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

- 1:10-2:00: Preliminary Lecture on Parallel Training
- 2:00-2:45: PC Meeting Discussions
- 2:45-3:00: Break
- 3:00-4:00: Guest Lecture by MSFT DeepSpeed Team

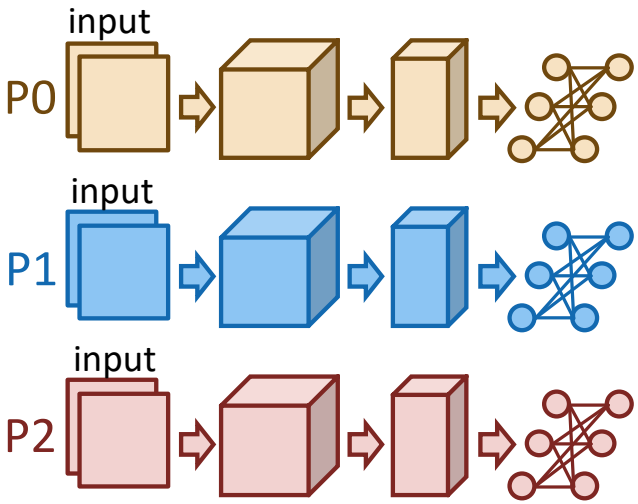
Distributed Deep Learning

Objectives For Today

- Challenges with Data Parallel Training
- Model Parallelism
- Pipeline Parallelism

Parallel and distributed training

Data parallelism



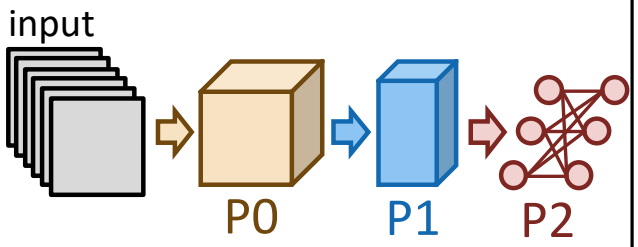
Pros:

a. Easy to realize

Cons:

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

Pipeline parallelism



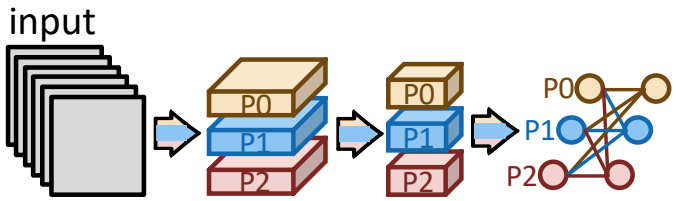
Pros:

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:

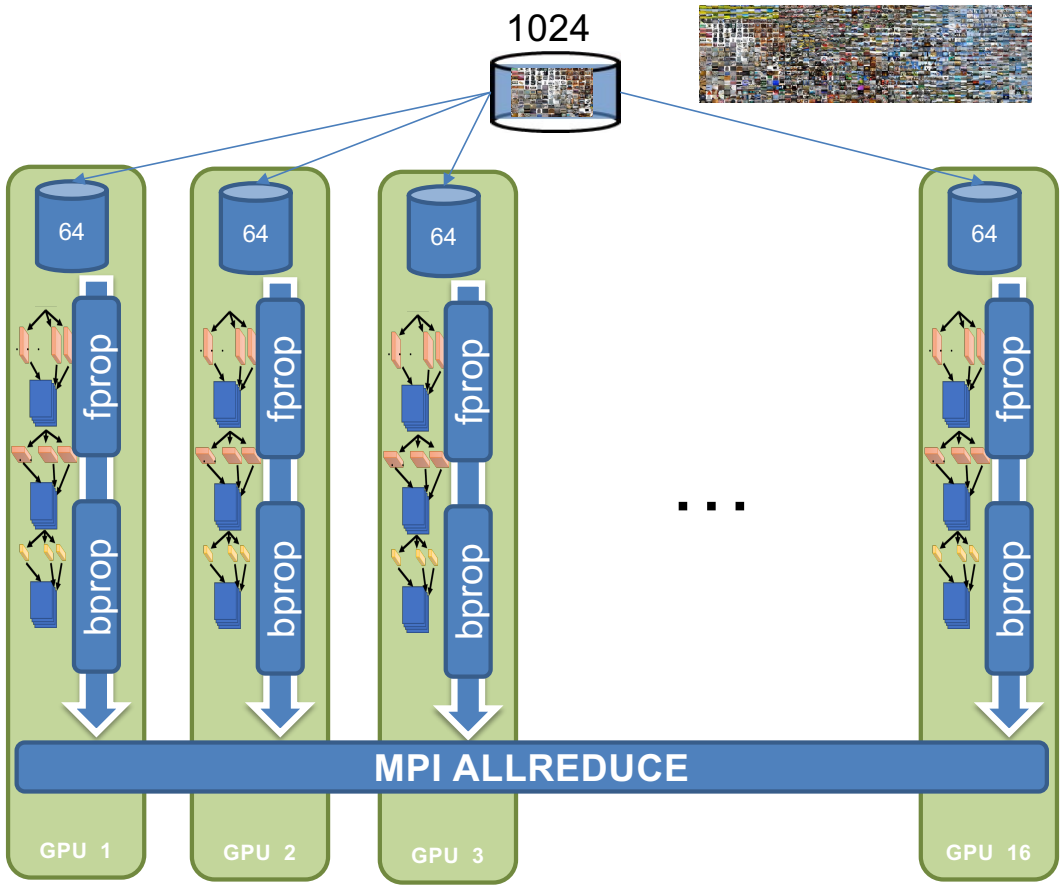
b. Communication for each operator (or each layer)

Slide: Courtesy of Shigang Li

Synchronous Data Parallelism

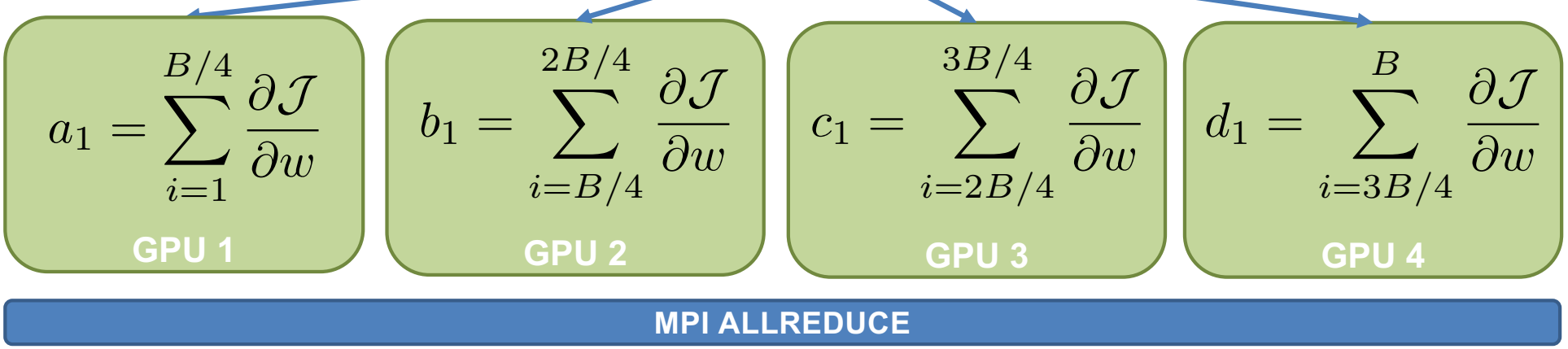
- Compute the entire model on each processor
- Distribute the batch evenly across each processor:
  - 1024 batch distributed over 16 PEs: 64 images per GPU
- Communicate gradient updates through **allreduce**

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



All Reduce

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



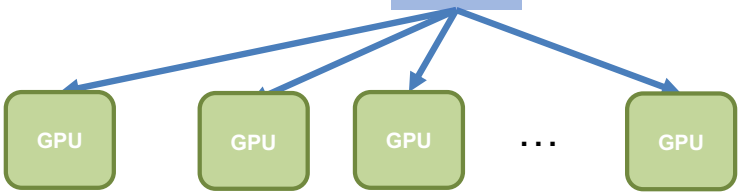
$$\sum_{i=1}^{B} \frac{\partial \mathcal{J}}{\partial w} = a_1 + b_1 + c_1 + d_1$$

Data Parallel Training Complexity Analysis

- Question: Comm time of ring allreduce is independent of the number of processors. So what limits scalability?

