an all gather communication so that all processes get each others activation data
- Same cost as all reduce without the 2x factor

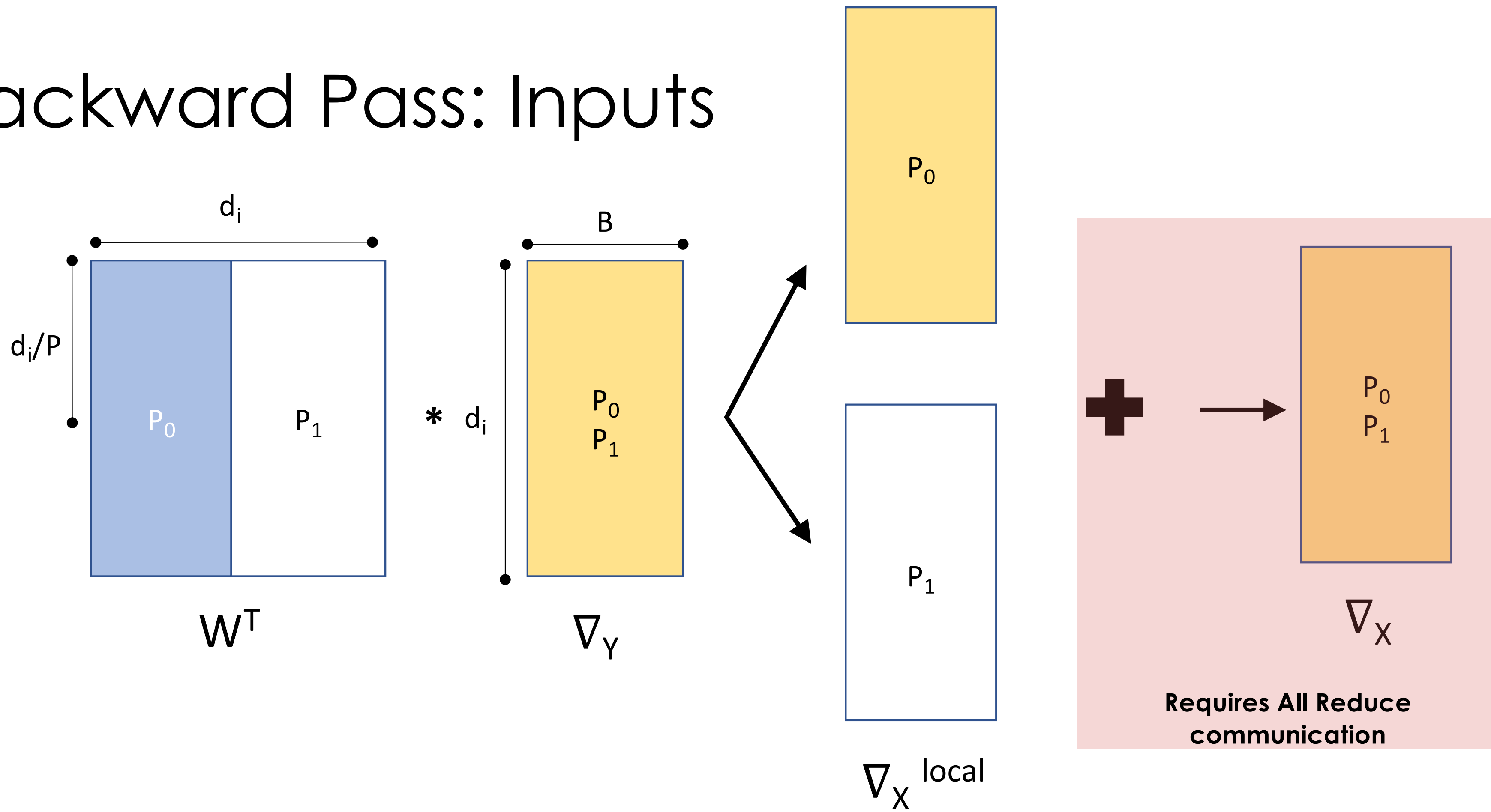
* Ignoring latency term for notational simplicity

Backward Pass: Weights



- No communication needed as every processor only needs the gradient of its own parameters
  - This makes model parallelism very effective for cases where the model size is large

Backward Pass: Inputs



- Aggregating input gradient requires an allreduce operation

$$2\sum_{i=2}^{L}\left(\beta(P-1)\frac{Bd_i}{P}\right)$$

Comm Complexity Analysis

In Model Parallelism we need two forms of communication:

1. All Gather operation so that all processors get all the activations
2. All reduce operation for backpropagating activation gradients

$$T_{comm}(model) = \underbrace{\sum_{i=1}^{L} \left(\beta (P-1) \frac{Bd_i}{P} \right)}_{\text{All Gather}} + 2 \underbrace{\sum_{i=2}^{L} \left(\beta (P-1) \frac{Bd_i}{P} \right)}_{\text{All Reduce}}$$

Model vs Data Parallelism?

