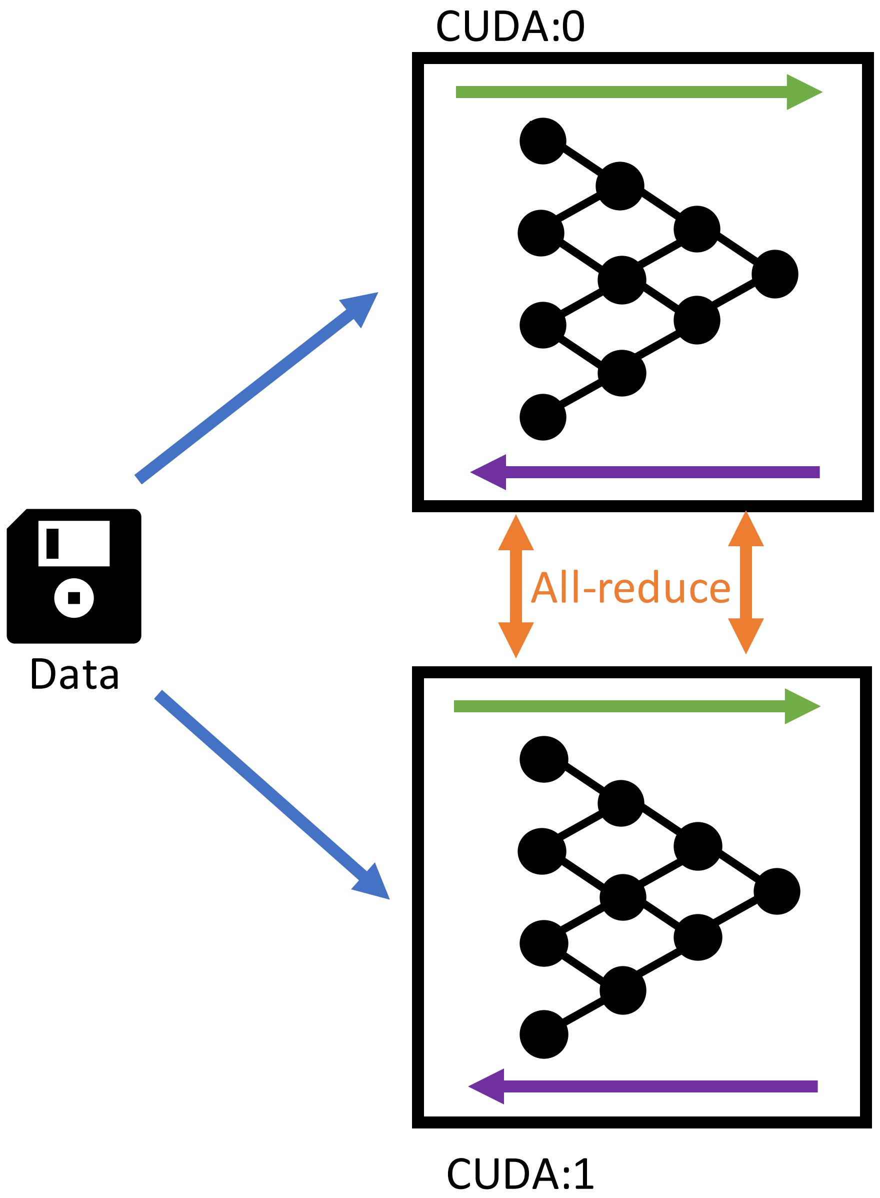
Pipeline Model Parallelism

Amin Saied

Parallelism

Data Parallelism



Models in sync

1. Partition data
2. Forward pass
3. Backward pass
4. All-reduce
5. Update gradients

```python
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Gradient bucketing