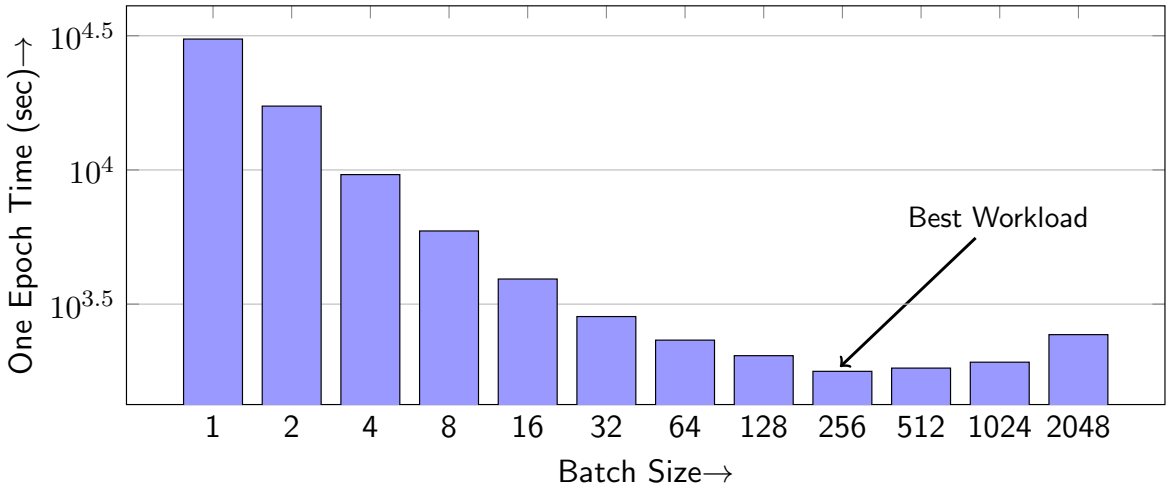
$$T_{comm}(batch) = 2\sum_{i=0}^{L}\left(\alpha(P-1) + \beta\frac{P-1}{P}|W_i|\right)$$

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



Limits of Data Parallel Scaling

- The maximum limit of processors that you can use is P=B
- But this often leads to very low utilization of the hardware and would not yield any speed up

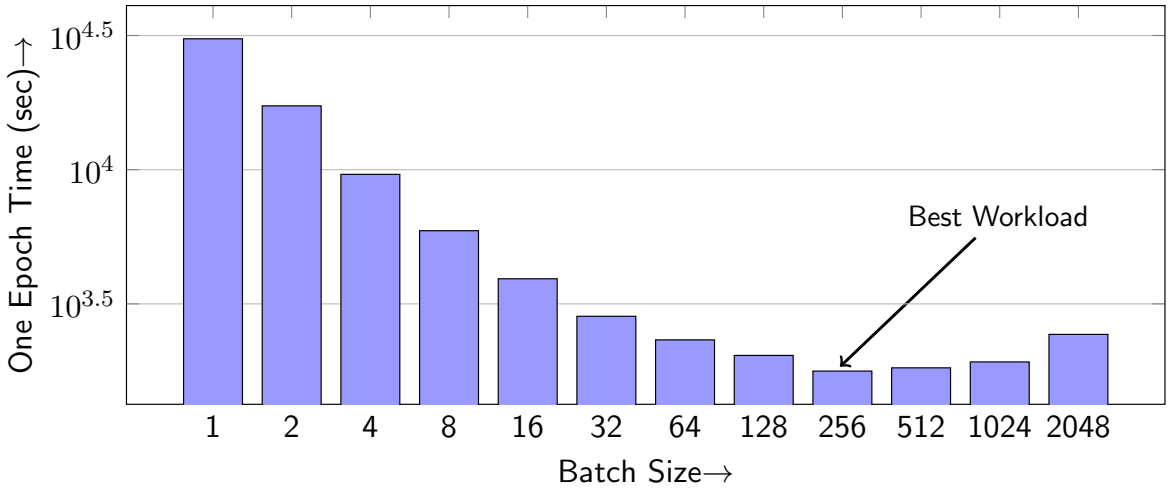


One epoch training time of AlexNet computed on an Intel KNL system

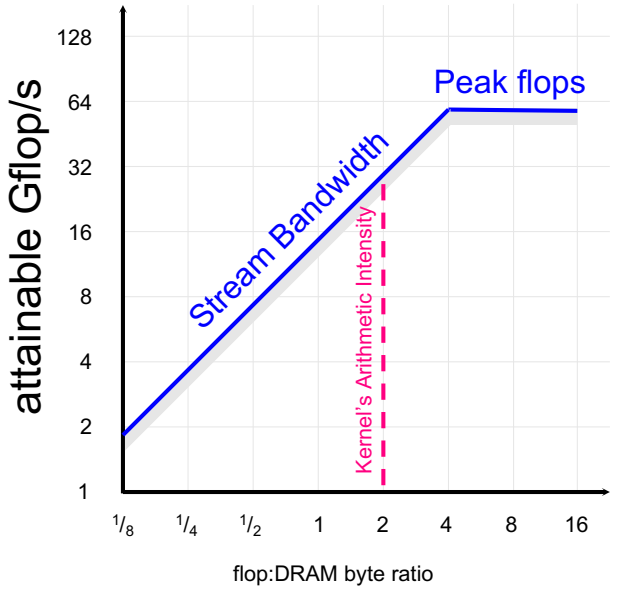
- Why?
  - Roofline model?

Limits of Data Parallel Scaling

- The maximum limit of processors that you can use is P=B
- But this often leads to very low utilization of the hardware and would not yield any speed up

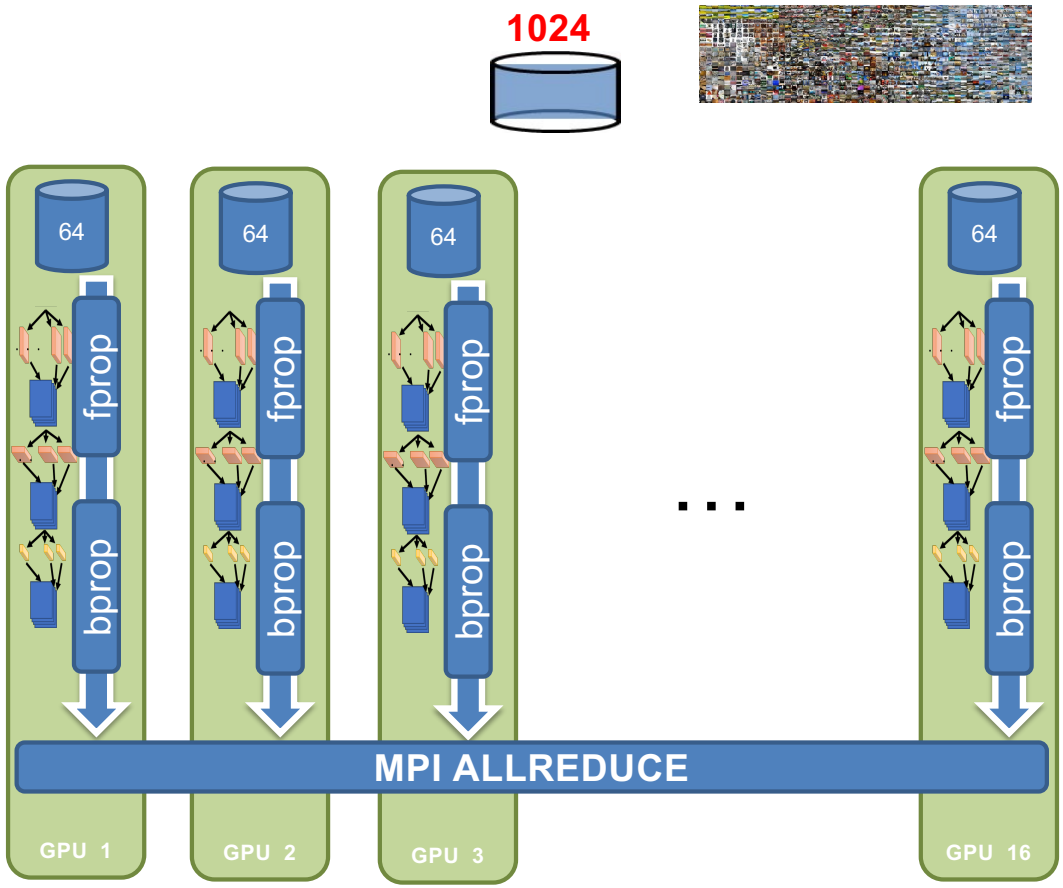


One epoch training time of AlexNet computed on an Intel KNL system

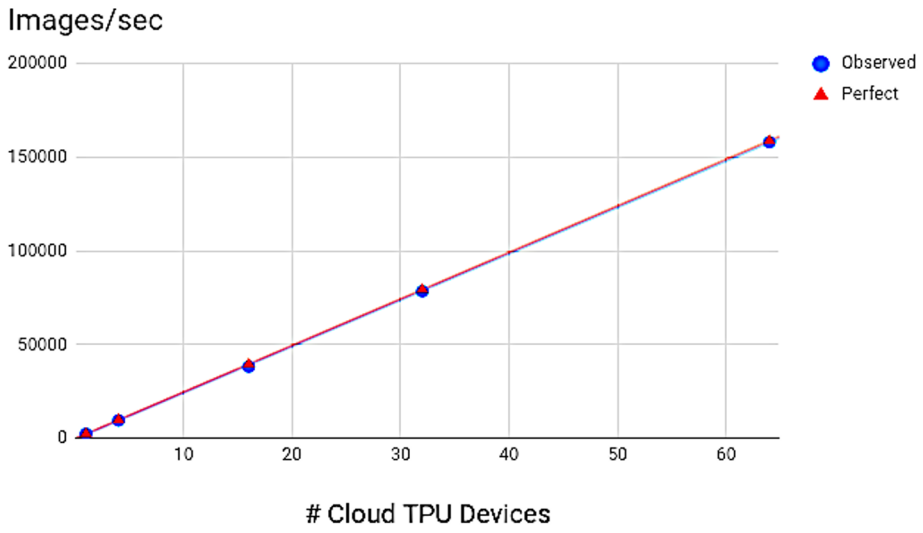


Scaling Data Parallel Training

If we want to keep scaling synchronous SGD then we have to keep increasing the batch size.