- When does it make sense to use Model vs Data Parallelism?

$$T_{comm}(model) = \sum_{i=1}^{L} \left(\beta(P-1)\frac{Bd_i}{P} \right) + 2\sum_{i=2}^{L} \left(\beta(P-1)\frac{Bd_i}{P} \right)$$

$$T_{comm}(data) = \sum_{i=1}^{L} \left(\beta(P-1)\frac{d_i^2}{P} \right)$$

- Model parallelism reduces the quadratic complexity of d_i
  - It is useful for layers with very large weights $d_i >> 1$
- It makes sense to use an integrated/hybrid data and model parallelism

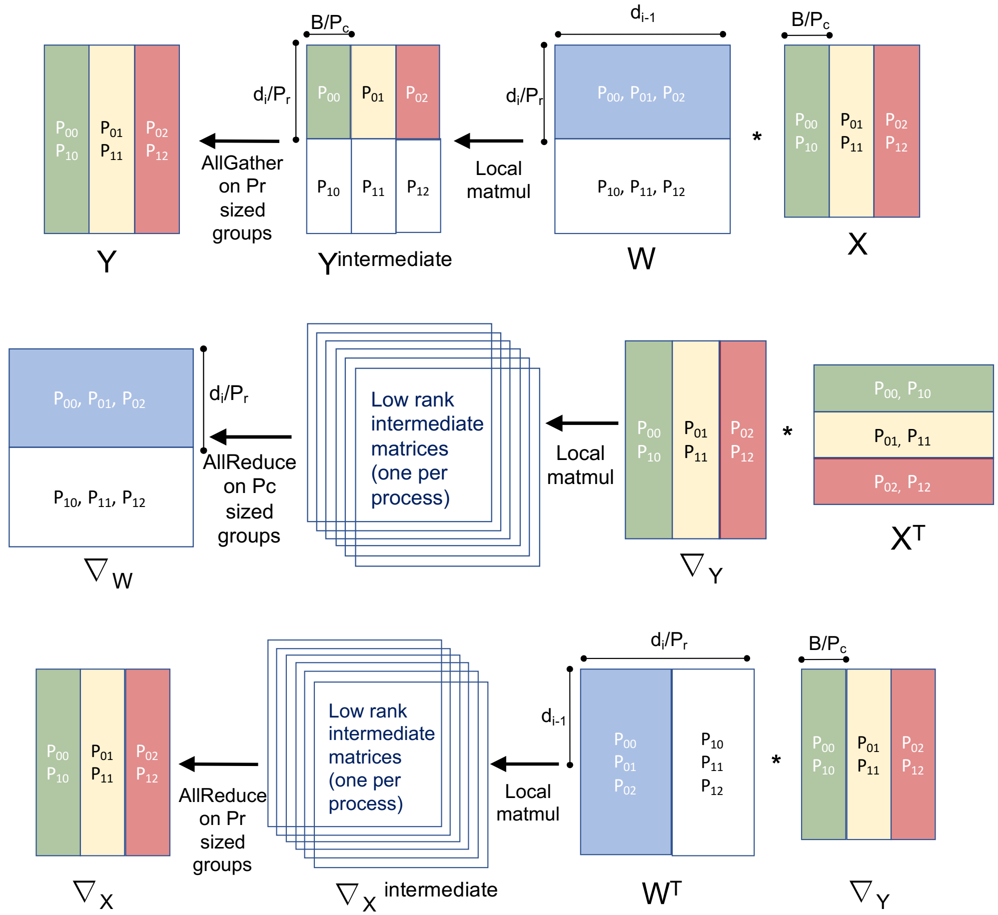
Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

Model Parallelism Summary

- Has better comm complexity for **large** FC layers than Data parallel approach
- Makes training large models feasible by breaking it into smaller parts
- However, requires **blocking collective communication** during **both** forward pass (all gather), as well as backwards pass (all reduce)
- Slightly **harder to implement** than data/pipeline parallel

Integrated Model and Data Parallelism

For a linear graph we can find the optimal hybrid method for analyzing the communication complexity, coupled with hardware utilization [1]



Processes are 2D indexed: **$P = P_r \times P_c$**

[1] Gholami, Amir, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated model, batch, and domain parallelism in training neural networks." SPAA, 2018.

General Hybrid Methods

For a general computational graph we need to decide on:

- How many processes to assign for DP
- Which axes to break the model: operator vs pipeline
- How to efficiently map the GPUs to the resulting execution graph
- …

For a general non-linear graph this leads to a combinatorically large search space