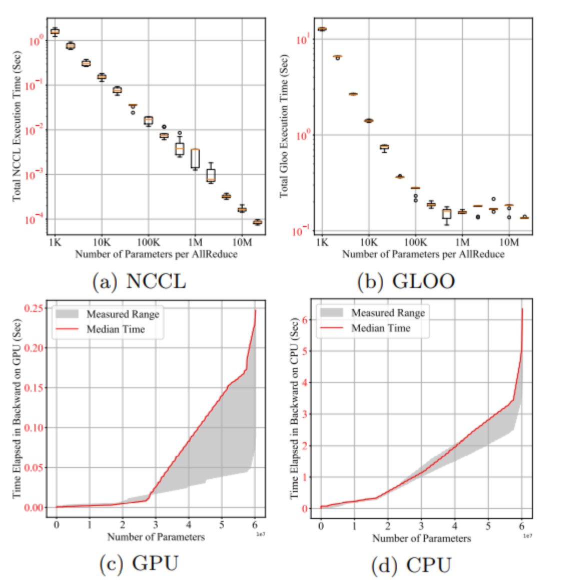
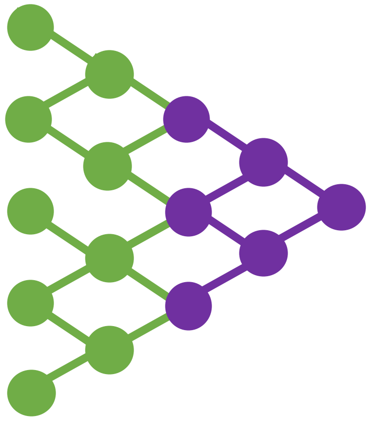


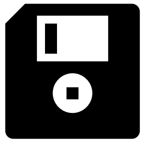
Figure 2: Communication vs Computation Delay

[ ] - PyTorch Distributed: Experiences on Accelerating Data Parallel Training – Li et. al.