Naively increasing Batch size leads to perfect results but …



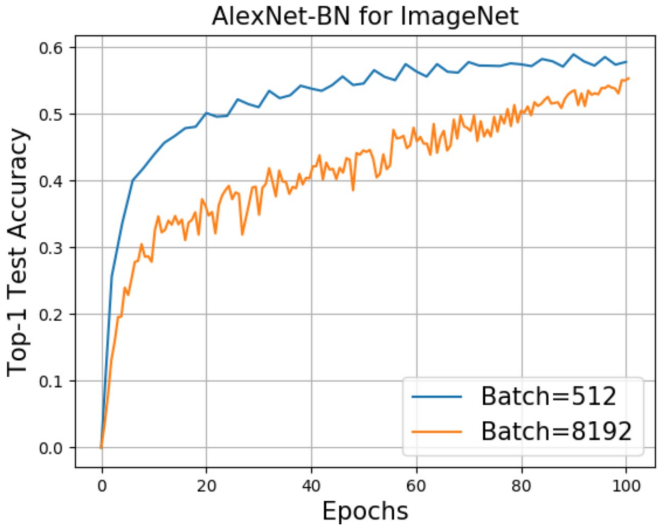
$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

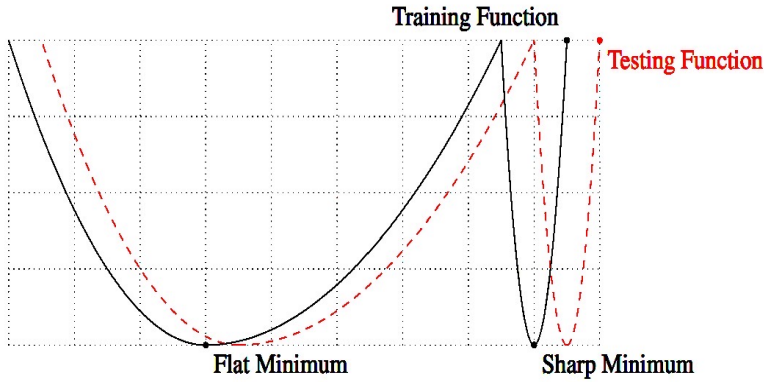
Convergence
Machine Learning
Property

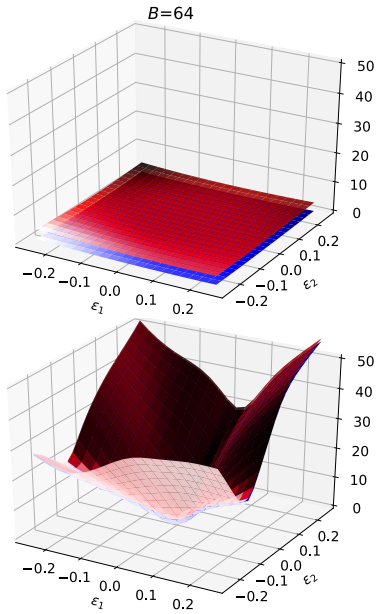
Throughput
System
Property

Problems with Large Batch Training

- Larger Batch leads to sub-optimal generalization
- A common belief is that large batch training gets attracted to "sharp minimas"

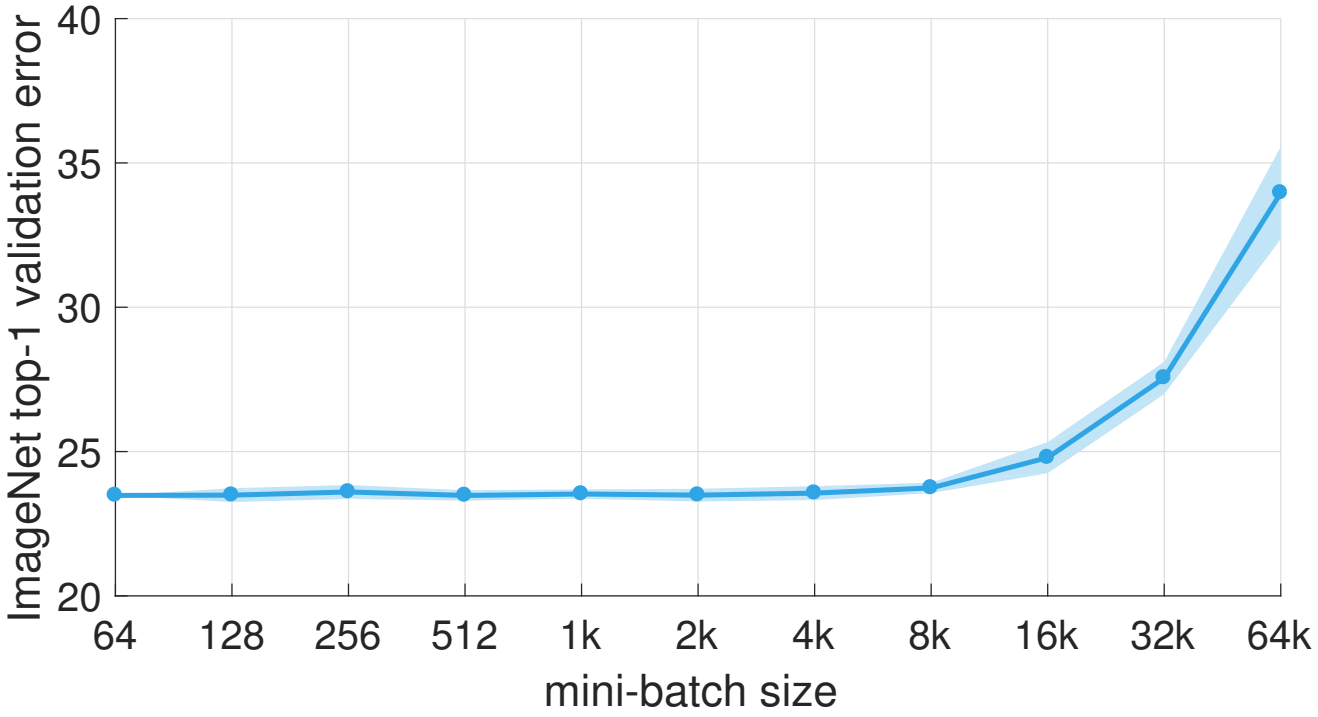






Keskar et al., On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, ICLR'16.
Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. Mahoney. Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18.
Ginsburg, Boris, Igor Gitman, and Yang You. "Large Batch Training of Convolutional Networks with LARS." arXiv:1708.03888, 2018.

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).

Bigger isn't Always Better

- Motivation for larger batch sizes
  - More opportunities for parallelism → but is it useful?
  - Recall (1/n variance reduction):

$$\frac{1}{n}\sum_{i=1}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

- Is a variance reduction helpful?
  - Does it affect the final prediction accuracy?

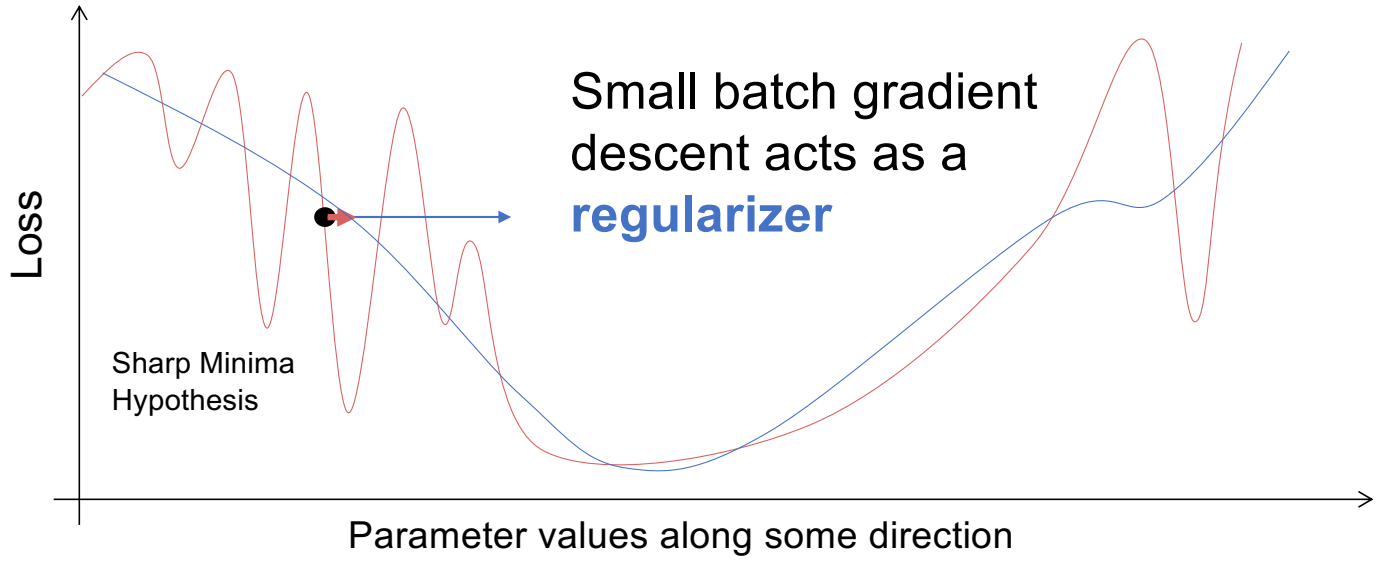
Why? Large Batch Reduces Noise and may Get Trapped in Local Minima