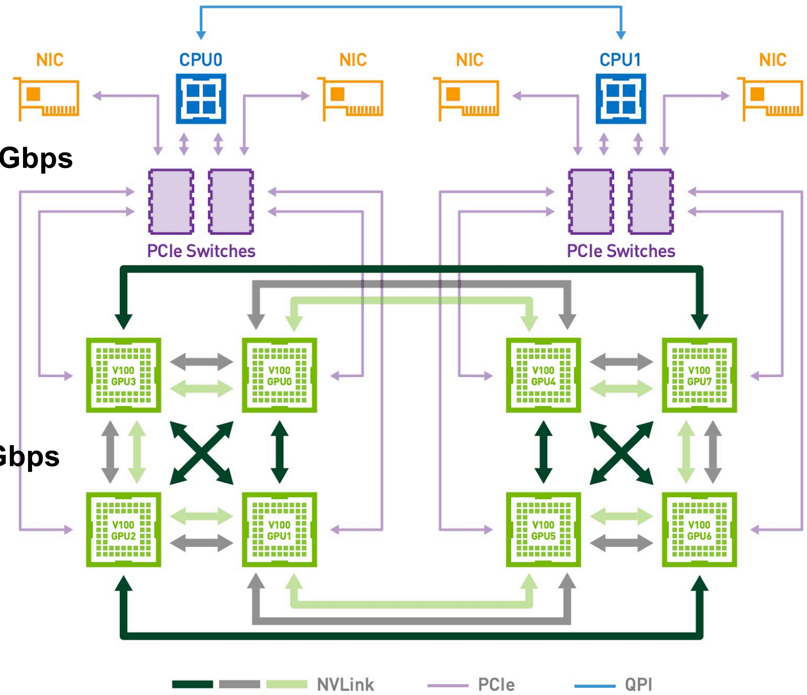
Hybrid Methods: Alpa

Pipeline Parallelism*
(less communication)
*aka inter-op parallelism

10-50 Gbps

Model Parallelism*
(more communication)
* aka intra-op parallelism

400 Gbps



- Organize inter- and intra-op parallelism as a two-level hierarchical space
- Design algorithms to derive optimal plans at each level

Slide: Courtesy of Lianming Zheng
Zheng, Lianmin, et al. "Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning." arXiv preprint arXiv:2201.12023 (2022).

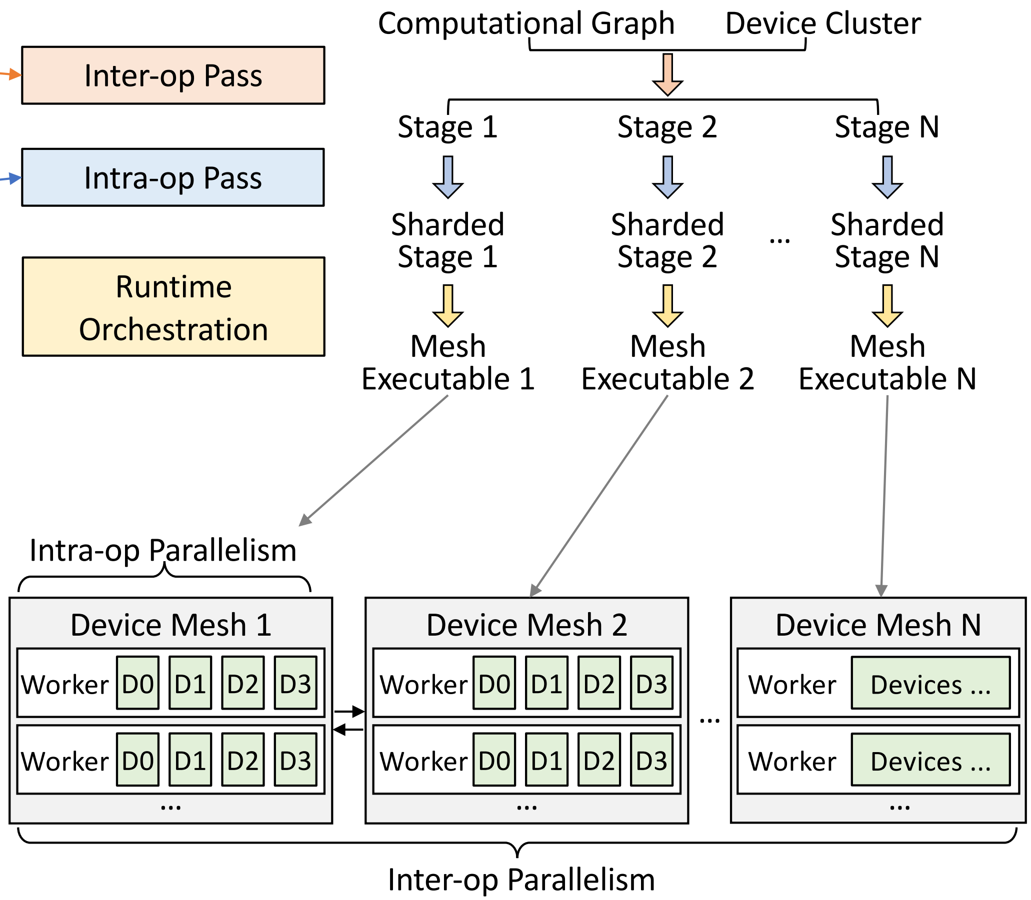
Alpa: Architecture Overview

locally optimal solution solved by DP

locally optimal solution solved by ILP

Compiler

Runtime



Slide: Courtesy of Lianming Zheng
Zheng, Lianmin, et al. "Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning." arXiv preprint arXiv:2201.12023 (2022).

Project Proposals