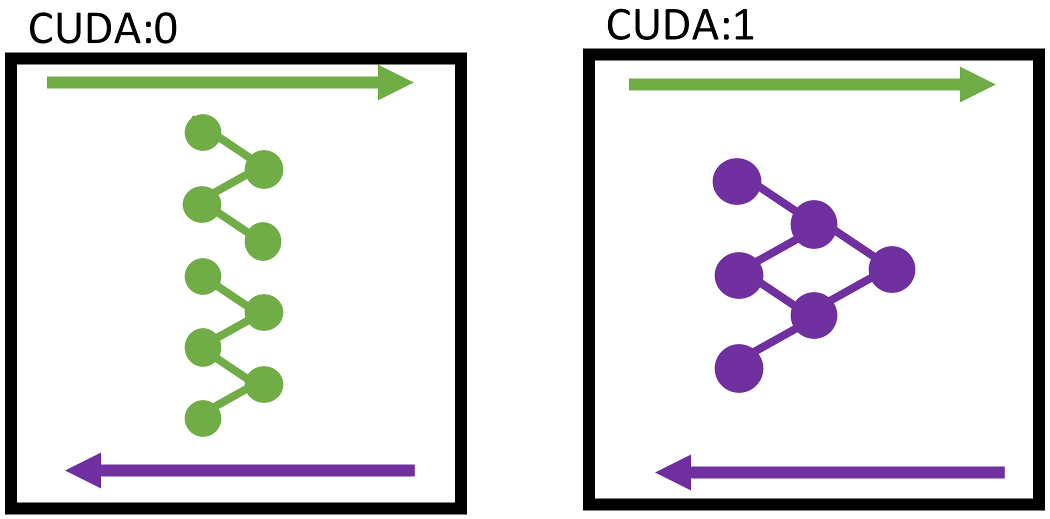
Model Parallelism

Large model





Data



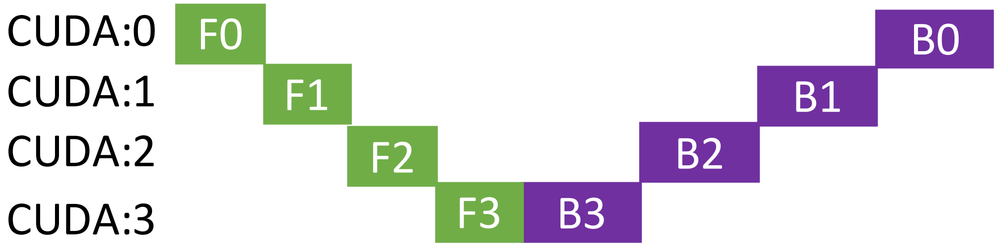
Naïve implementation very simple…

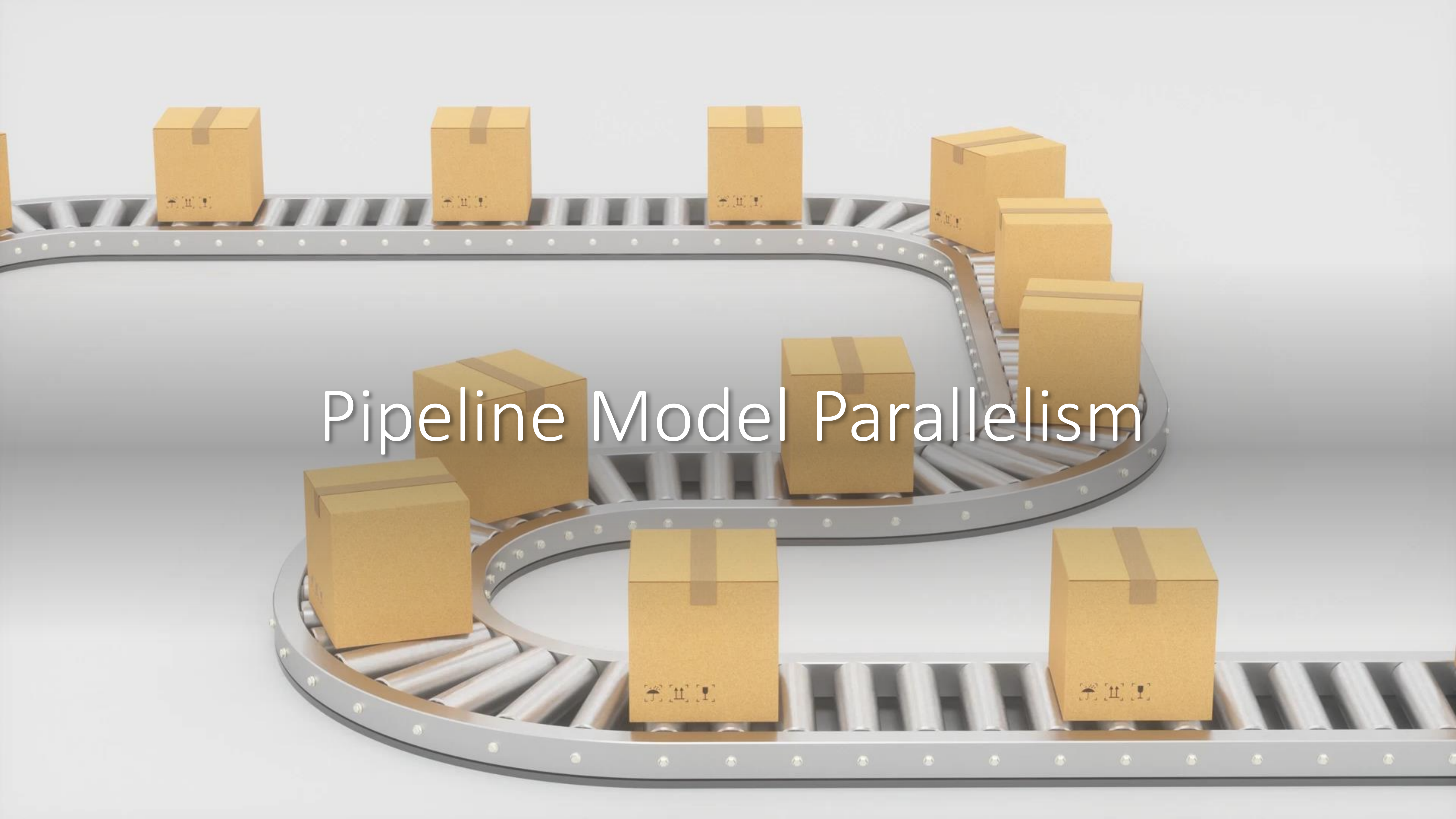
```
import torch
import torch.nn as nn
import torch.optim as optim