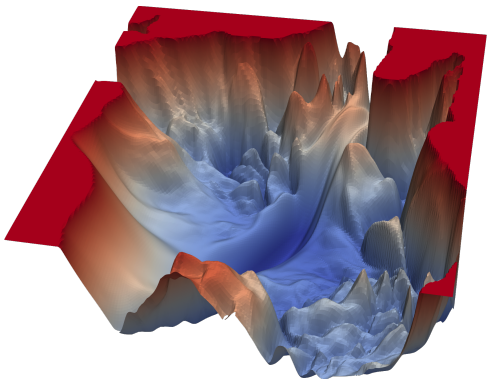
Objective function

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, y_i, \theta)$$

Update rule

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|B|} \sum_{(x,y)\in B} \nabla_\theta l(x, y, \theta_t)$$





Active Research problem: *Addressing the generalization gap for large batch sizes.*

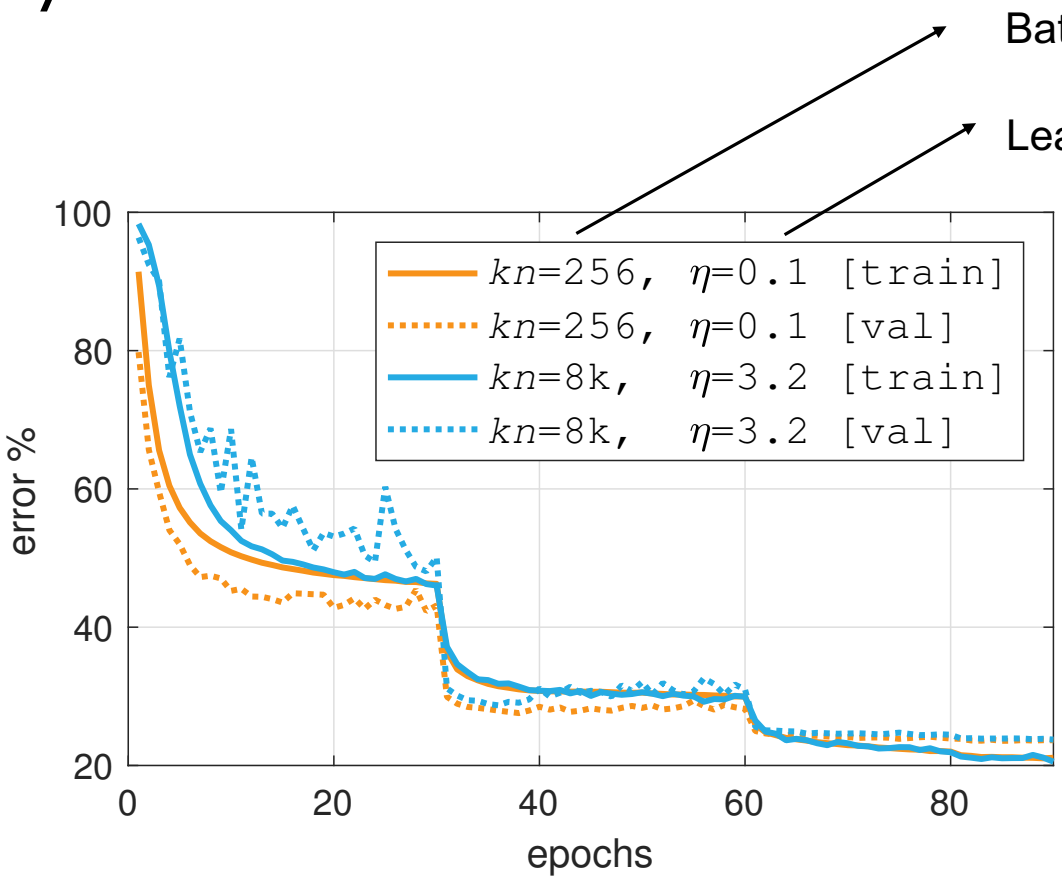
Solution: Linear Scaling Rule

- Scale the learning rate linearly with the batch size

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \overbrace{\hat{\eta}}^{=\eta k}\left(\frac{1}{k}\sum_{j=1}^{k}\frac{1}{|\mathcal{B}_j|}\sum_{i\in\mathcal{B}_j}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))\Big|_{\theta=\theta^{(t)}}\right)$$

- Addresses generalization performance by **taking larger steps** (also improves training convergence)
- **Sub-problem:** *Large learning rates can be destabilizing in the beginning.*
  - **Gradual warmup solution:** increase learning rate scaling from constant to linear in first few epochs
  - Doesn't help for very large k…

Key Results

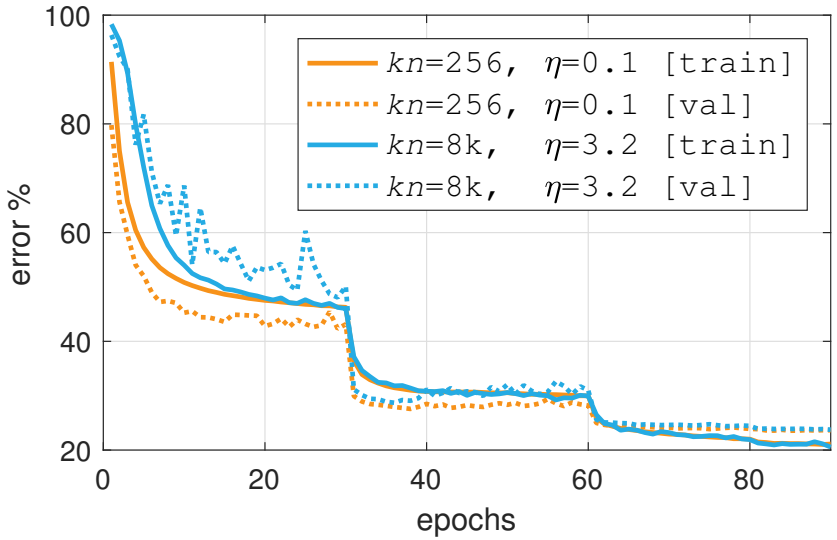


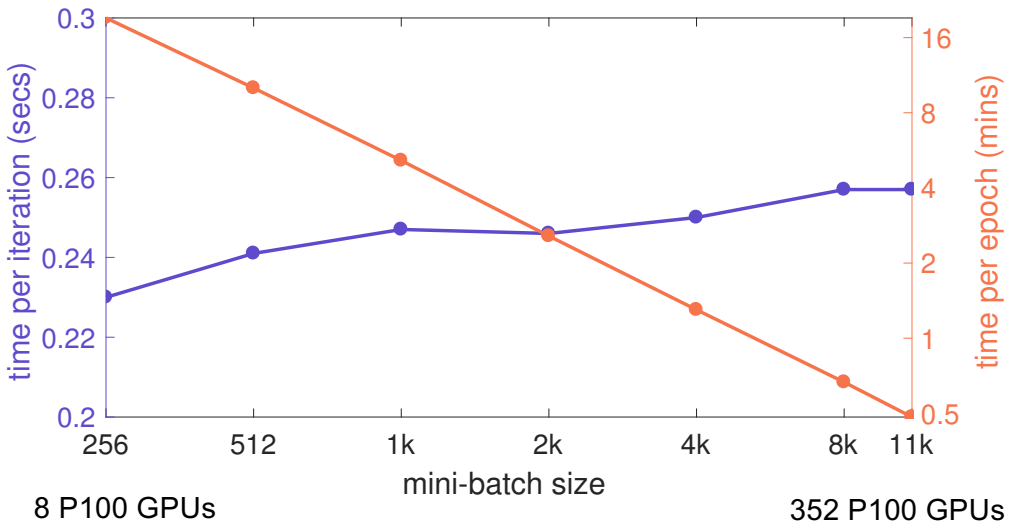
All curves closely match using the linear scaling rule.

Note learning rate schedule drops.

