CS 181AI
Lecture 21

Distributed Training Pt. 1

Arthi Padmanabhan

Apr 5 2023

Logistics

- Assignment 5 (last assignment!) due on Friday
- Outline of rest of semester (!!)
  - Today (4/5): Distributed Training part 1
  - 4/10: Distributed Training part 2
  - 4/12: Profiling + debugging
  - 4/17: Efforts to lower model resource usage (+ different types of models)
  - 4/19: Working session
  - 4/24: Scale of models/resources in industry + look at full stack of ML pipelines
  - 4/26: Project presentations

Today

- Why and how do we use multiple GPUs for training?
- What are the methods for aggregating gradients across GPUs?

Distributed Training

- Distributed training is the use of multiple GPUs working together on one training job

- Why would we need more than one GPU?

Distributed Training

- Distributed training is the use of multiple GPUs working together on one training job
- It can lower the time needed for training
- It can allow you to train when a model is too large for a single GPU

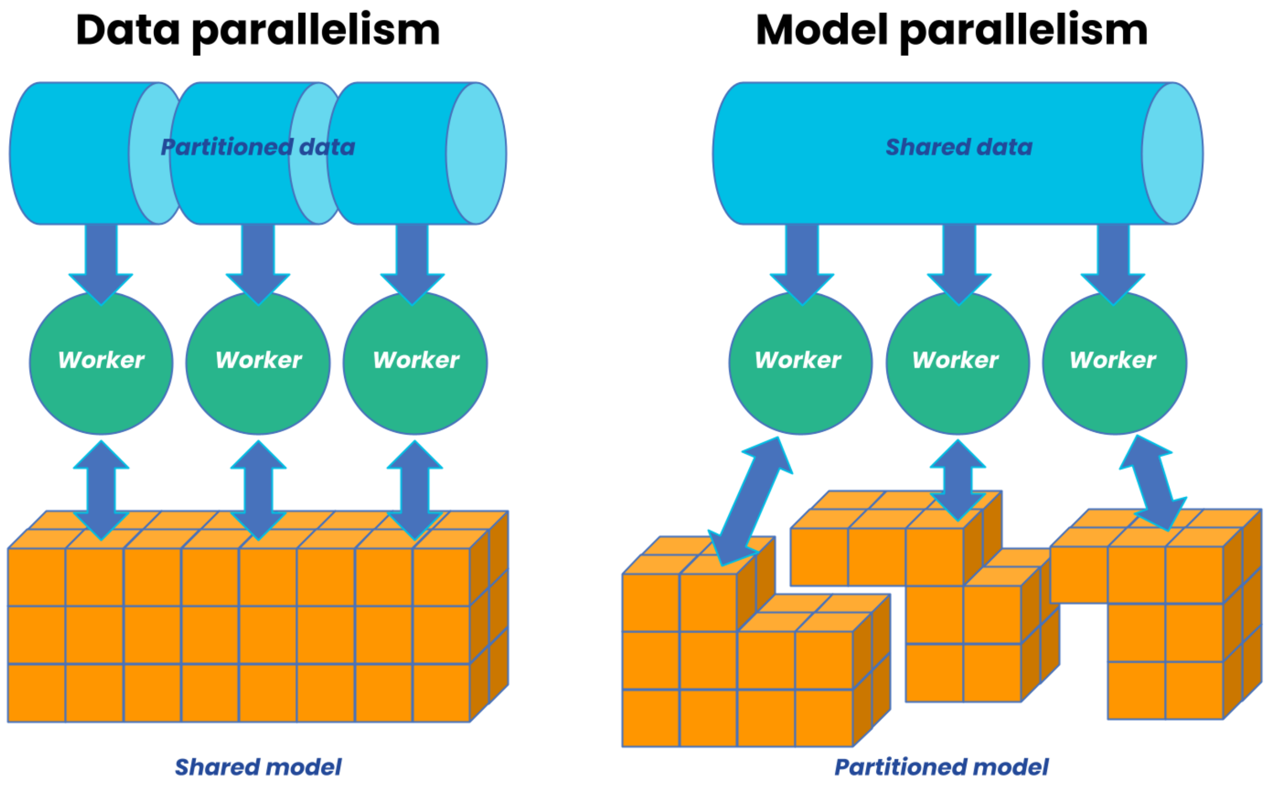
Distributed Training

- Distributed training is the use of multiple GPUs working together on one training job

- It can lower the time needed for training
  - Running a single batch across several GPUs allows for increased batch size and lower training time -> **Data Parallelism**

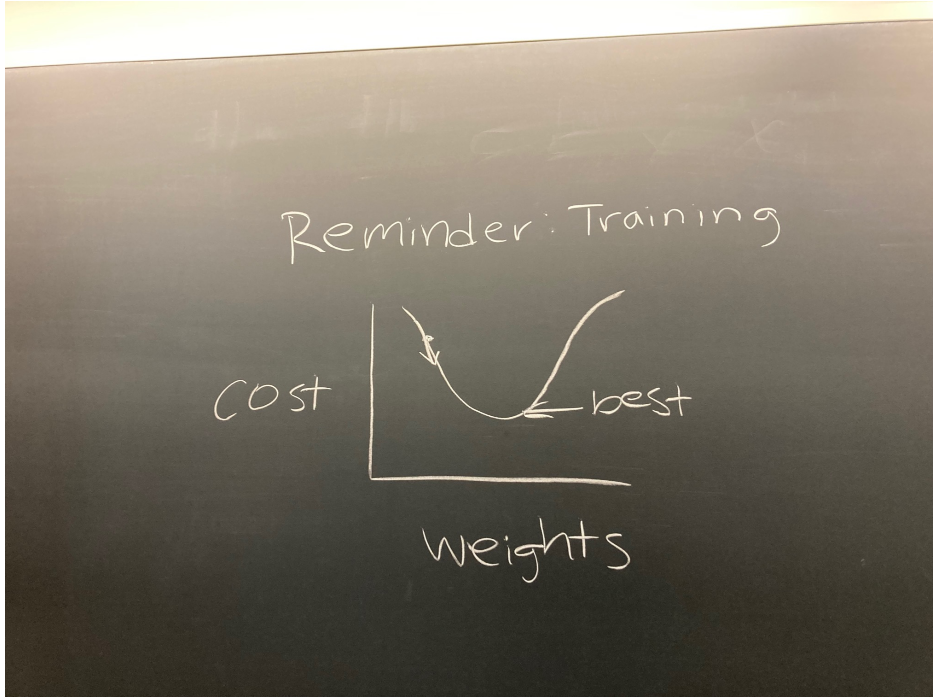
- It can allow you to train when a model is too large for a single GPU
  - Splitting up a model such that different parts are on different GPUs -> **Model Parallelism**

Data vs. Model Parallelism



Reminder: Training Process

- Run inputs through model with weights *w*
- Calculate loss function
- Calculate gradients of loss function
- Update *w*



Reminder: Training Process

- Run inputs through model with weights *w*
- Calculate loss function
- Calculate gradients of loss function
- Update *w*

- Inputs = 1 batch

```
train_gen = torch.utils.data.DataLoader(dataset = train_data,
                                        batch_size = batch_size,
                                        shuffle = True)
```

Training Batch Size

- Higher batch size = faster training
- What is limiting factor for batch size?

Training Batch Size

- Higher batch size = faster training

- GPU memory is limiting factor for batch size

- Using multiple GPUs allows us to effectively use a higher batch size
  - batch_size = batch_size per GPU * num_GPUs

Data Parallelism quiz

- Which of the following is true about data parallelism across 4 GPUs?