class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

```
labels = torch.randn(20, 5).to('cuda:1')
```

https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html

…very inefficient





Pipeline Model Parallelism

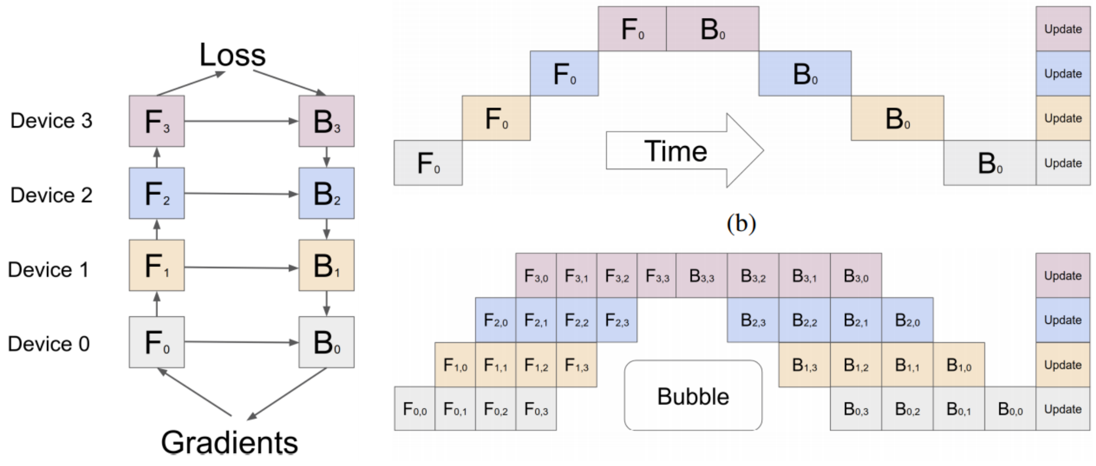
Pipeline Model Parallelism

Pipeline model parallelism:

- Model has k partitions $\mu_1, \dots, \mu_k$
- Break mini-batch D into m **micro-batches** D_i
- Let $F_{i,j}$ denote forward pass of D_j through μ_i
- Let $B_{i,j}$ denote the backward pass similarly

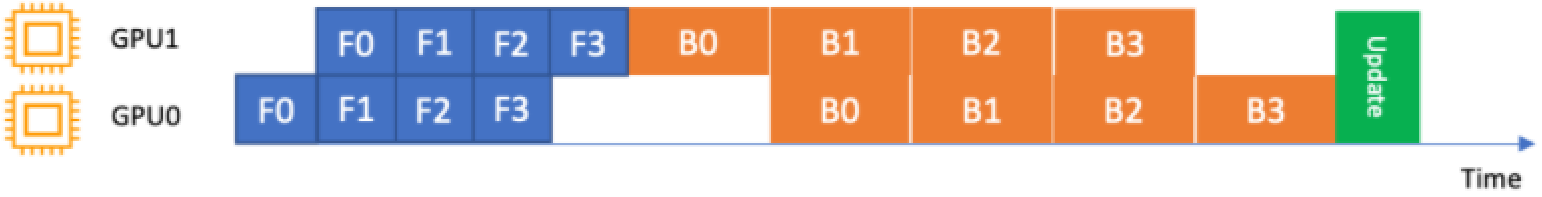
Key upshot:

- $F_{*,j+1}$ can proceed **immediately** after $F_{*,j}$
- $B_{*,j-1}$ can proceed **immediately** after $B_{*,j}$

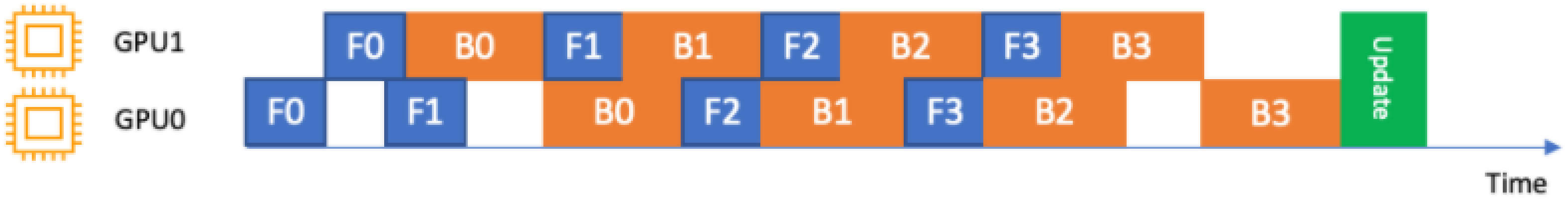


[ ] - GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism – Huang et. al

Simple



Interleaved



https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-core-features.html

Partitioning your model

TensorFlow

- Static computation graph
- tf.Operation
- Model == DAG of tf.Operations

PyTorch

- Dynamic computation graph
- nn.Module
- Model = Sequence of submodules, OR
- Trace model -> build graph of nn.Modules

Optimize for:

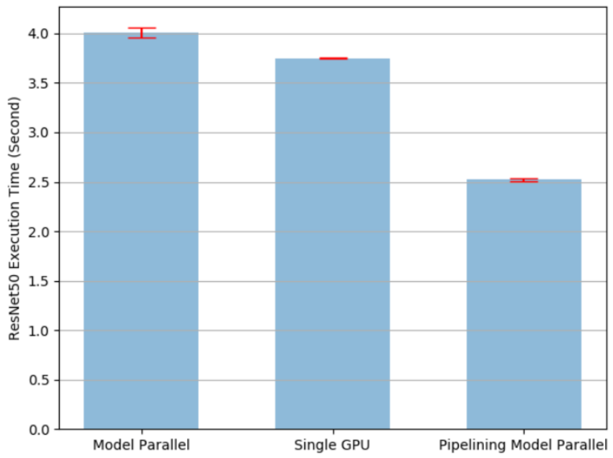
- Speed: min(Var(compute time))
- Memory: min(Var(num parameters))

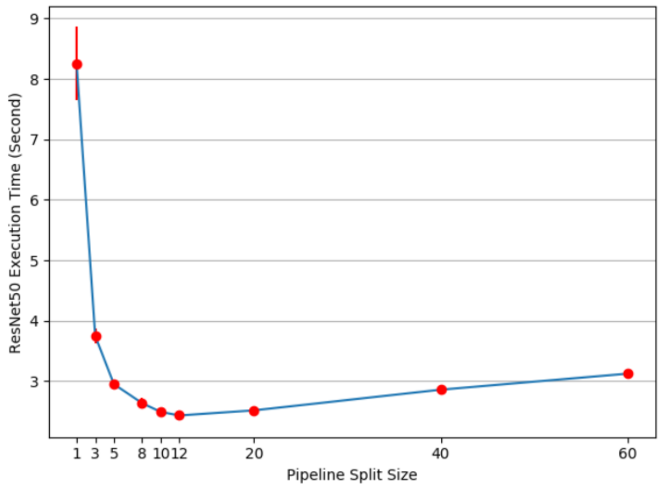
DIY

```python
class ModelParallelResNet50(ResNet):
    def __init__(self, *args, **kwargs):
        super(ModelParallelResNet50, self).__init__(
            Bottleneck, [3, 4, 6, 3], num_classes=num_classes, *args, **kwargs)

        self.seq1 = nn.Sequential(
            self.conv1,
            self.bn1,
            self.relu,
            self.maxpool,

            self.layer1,
            self.layer2
        ).to('cuda:0')

        self.seq2 = nn.Sequential(
            self.layer3,
            self.layer4,
            self.avgpool,
        ).to('cuda:1')

        self.fc.to('cuda:1')

    def forward(self, x):
        x = self.seq2(self.seq1(x).to('cuda:1'))
        return self.fc(x.view(x.size(0), -1))
```

```python
class PipelineParallelResNet50(ModelParallelResNet50):
    def __init__(self, split_size=20, *args, **kwargs):
        super(PipelineParallelResNet50, self).__init__(*args, **kwargs)
        self.split_size = split_size

    def forward(self, x):
        splits = iter(x.split(self.split_size, dim=0))
        s_next = next(splits)
        s_prev = self.seq1(s_next).to('cuda:1')
        ret = []

        for s_next in splits:
            # A. s_prev runs on cuda:1
            s_prev = self.seq2(s_prev)
            ret.append(self.fc(s_prev.view(s_prev.size(0), -1)))

            # B. s_next runs on cuda:0, which can run concurrently with A
            s_prev = self.seq1(s_next).to('cuda:1')

        s_prev = self.seq2(s_prev)
        ret.append(self.fc(s_prev.view(s_prev.size(0), -1)))

        return torch.cat(ret)
```