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

Key Results





$$\frac{\text{“Learning”}}{\text{Epoch}}$$

Machine Learning

$$\frac{\text{Epoch}}{\text{Second}}$$

System

Key Results

- Train ResNet-50 to state-of-the-art on 256 GPUs in 1 hour
  - 90% scaling efficiency

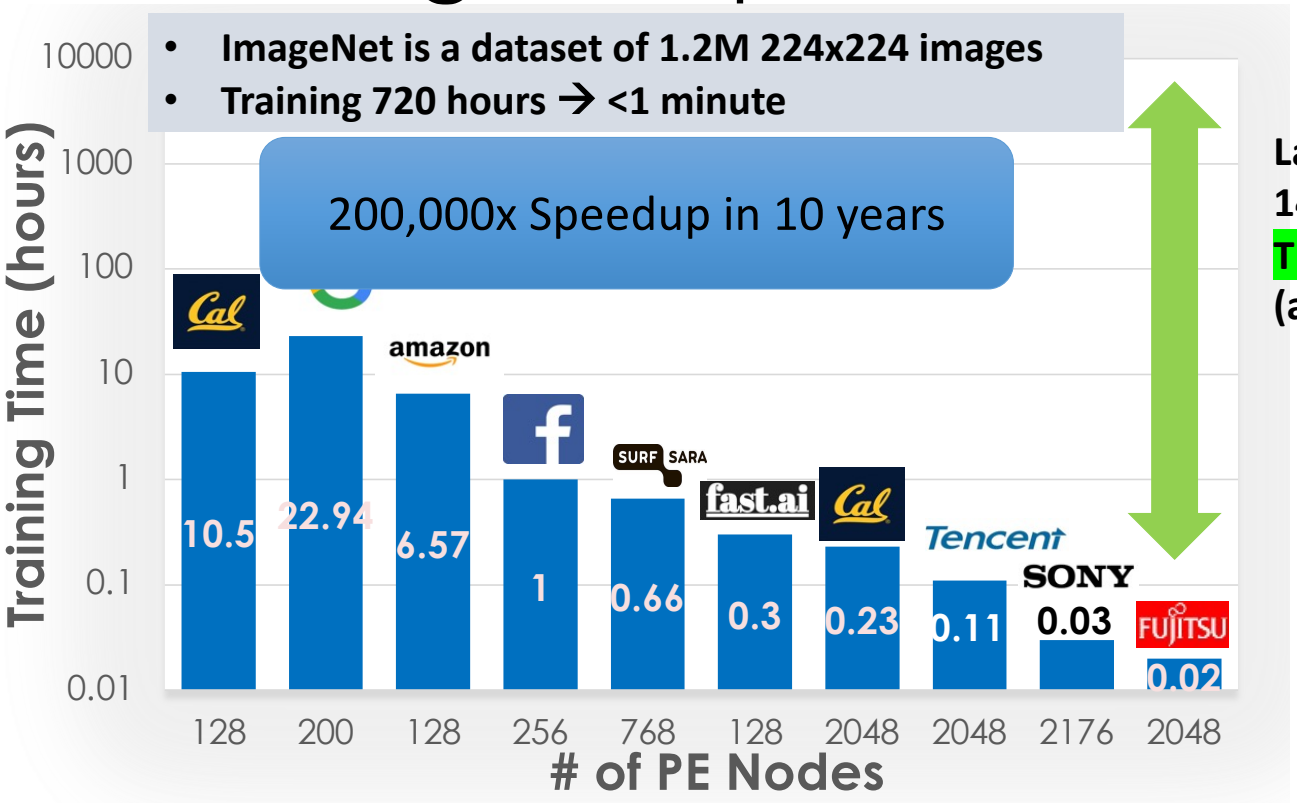
- Fairly careful study of the linear scaling rule
  - Observed limits to linear scaling do not depend on dataset size
  - But what is the limit?
    - You cannot indefinitely scale the learning rate …

Since then there has been a race to train ImageNet faster and several new large batch training methods have been developed (some with good foundation and some heuristics)

ImageNet Training Competition!

Iandola

Yang You



Latest record:
14 seconds on
TPUv4-6912
(as of Feb 2022)

- Iandola FN, Moskewicz MW, Ashraf K, Keutzer K. **FireCaffe**: near-linear acceleration of deep neural network training on compute clusters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2016 (pp. 2592-2600).
- You Y, Zhang Z, Hsieh CJ, Demmel J, Keutzer K. **Imagenet training in minutes**. In Proceedings of the 47th International Conference on Parallel Processing 2018 Aug 13 (p. 1) ACM **(Best Paper Award)**

Very active area of research

Large batch training is very brittle and often requires a lot of tuning to the point that it is often not worth the extra compute. Solving this is still an open research area:

- Golmant, Noah, et al. "On the computational inefficiency of large batch sizes for stochastic gradient descent" **(Cal)**
- Shallue et al. "Measuring the Effects of Data Parallelism on Neural Network Training" **(Google)**
- You, Yang, et al. "Large batch optimization for deep learning: Training BERT in 76 minutes." **(Cal)**

Data Parallelism Summary

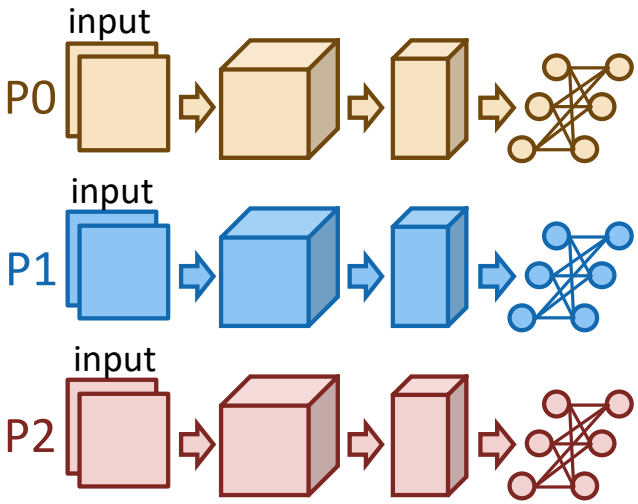
- An efficient parallel training method where the comm time is independent of processors with ring allreduce
- Very easy to implement. Only requires allreduce operation before updating parameters
- Very challenging to scale. Using large batch training is not an option as it hurts generalization performance.
  - 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
- Processes are never idle

Pipeline Parallelism

Really a form of model parallelism

Parallel and distributed training

Data parallelism



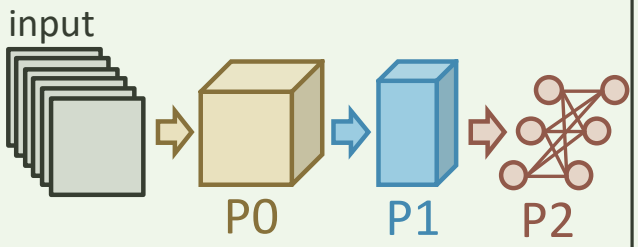
Pros:

a. Easy to realize

Cons:

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

Pipeline parallelism



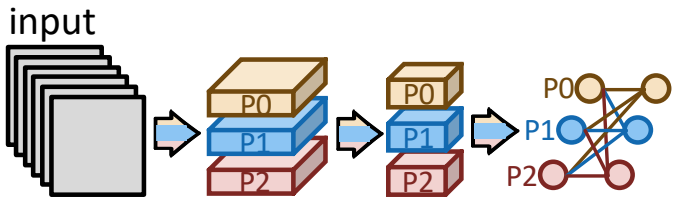
Pros:

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:

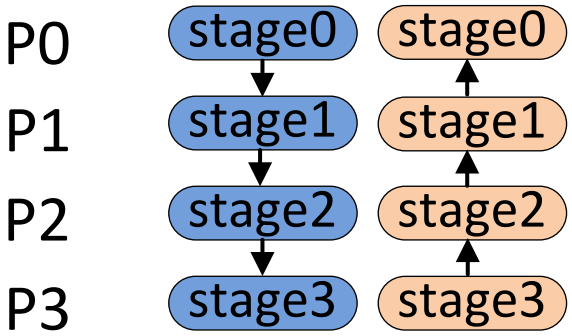
a. Make large model training feasible

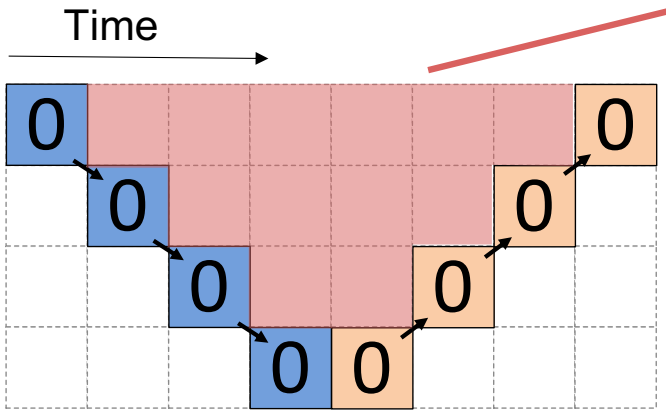
Cons:

b. Communication for each operator (or each layer)