1. The time per batch remains about the same. The number of batches per epoch decreases by a factor of 4. The time per epoch remains about the same. The overall training time decreases by a factor of 4.
2. The time per batch remains about the same. The number of batches per epoch decreases by a factor of 4. The time per epoch decreases by a factor of 4. The overall training time decreases by a factor of 4.
3. The time per batch decreases by a factor of 4. The number of batches per epoch stays the same. The time per epoch decreases by a factor of 4. The overall training time decreases by a factor of 4.

Data Parallelism quiz

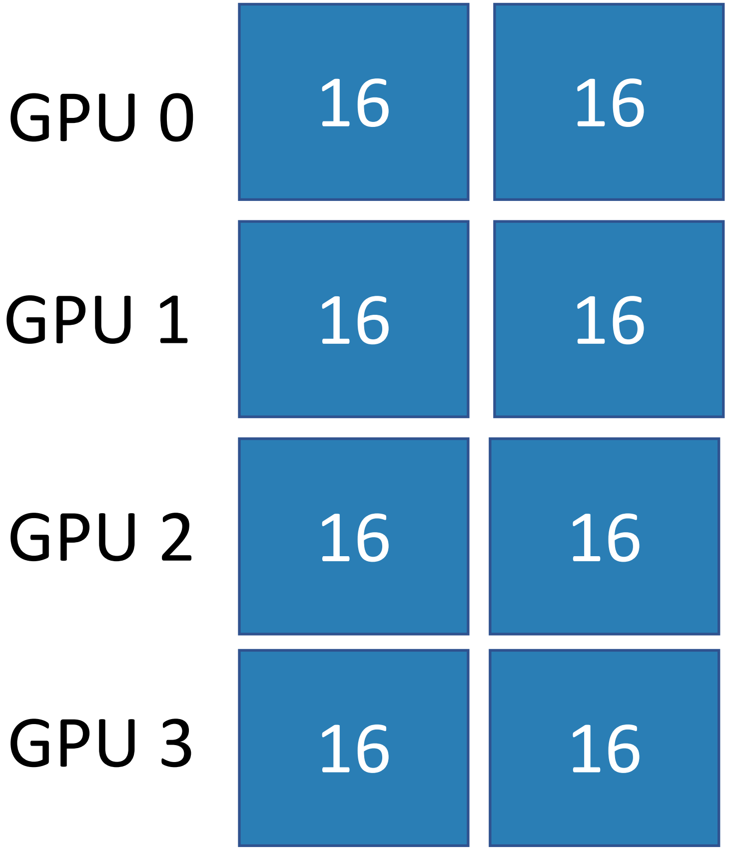
- Suppose batch size = 16, 1 GPU

All Data (128 samples)

Data Parallelism quiz

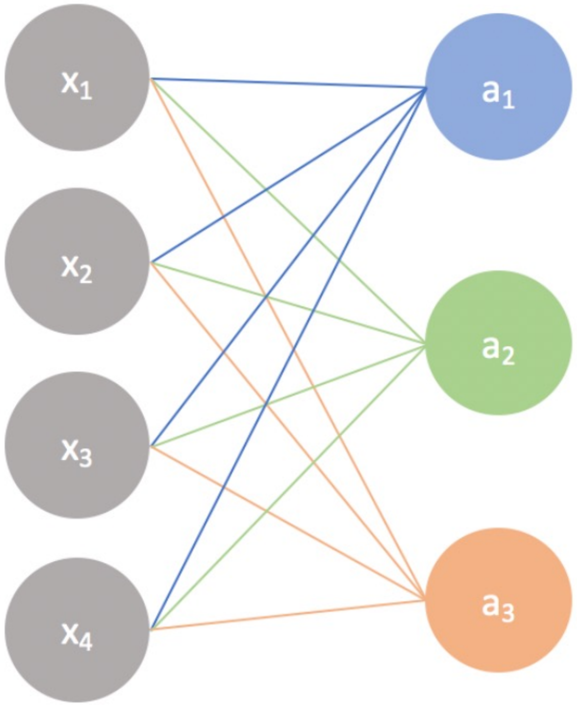
- Suppose batch size = 16, 1 GPU
- 8 batches of 16 = 128 samples

Data Parallelism quiz



- Suppose batch size = 16, 4 GPUs
- 2 batches of 16*4 = 128 samples
- Time per batch is similar*
- Number of batches per epoch decreased
- Time per epoch decreased
- Overall training time decreased