- Probably don’t do this
- Use a framework /library





https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html

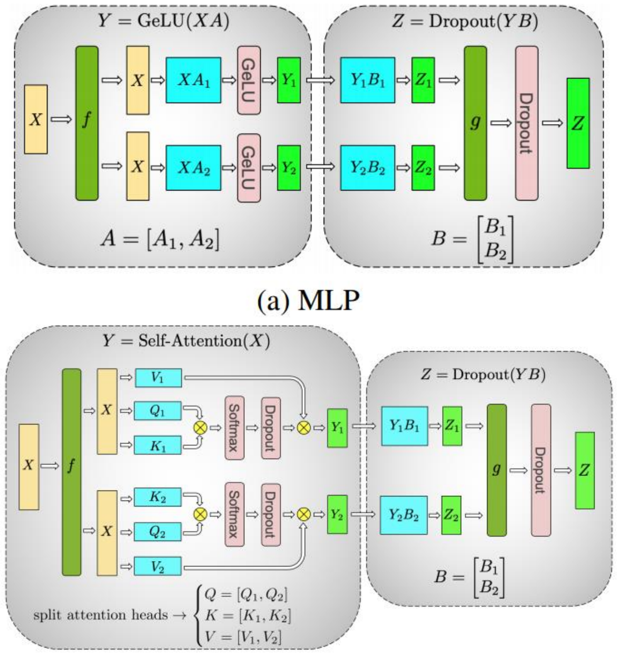
Frameworks

- Lingvo
  - TensorFlow only
  - Implements Gpipe
- Fairscale
  - PyTorch
  - Automatic splitting
  - Requires nn.Sequential
- SageMaker
  - TensorFlow and PyTorch
  - Automatic partitioning
    - Optimize for speed or memory
  - Simple/interleaved pipelines
  - Available in SageMaker Python SDK
  - Integrated with DDP
- Deepspeed
  - PyTorch
  - 3D-parallelism

Tensor Model Parallelism

Tensor Model Parallelism

- Another paradigm in model parallelism
- Split tensor between devices
- Architecture specific and nuanced
- Can reduce the effective batch size to be less than 1 per GPU
- E.g. Megatron-LM from NVIDIA [ ]



(a) MLP

(b) Self-Attention

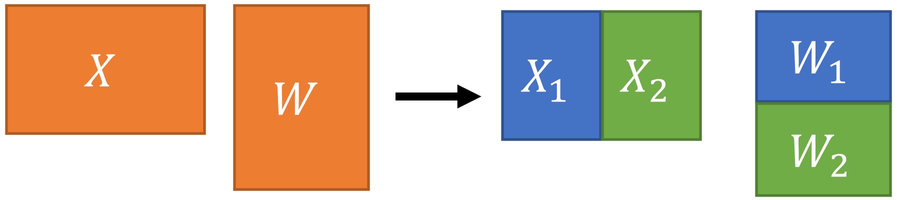
Example

Consider the operation $Z = f(X \cdot W)$ where:

- $X \in R^{n \times m}, W \in R^{m \times l}$
- f non-linearity e.g. ReLU

Consider two ways to parallelize:

1.



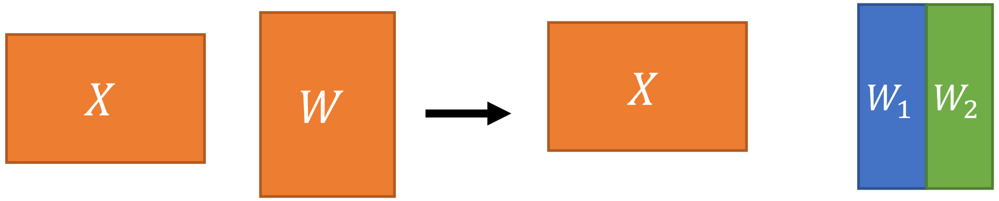
Split X along columns, W along rows:

$$X = [X_1, X_2], W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}$$

Then

$$Z = f(X_1 W_1 + X_2 W_2) \neq f(X_1 W_1) + f(X_2 W_2)$$

2.