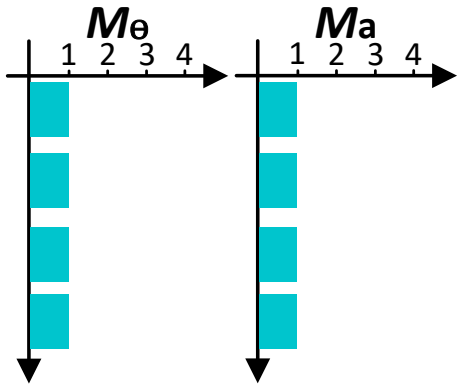
Slide: Courtesy of Shigang Li

Pipeline Parallelism





Bubble where processes are idle



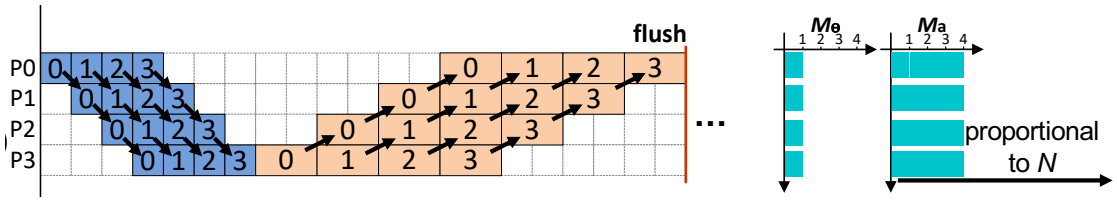
Bubble

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

GPipe [NeurIPS'19]: Reduce Bubble with Micro-Batching



- GPipe reduces the bubble size by breaking the batch size into smaller pieces to reduce the idle time of the processes
- Pro: Reduces bubble size in an easy to implement manner
- Con: Significantly increases activation memory

Bubble

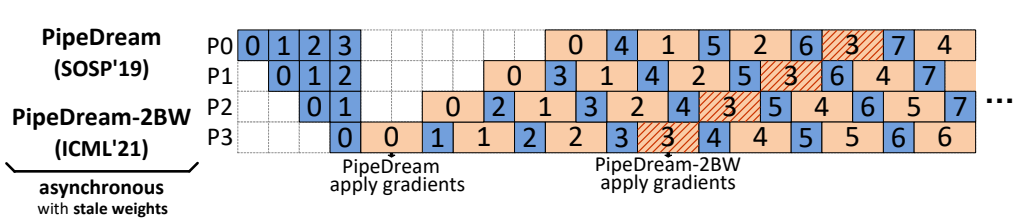
X X Forward and backward passes of ***model replica*0** for micro-batch **x**

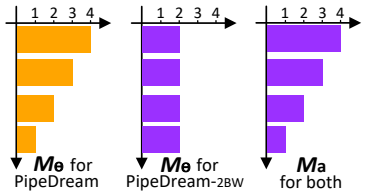
M_θ Memory consumption for the weights

M_a Memory consumption for the activations

Slide: Courtesy of Shigang Li

PipeDream[SOSP'19]: Use Async Updates to remove Bubble





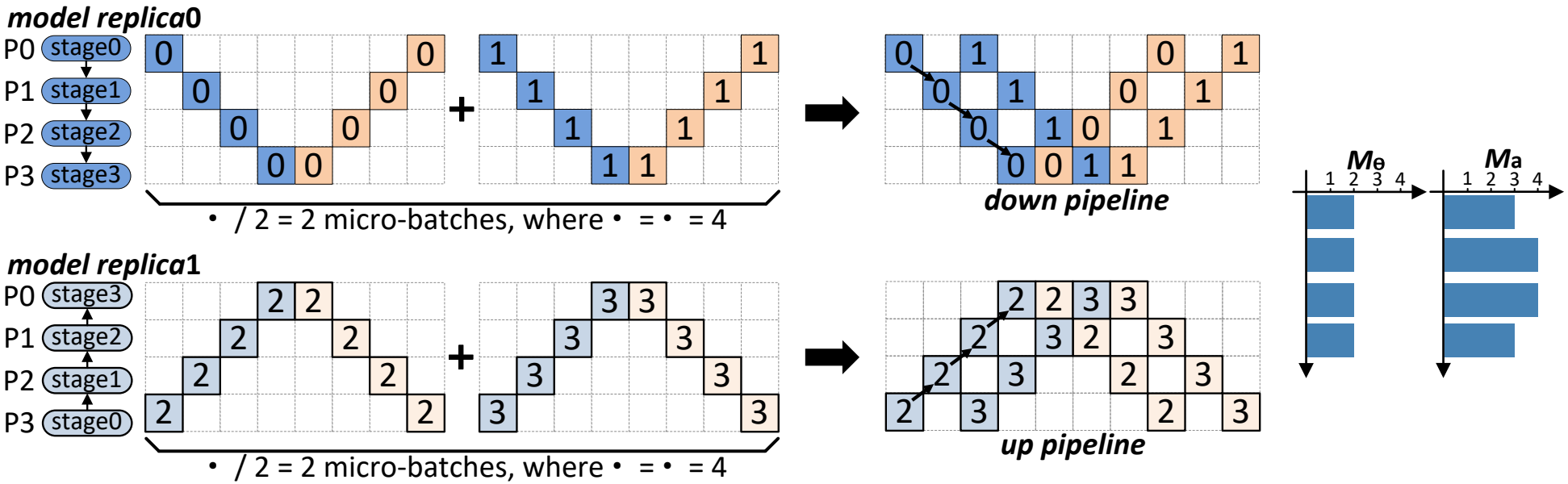
- Pipedream uses asynchronous training: Avoid any idling by always doing a forward/backward pass irrespective of stale gradients/weights
- Pro: No bubble
- Con: As with other async methods this does affect model accuracy and convergence, and as such has not been adopted in industry.

Slide: Courtesy of Shigang Li

Asynchronous Methods

- General advice: Training methods that adversely affect generalization are not adopted, unless there is a 10x speed improvement.
- Otherwise, there are so many moving parts that can go wrong in training NNs, that most often practitioners stay away from async methods unless absolutely necessary
  - For example training very large rec systems.

Chimera: Bidirectional Pipeline



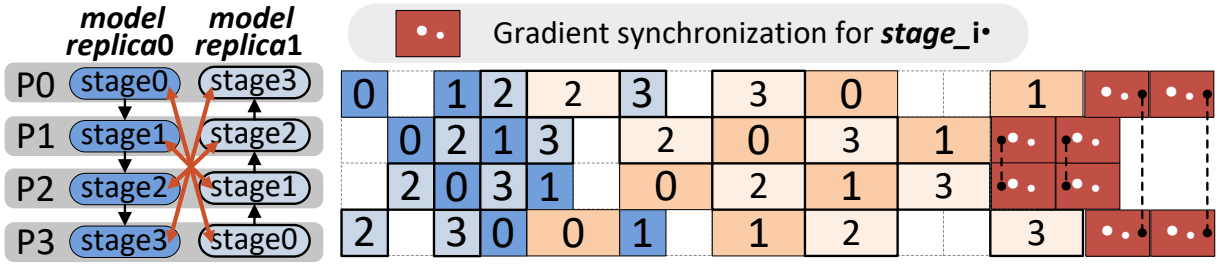
- Big idea: Replicate the model to the other processes so that we can do forward pass in two directions

x x Forward and backward passes of ***replica0***

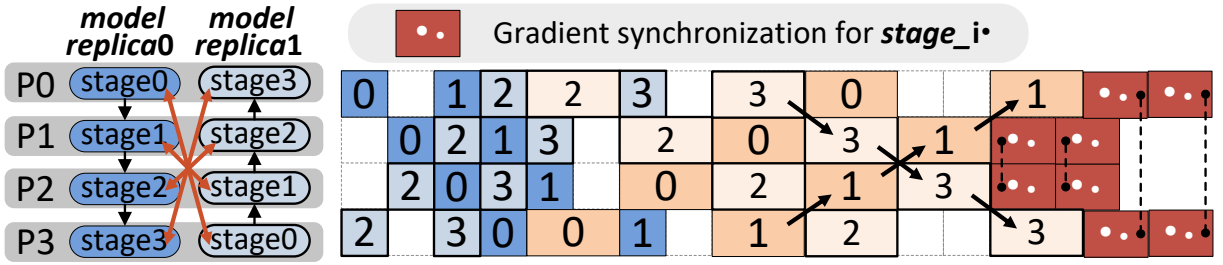
y y Forward and backward passes of ***replica1***