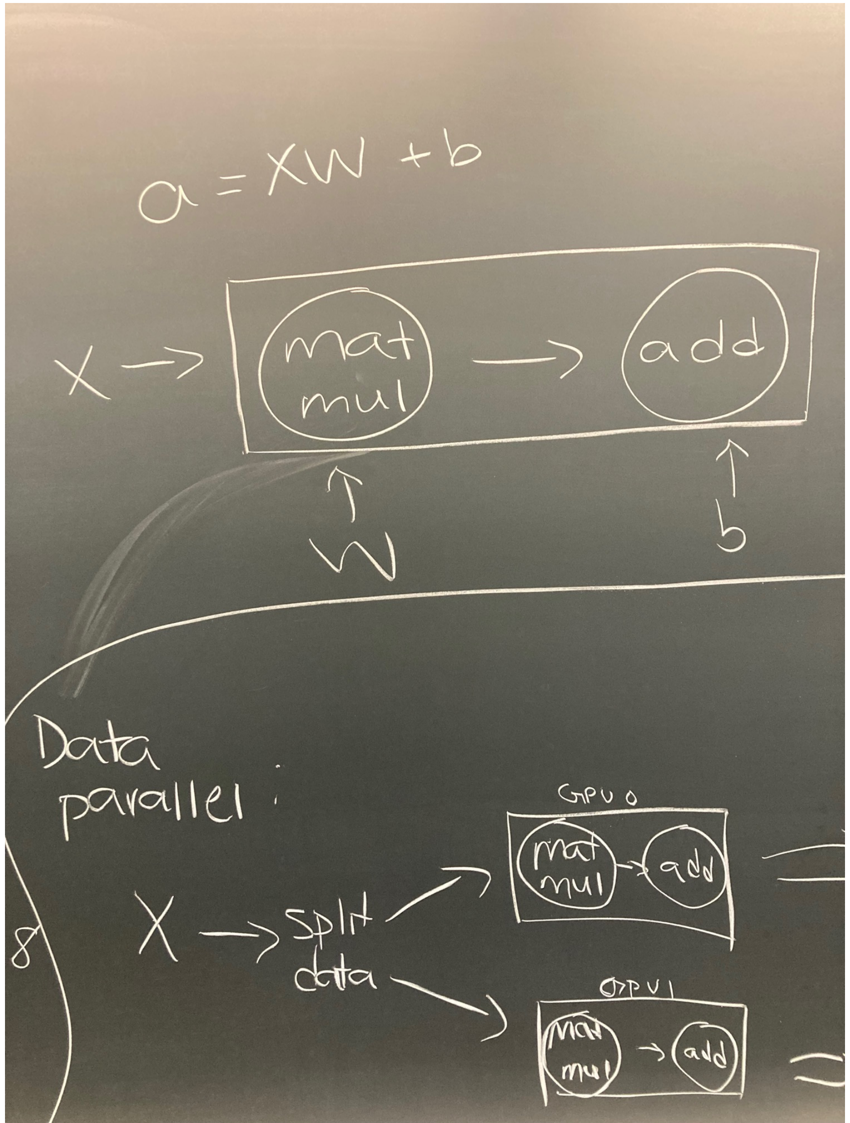
Reminder: Matrix Multiplication + Bias

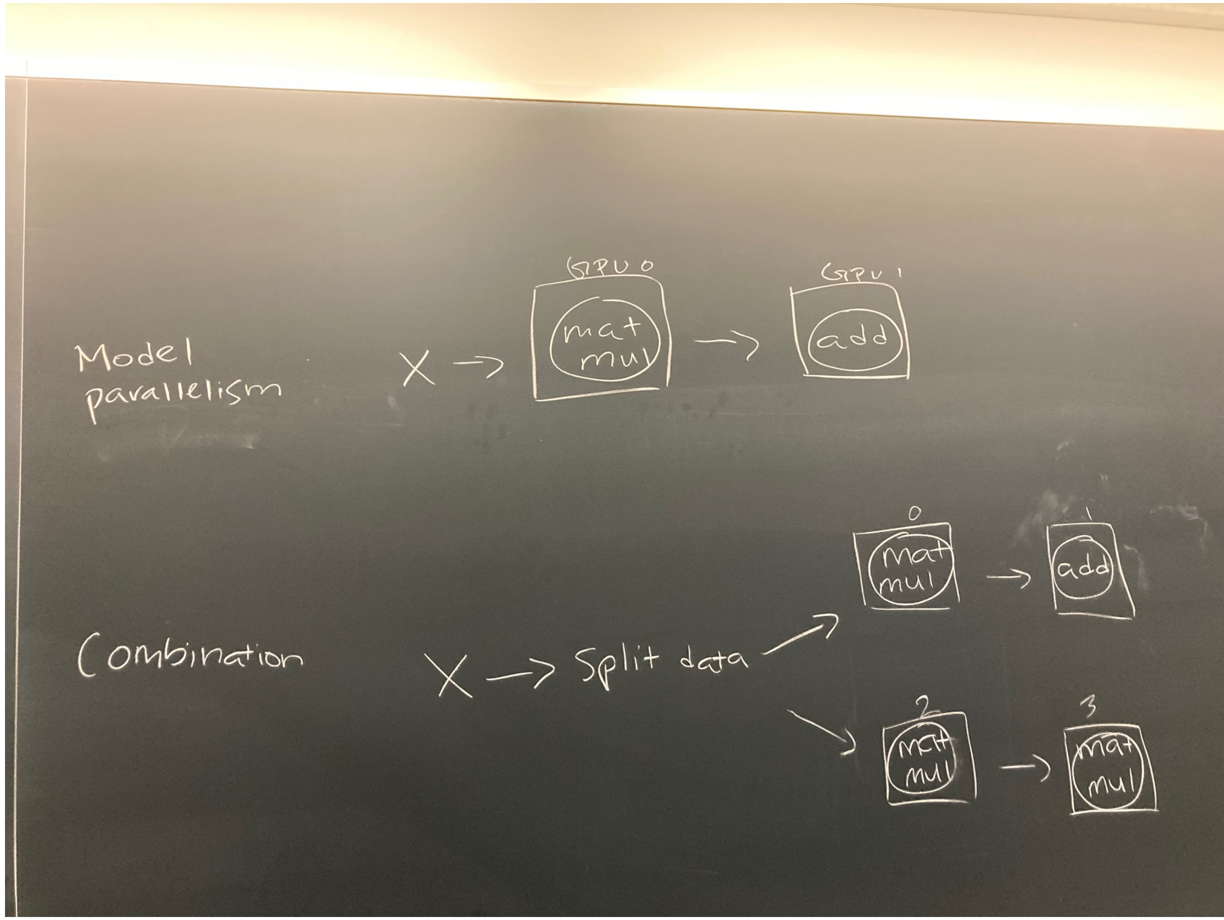


$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[\text{activation}]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