Split W along its columns

$$[Z_1, Z_2] = [f(XW_1), f(XW_2)]$$

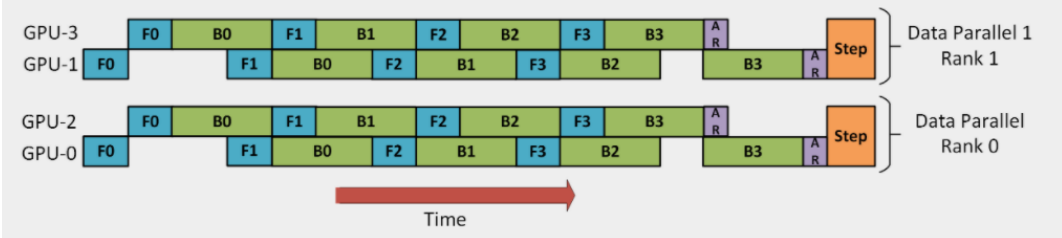
=> f can be applied on each GPU

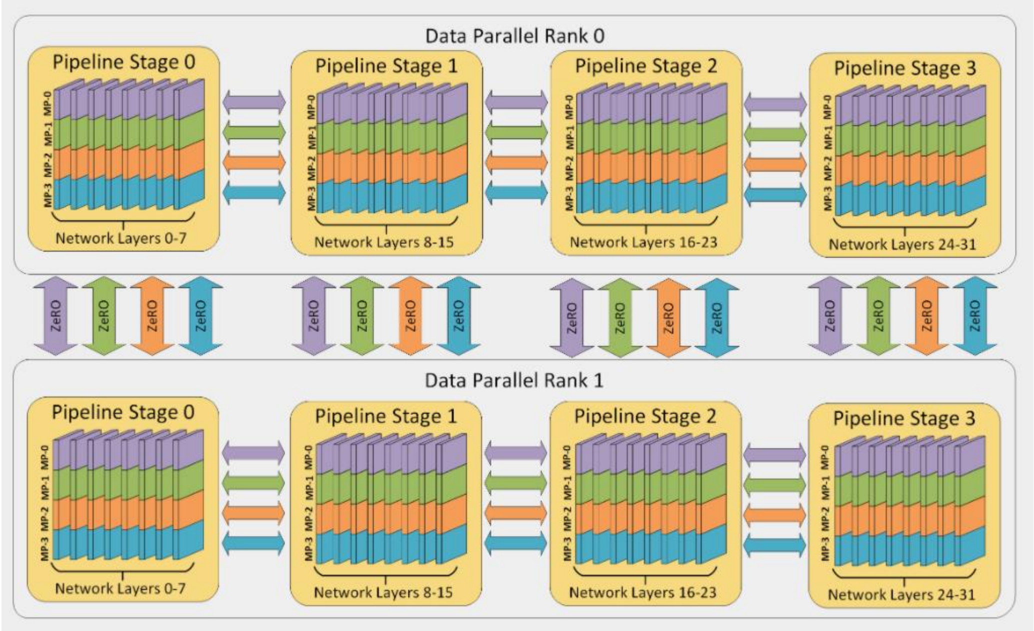
https://github.com/NVIDIA/Megatron-LM

[ ] - Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism – Shoeybi et. Al.

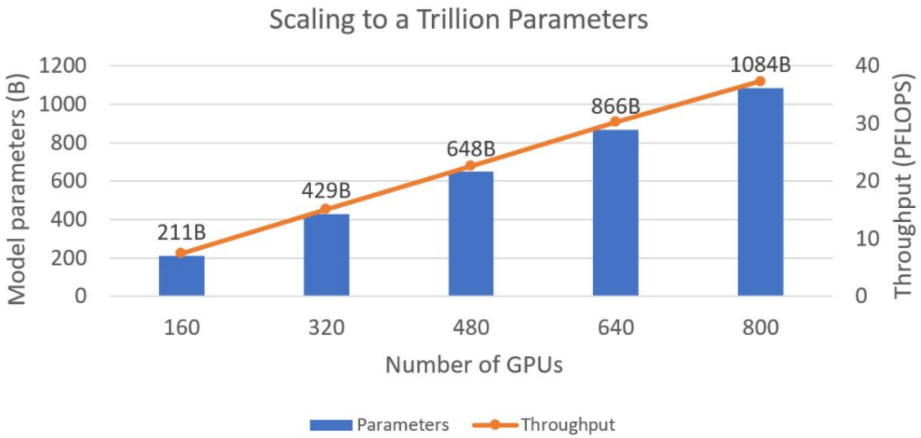
Deepspeed

- Hybrid data and model parallelism
- 3D-parallelism:
  - Data parallel
  - Pipeline model parallel
  - Tensor model parallel
- ZeRO:
  - Memory optimization techniques
- Train models with > 1 trillion parameters





https://www.deepspeed.ai/tutorials/pipeline/



https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

[ ] - ZeRO: Memory Optimizations Toward Training Trillion Parameter Models - Rajbhandari et. Al.

Announcements

- Azureml-examples: deepspeed + transformers
  - Let me know if you're interested!
- ERL2 library → deepspeed trainer w/ Han
- More goodness:
  - 1-bit adam
  - Sparse attention
  - FP16
- Resources:
  - https://www.deepspeed.ai/
  - https://github.com/microsoft/DeepSpeed
  - azureml-examples/deepspeed
  - Example on azureml