Slide: Courtesy of Shigang Li

Gradient Synchronization



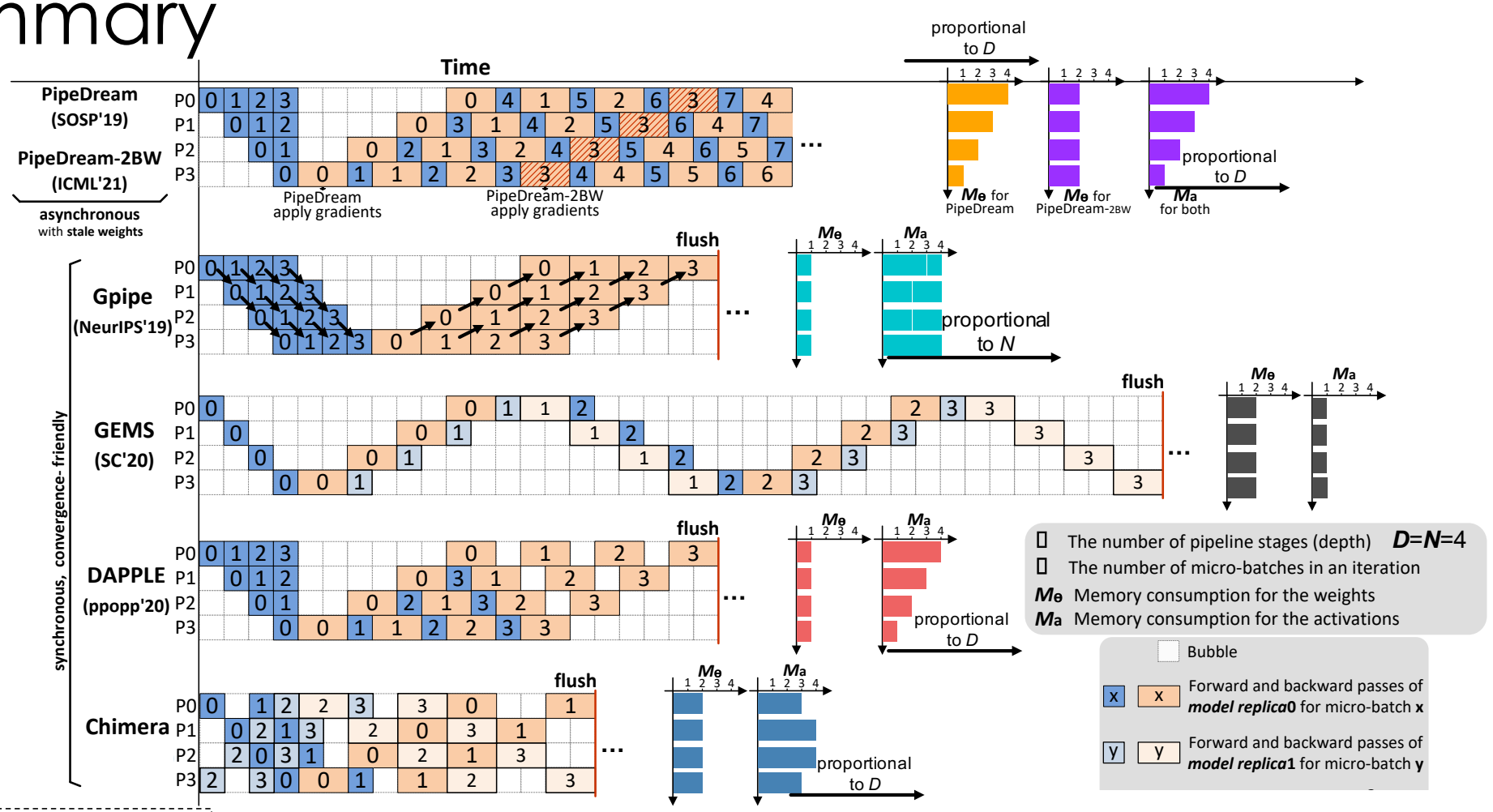
(a) Gradient synchronization after all local computation is finished



(b) Eager gradient synchronization for deeper overlapping

Slide: Courtesy of Shigang Li

Summary



Slide: Courtesy of Shigang Li

Pipeline Parallelism Summary

- Slightly more involved algorithm than data parallel method but with the advantage of only requiring point to point communication
- Ideal 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
- Requires special handling of bubble that results in idle processes

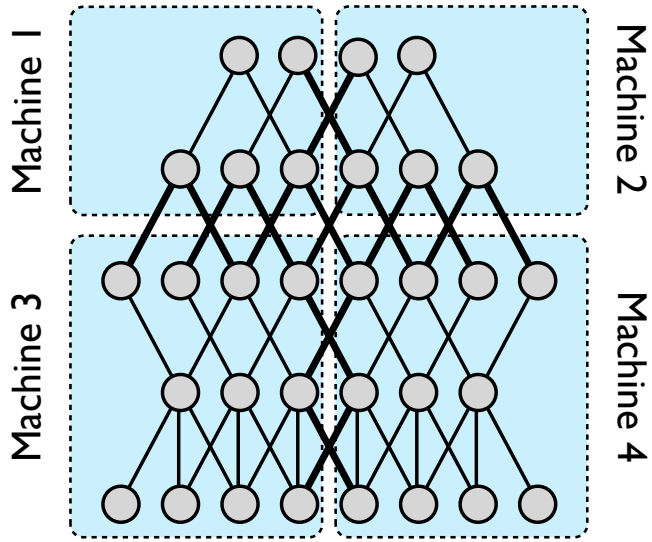
Model Parallelism

AKA Operator Parallelism

Model Parallelism

Divide the model across machines and replicate the data.

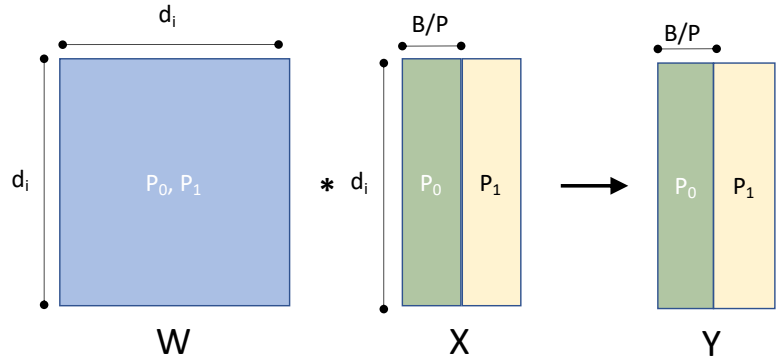
- Supports large models and activations
- Requires communication within single evaluation
- How to best divide a model?
  - Split individual layers
    - which dimension?
      - Weights or spatial → depends on operation
  - Split across layers
    - Only one set of layers active a time → poor work balance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



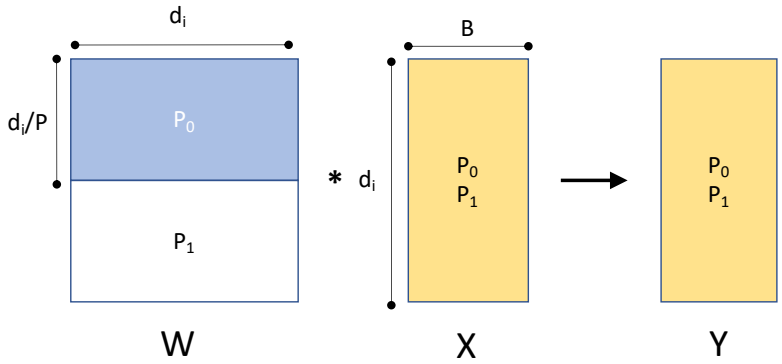
Model Parallelism: Weights

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

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

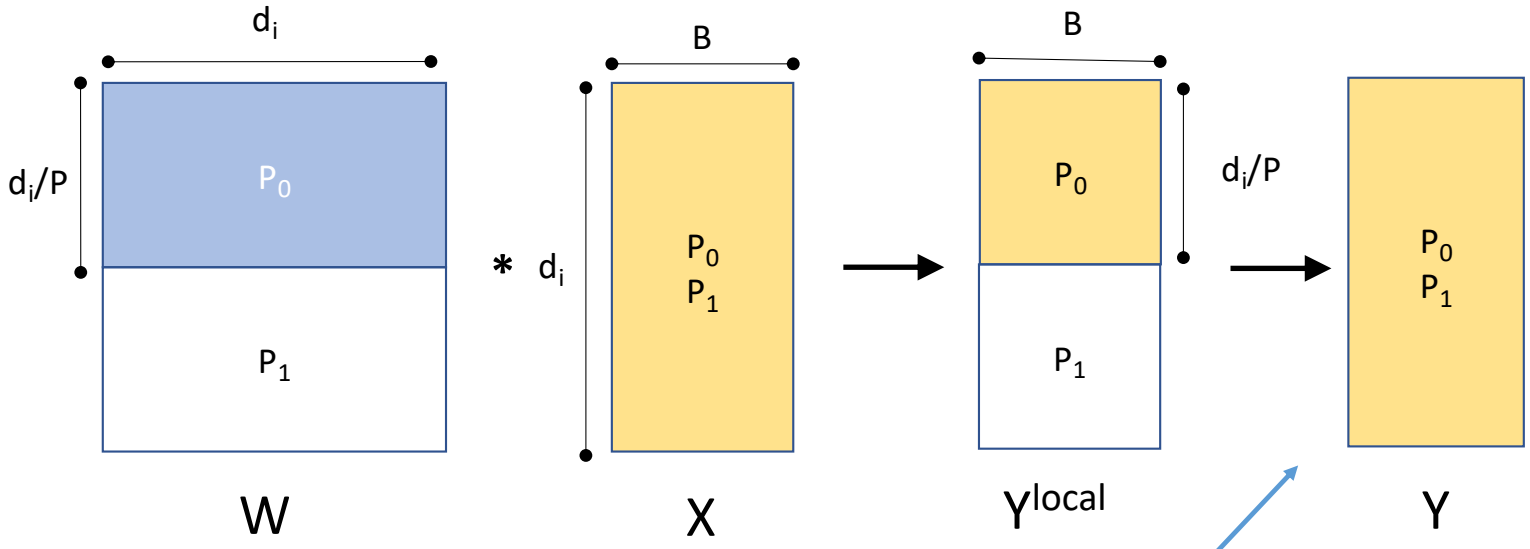


Model Parallelism: Partition weights across different Processes (W dimension)