X*W + b = a

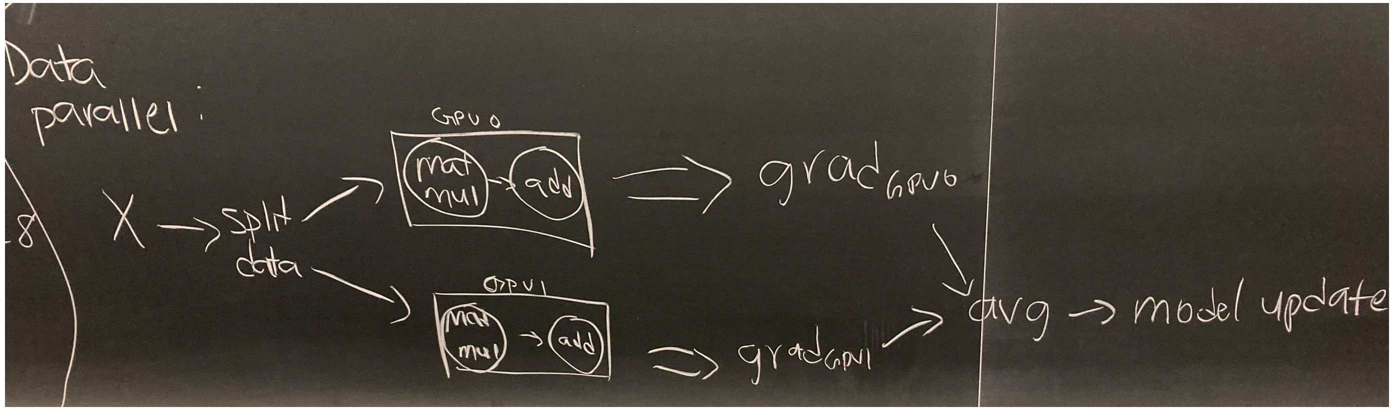
Data Parallelism





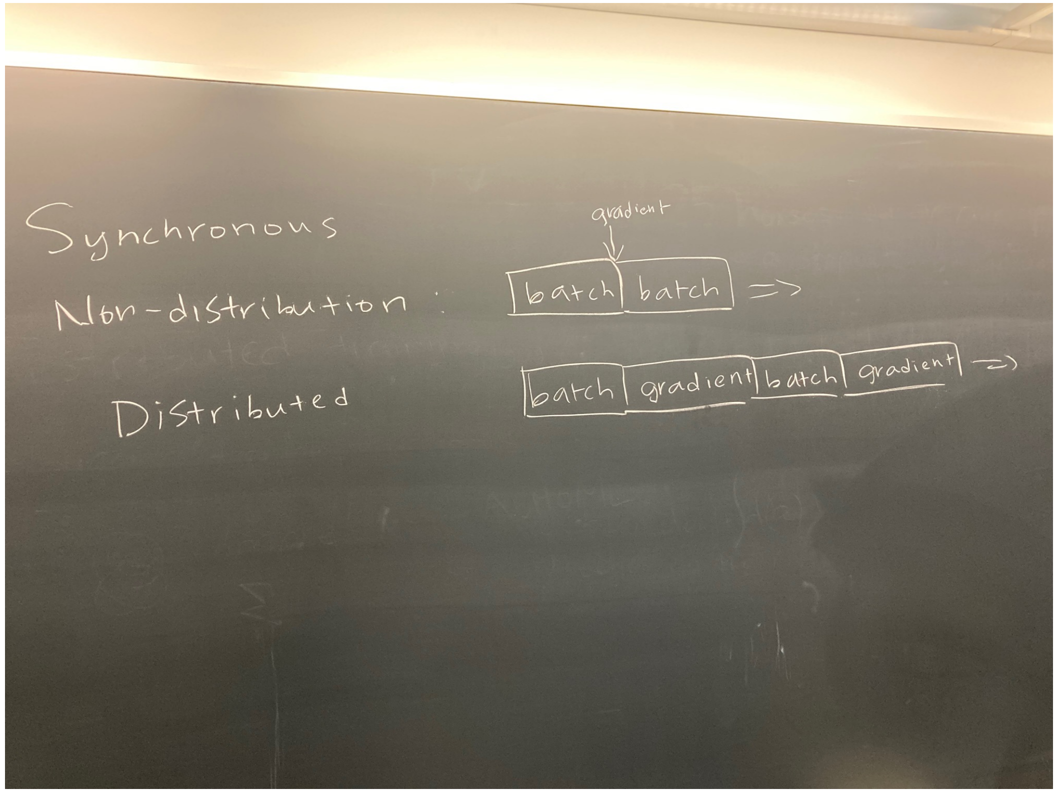
Gradient Update

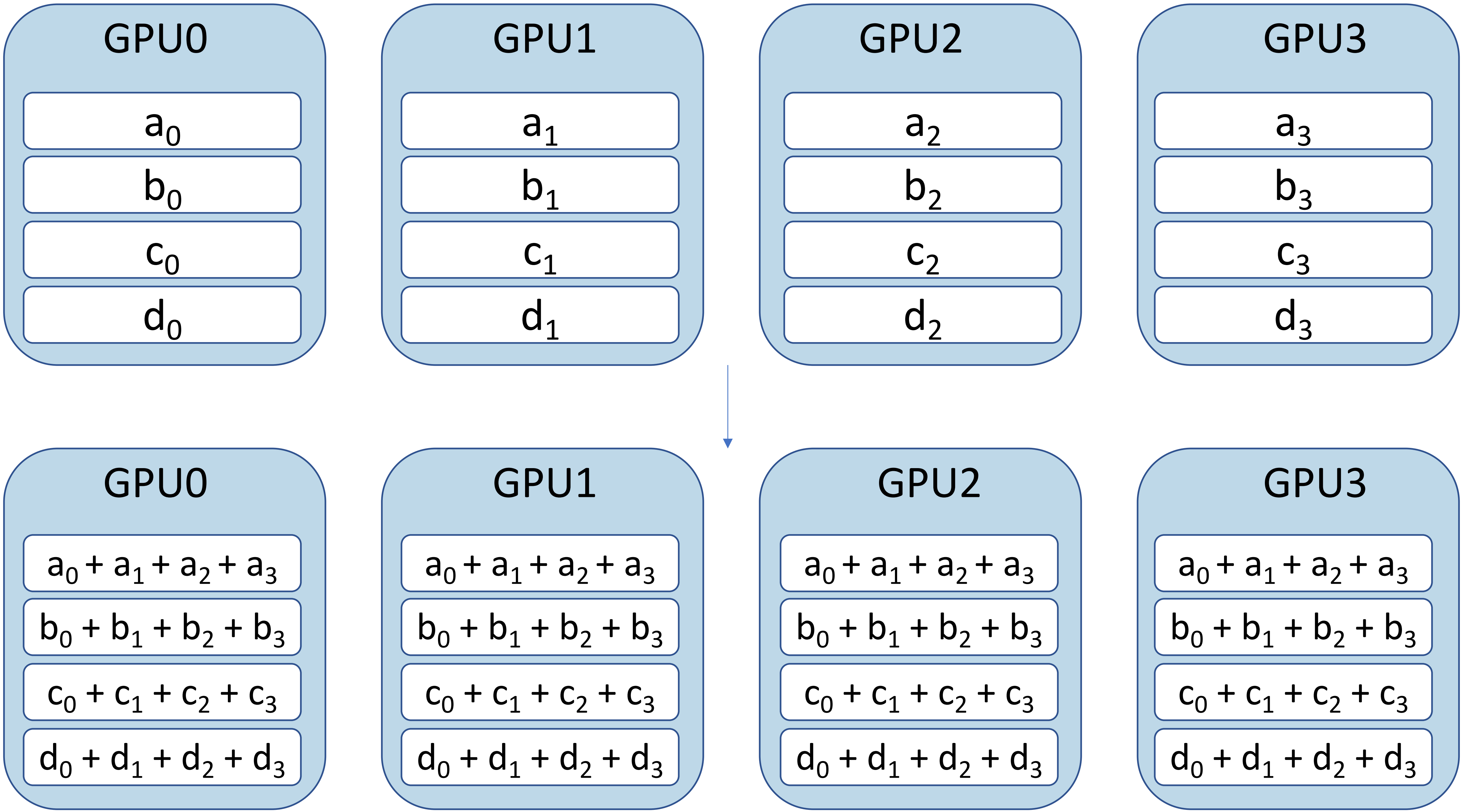
- Reminder: at the end of every batch, gradients for the whole model are updated
- How does this work if each GPU sees part of the batch?
  - Each GPU computes its own loss and gradients. Gradients are averaged and result is used to update the model



Synchronous

- Gradients are computed and model is updated after each batch
- Calculation must happen very efficiently



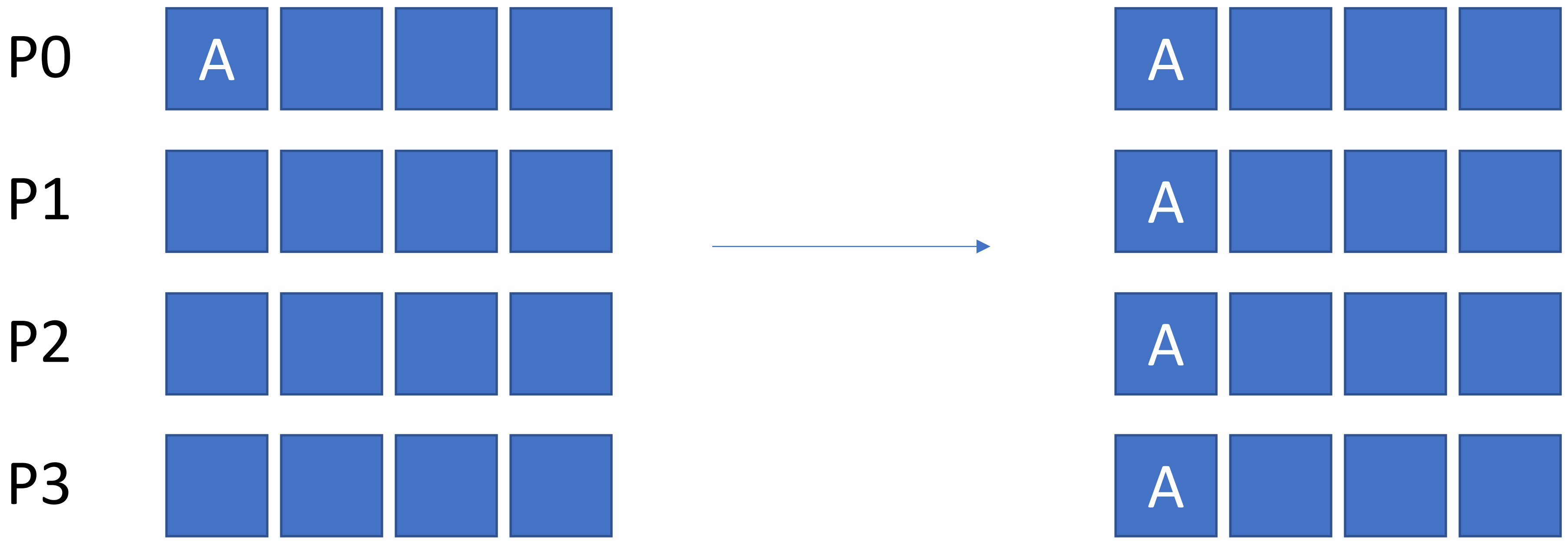


GPU0
a_0
b_0
c_0
d_0
GPU1
a_1
b_1
c_1
d_1
GPU2
a_2
b_2
c_2
d_2
GPU3
a_3
b_3
c_3
d_3
GPU0
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU1
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU2
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU3
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$

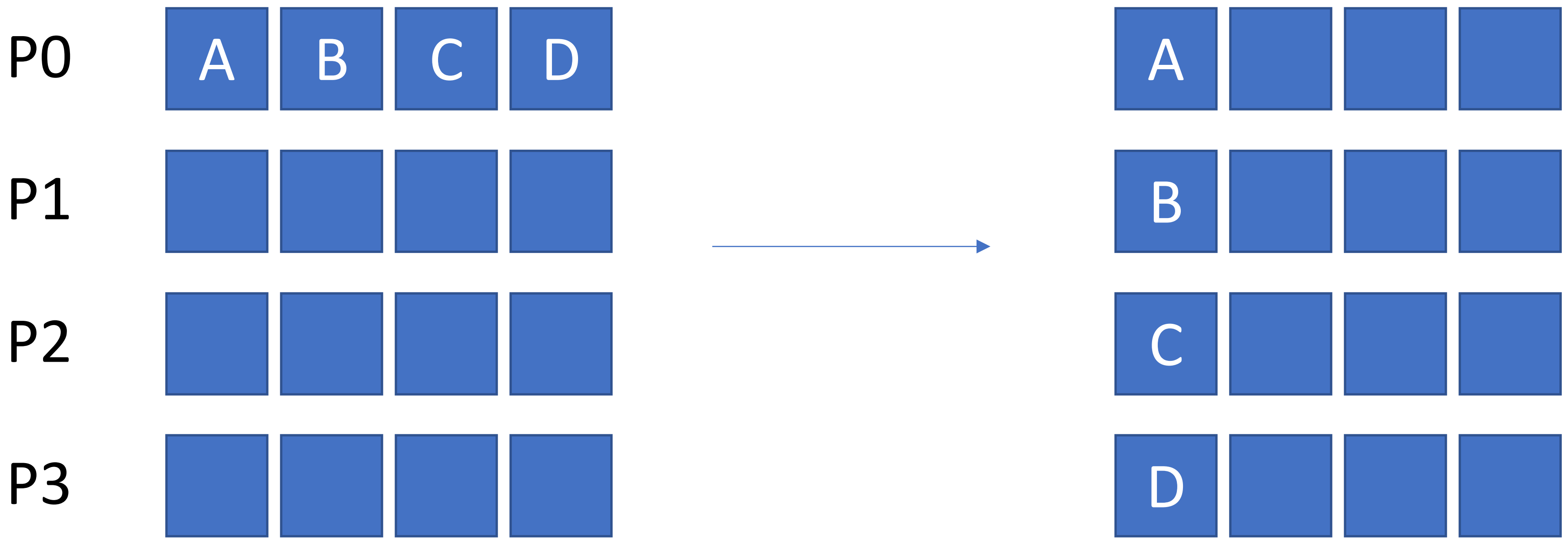
Message Passing Interfaces (MPI)

- We are going to explore how to do that, but first, we'll look at the common message passing operations in parallel computing