Let's discuss the communication details, step by step

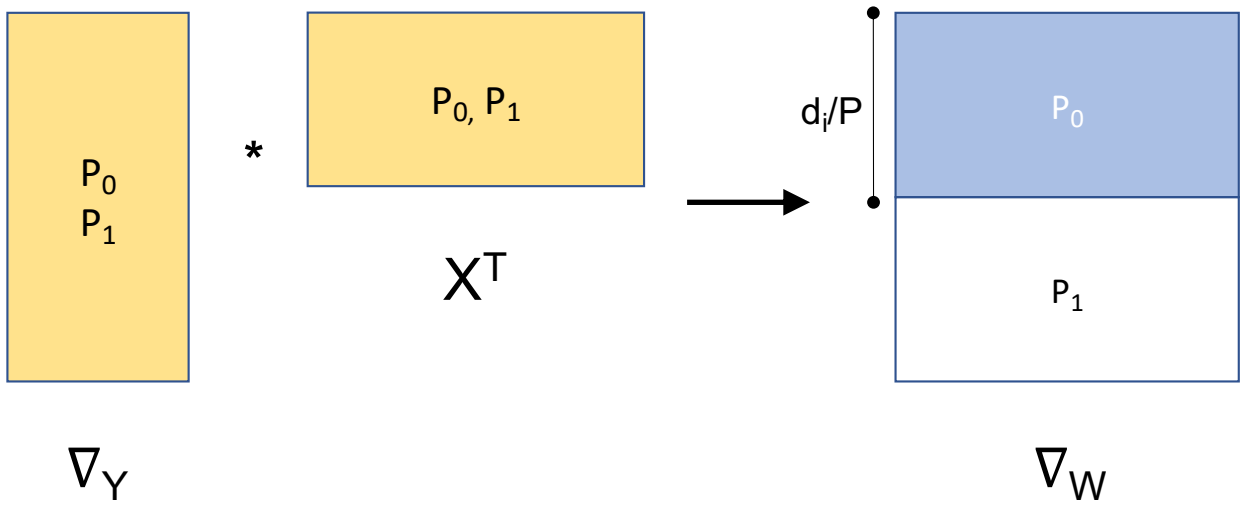
Model Parallelism: Forward Pass



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

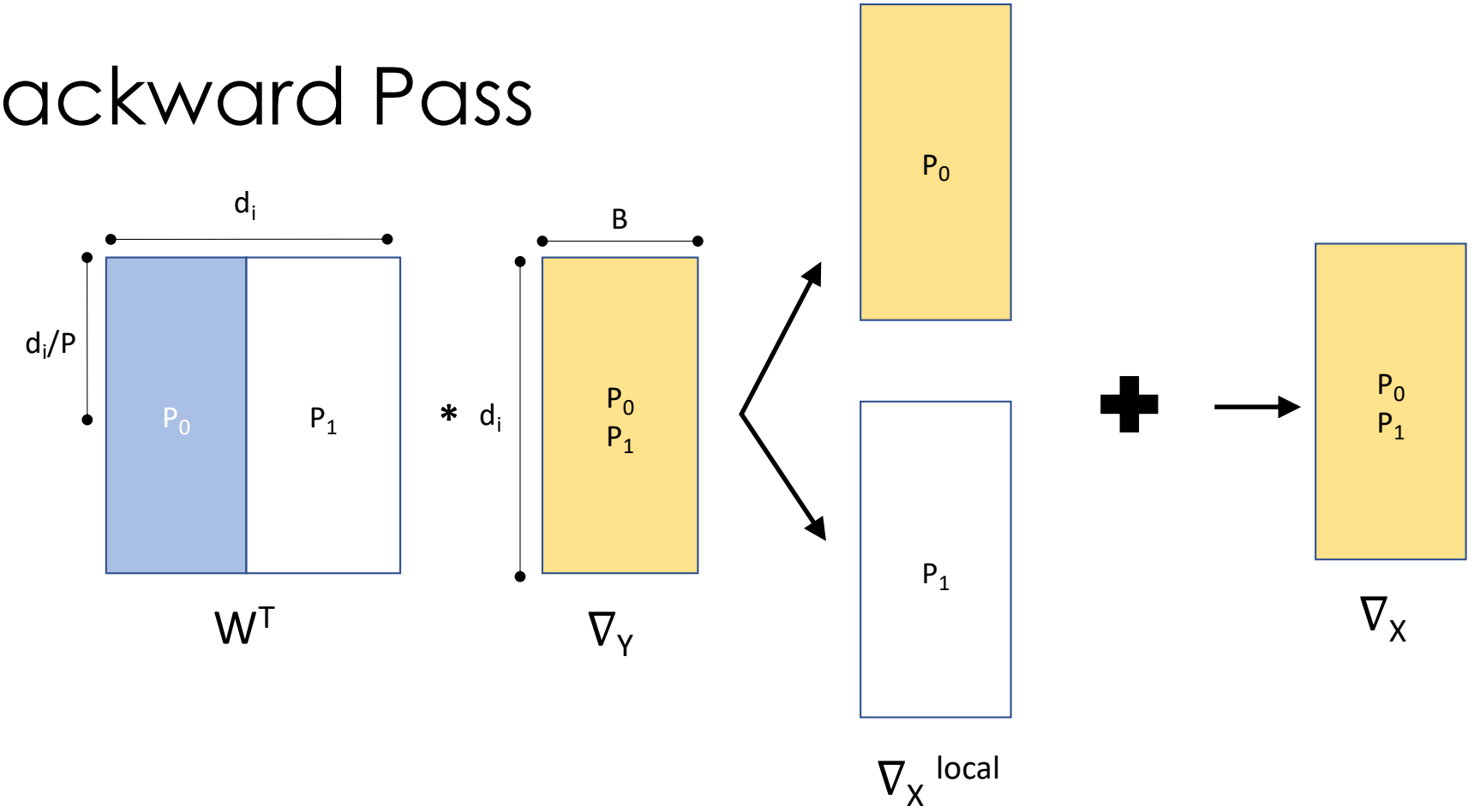
$$\sum_{i=1}^{L} \left(\beta(P-1)\frac{Bd_i}{P} \right)$$

Model Parallelism: Backward Pass



No communication needed as every processor only needs the gradient of its own parameters

Backward Pass



- Aggregating input gradient requires an allreduce operation

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

Communication 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

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

All Gather 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 comm on 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.

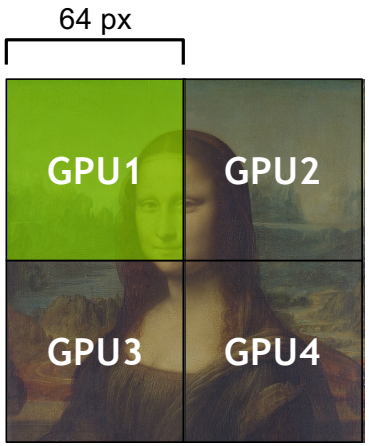
Model Parallelism Summary

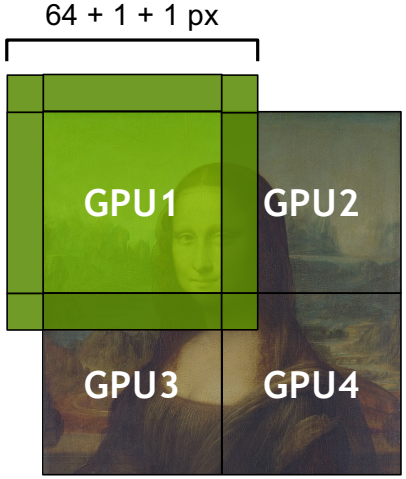
- More optimal comm time for large FC layers than Data parallel approach
- Makes training large models feasible by breaking it into smaller parts
- However, requires blocking collective communication during **both** forward pass (all gather), as well as backwards pass (all reduce)
- Slightly harder to implement than data parallel
- Processes are never idle

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

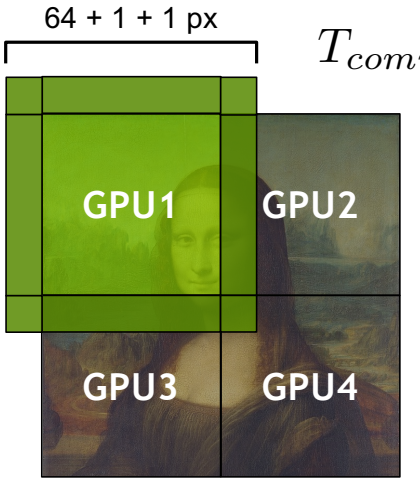




$$T_{comm}(domain) = \sum_{i=1}^{L} \left(\alpha + \beta B X_W^i X_C^i k_h^i / 2\right) + \sum_{i=1}^{L} \left(\alpha + \beta B Y_W^i Y_C^i k_w^i / 2\right) + 2 \sum_{i=1}^{L} \left(\alpha \log(P) + \beta \frac{P-1}{P} |W_i|\right)$$

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

Communication Complexity



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

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

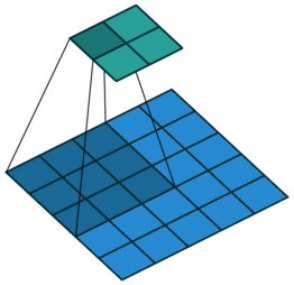
$$+2 \sum_{i=0}^{L} \left(\alpha \log(P) + \beta \frac{P-1}{P} |W_i|\right)$$ All reduce Cost (same as before)

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