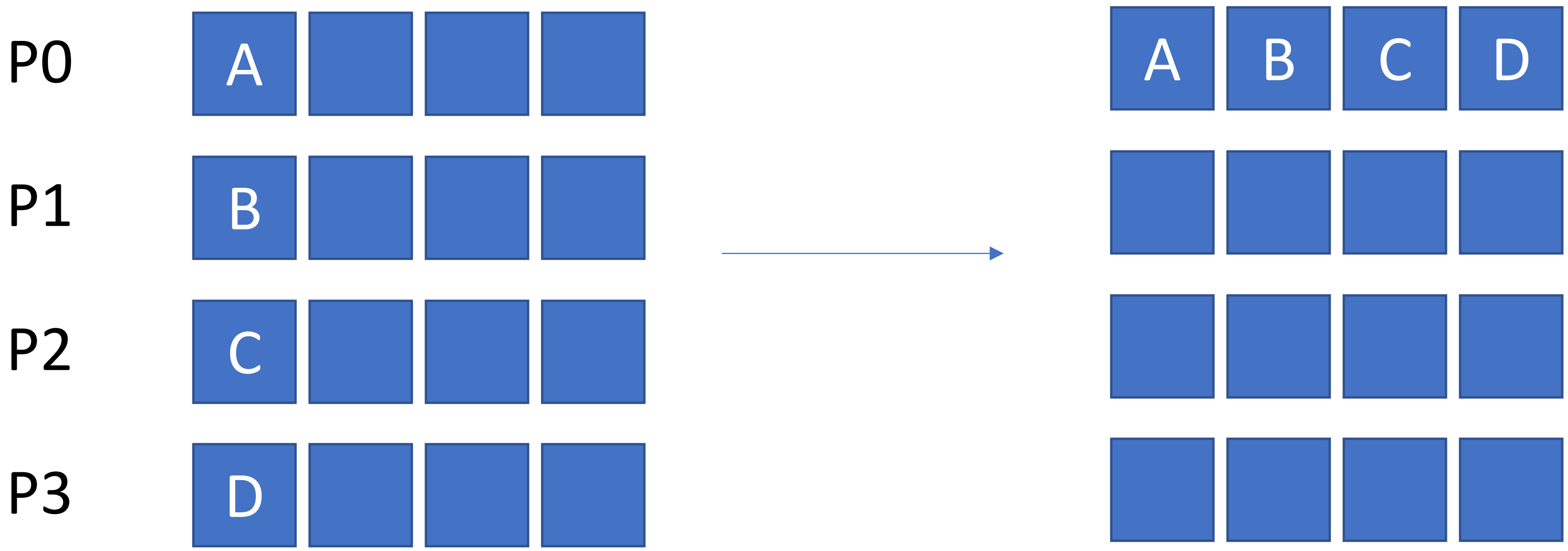
Broadcast



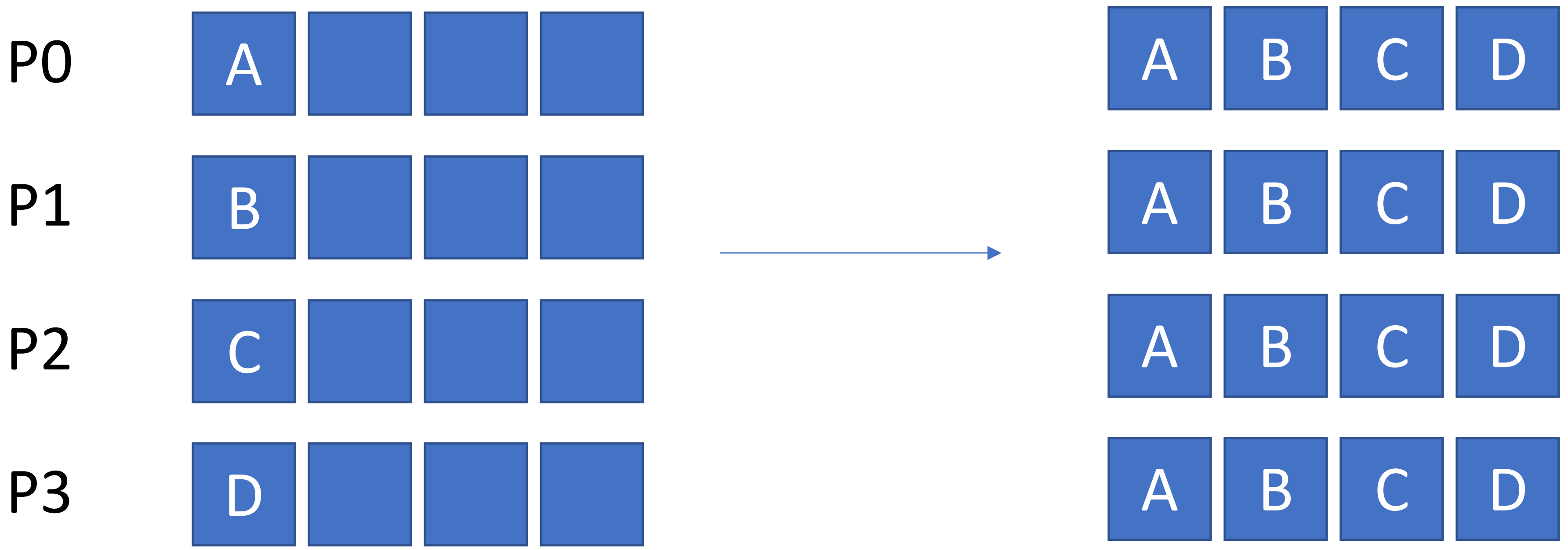
Scatter



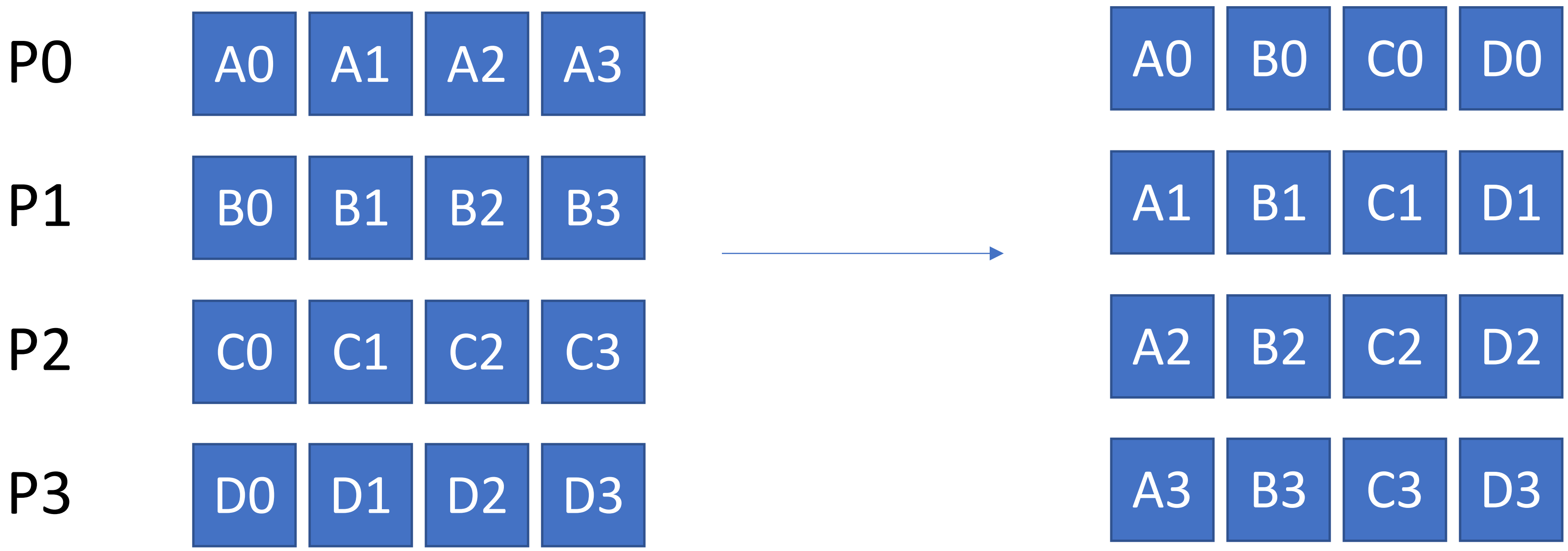
Gather



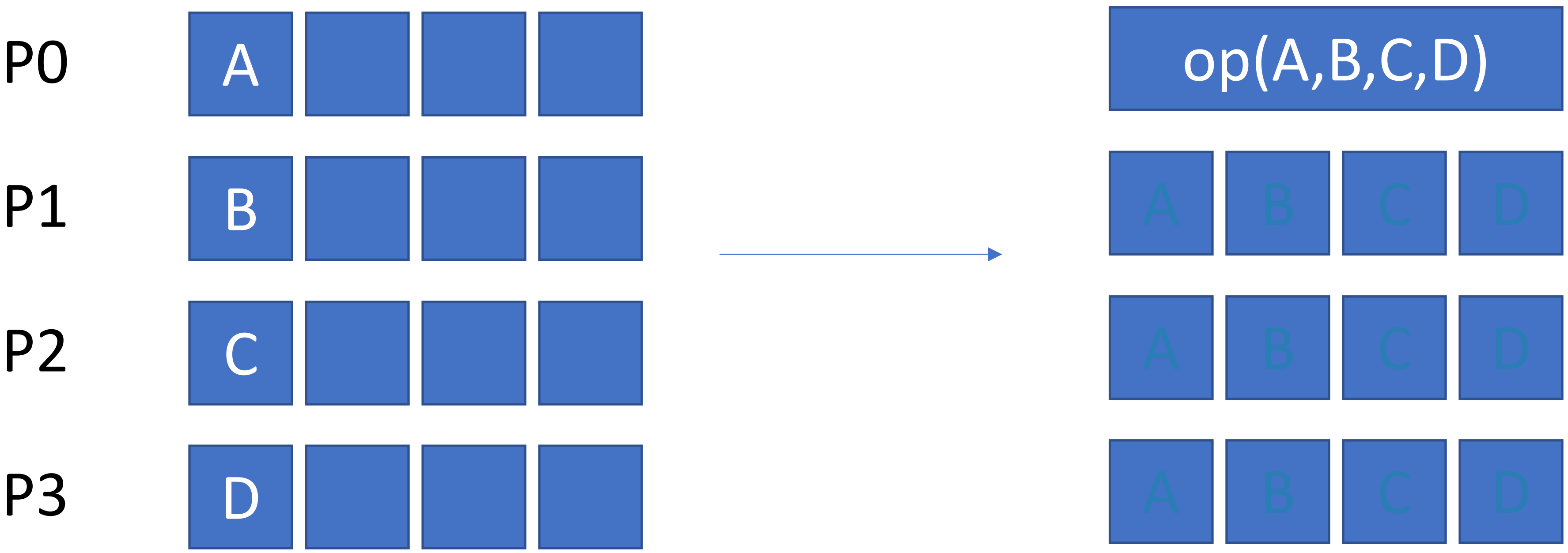
All-Gather



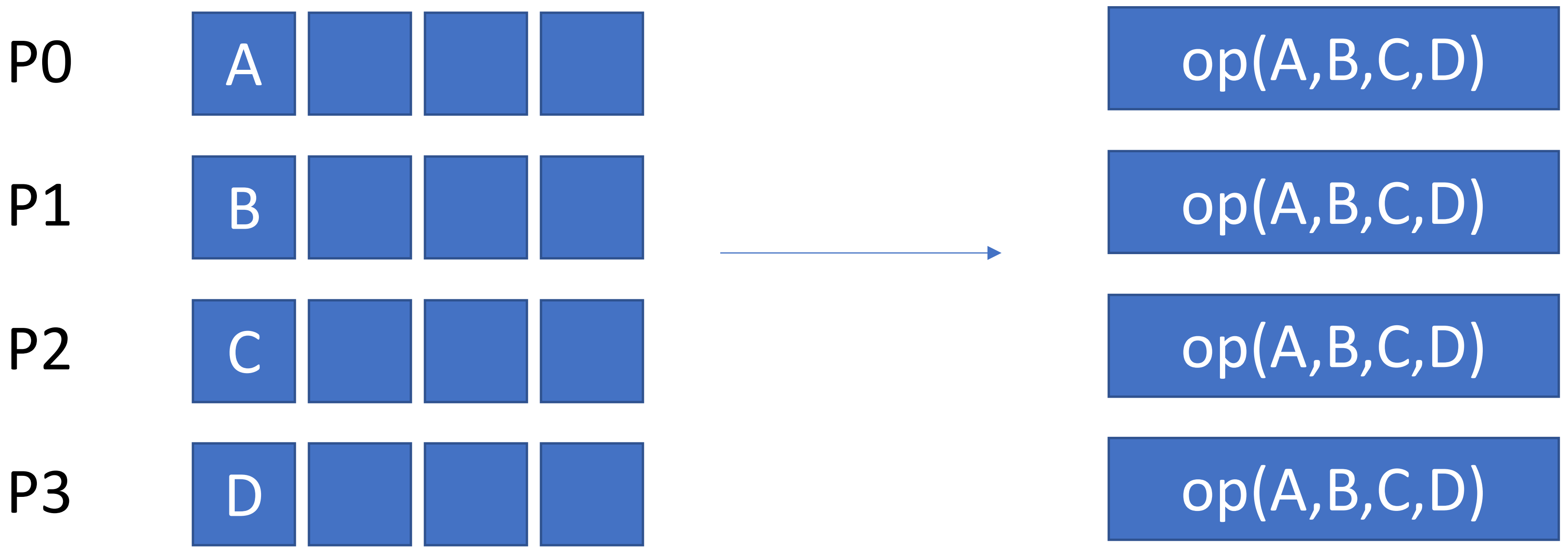
All-to-All

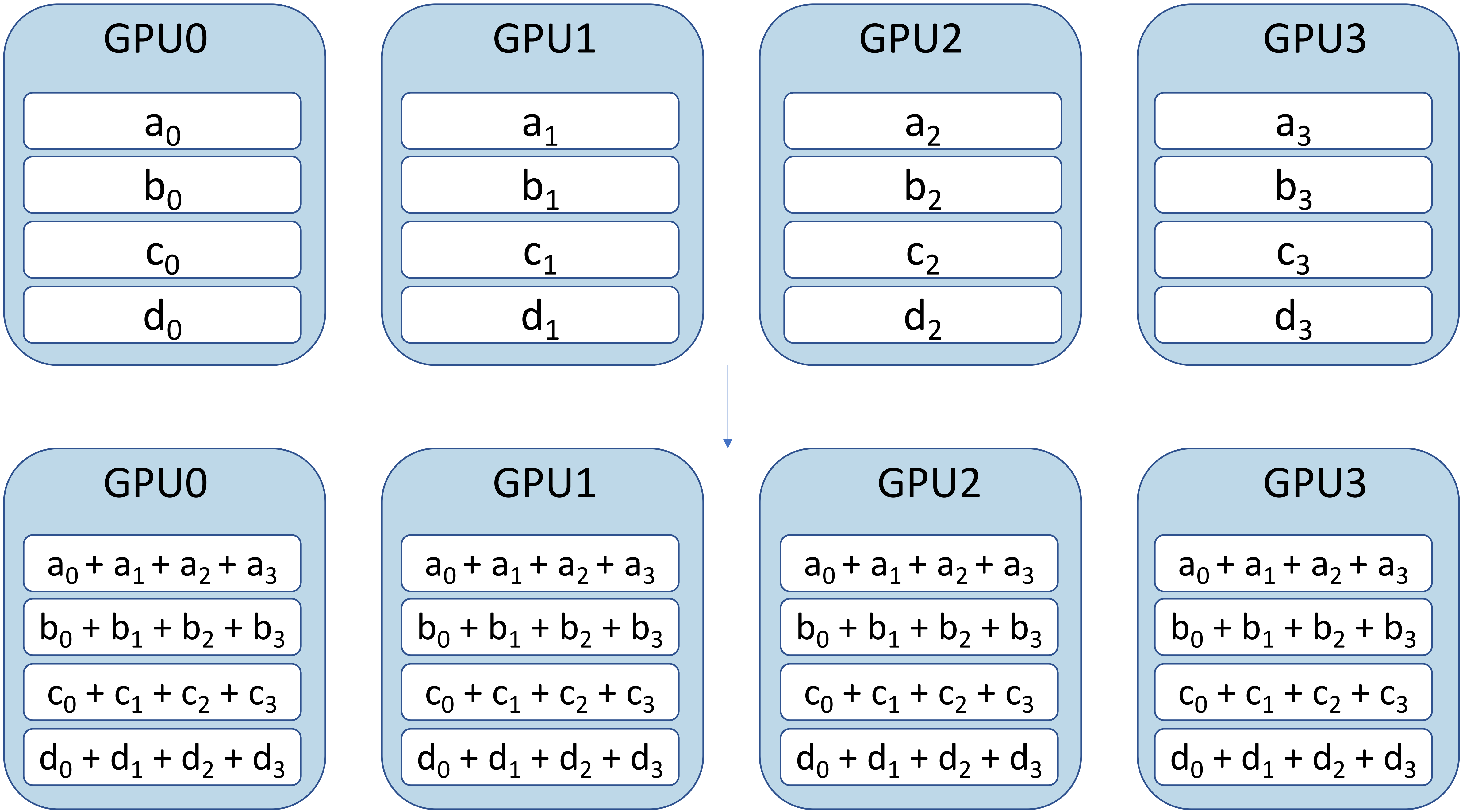

P0
P1
P2
P3
A0 A1 A2 A3
B0 B1 B2 B3
C0 C1 C2 C3
D0 D1 D2 D3
A0 B0 C0 D0
A1 B1 C1 D1
A2 B2 C2 D2
A3 B3 C3 D3

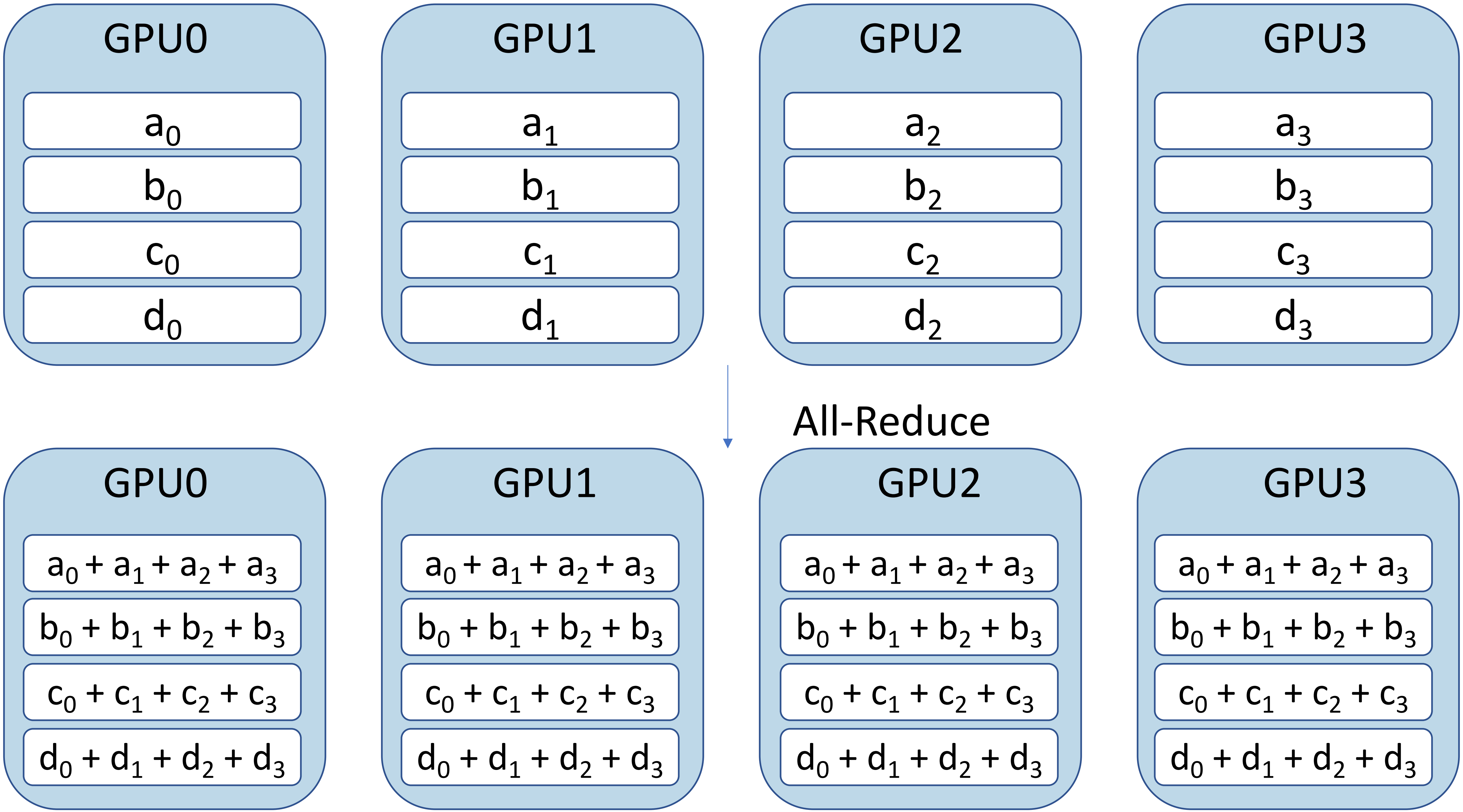
Reduce



All-Reduce




GPU0
a0
b0
c0
d0
GPU1
a1
b1
c1
d1
GPU2
a2
b2
c2
d2
GPU3
a3
b3
c3
d3
GPU0
a0 + a1 + a2 + a3
b0 + b1 + b2 + b3
c0 + c1 + c2 + c3
d0 + d1 + d2 + d3
GPU1
a0 + a1 + a2 + a3
b0 + b1 + b2 + b3
c0 + c1 + c2 + c3
d0 + d1 + d2 + d3
GPU2
a0 + a1 + a2 + a3
b0 + b1 + b2 + b3
c0 + c1 + c2 + c3
d0 + d1 + d2 + d3
GPU3
a0 + a1 + a2 + a3
b0 + b1 + b2 + b3
c0 + c1 + c2 + c3
d0 + d1 + d2 + d3

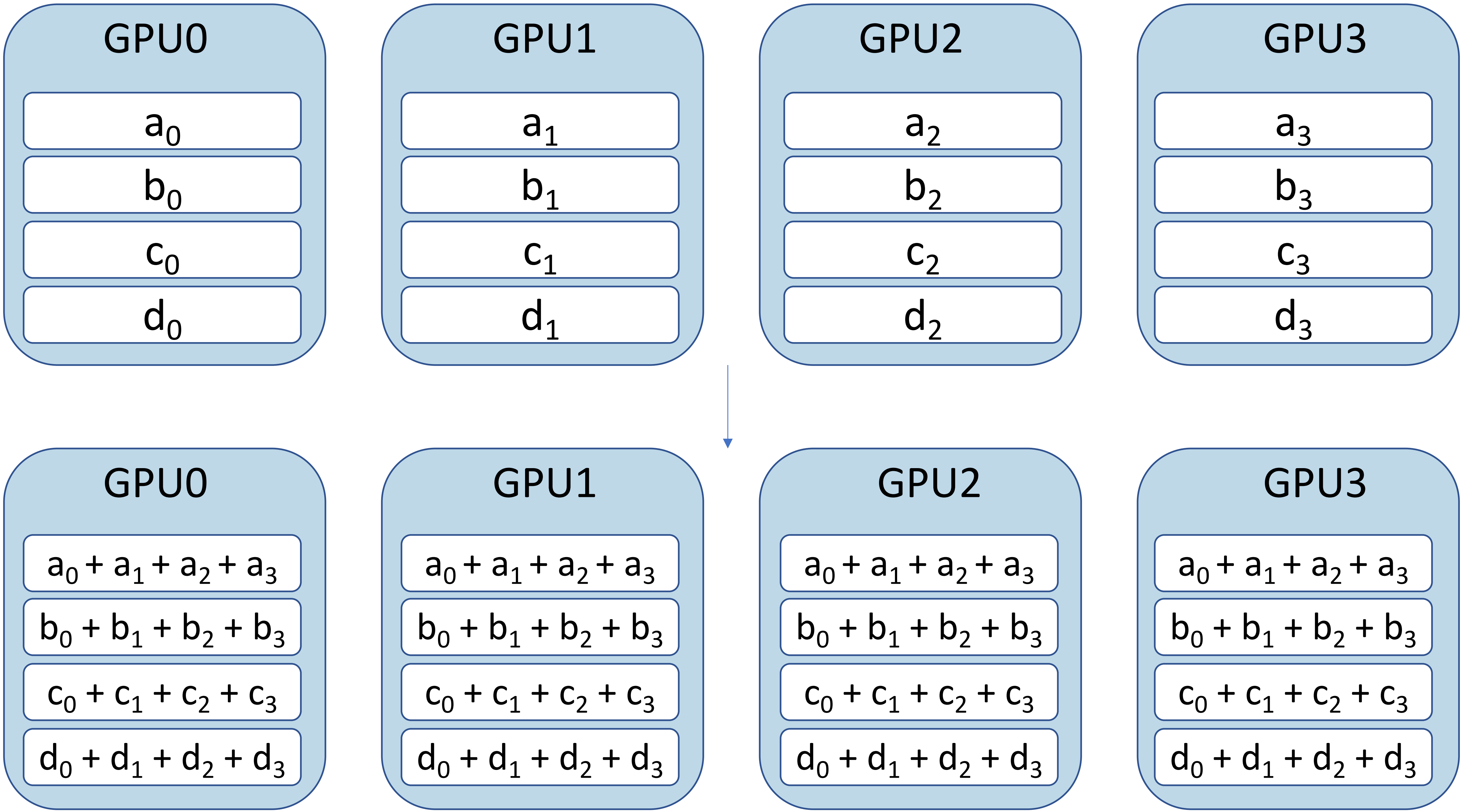

GPU0
a_0
b_0
c_0
d_0
GPU1
a_1
b_1
c_1
d_1
GPU2
a_2
b_2
c_2
d_2
GPU3
a_3
b_3
c_3
d_3
All-Reduce
GPU0
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU1
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU2
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU3
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$

All-Reduce

- “Operation that reduces a set of arrays on distributed workers to a single array that is then redistributed back to each worker”

All-Reduce

- Metrics
  - Bandwidth (number of messages)



GPU0
a_0
b_0
c_0
d_0
GPU1
a_1
b_1
c_1
d_1
GPU2
a_2
b_2
c_2
d_2
GPU3
a_3
b_3
c_3
d_3
GPU0
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU1
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU2
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$
GPU3
$a_0 + a_1 + a_2 + a_3$
$b_0 + b_1 + b_2 + b_3$
$c_0 + c_1 + c_2 + c_3$
$d_0 + d_1 + d_2 + d_3$

Next Time

- Ring-All-Reduce
- Analysis/comparison of Ring All-Reduce vs other strategies
- Asynchronous Gradient Update (strong vs. weak consistency)

Acknowledgments

- Nikita Namjoshi, Google Cloud Developer Advocate