Chapter 6

Pipeline input and Layer Split
Gpipe, PipeDream

SK telecom

개요

1. 이장의 주요 목표

- Pipeline Parallelism이 대표적인 기법인 Gpipe와 PipeDream의 소개

2. 주요 논의 내용

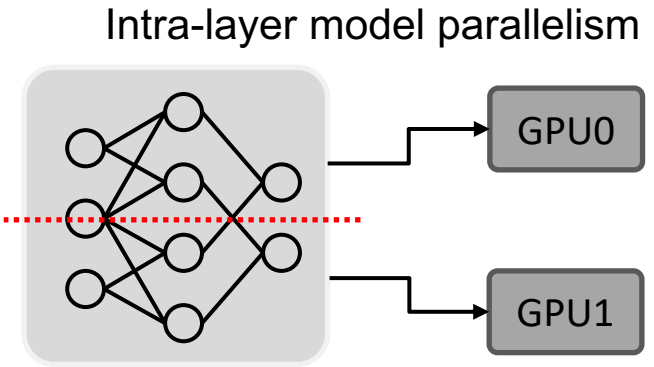
- Gpipe와 PipeDream의 특징 이해

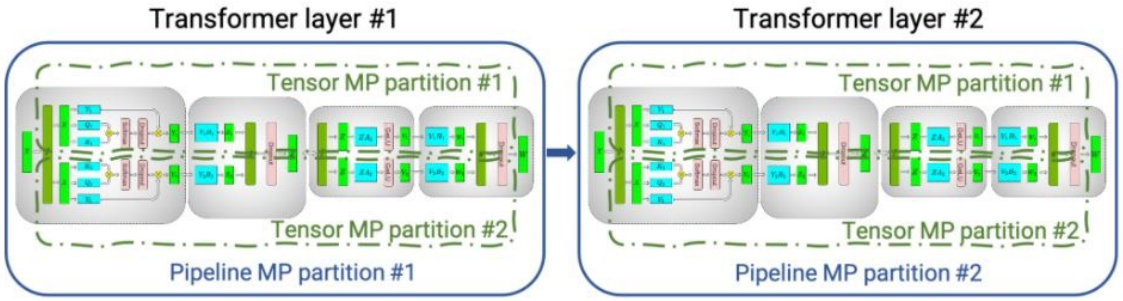
3. 참고

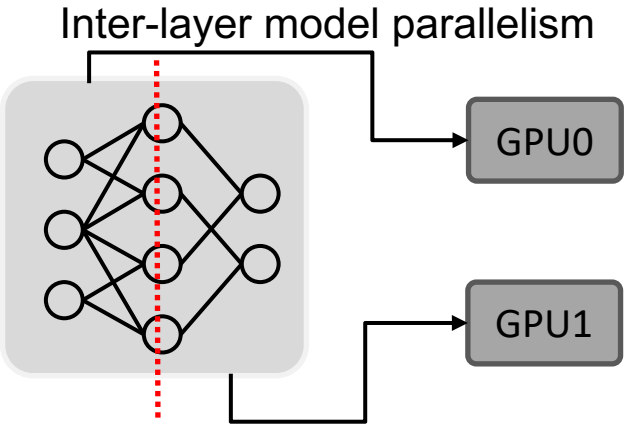
- GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
  - https://arxiv.org/pdf/1811.06965.pdf
- PipeDream: Generalized Pipeline Parallelism for DNN Training
  - https://www.microsoft.com/en-us/research/uploads/prod/2019/08/pipedream.pdf

Model Parallelism

1. **Intra-layer model parallelism**
   - 모델의 Tensor를 쪼개는 방식
   - Tensor parallel
2. **Inter-layer model parallelism**
   - 모델의 Layer를 기준으로 쪼개는 방식
   - Pipeline parallel





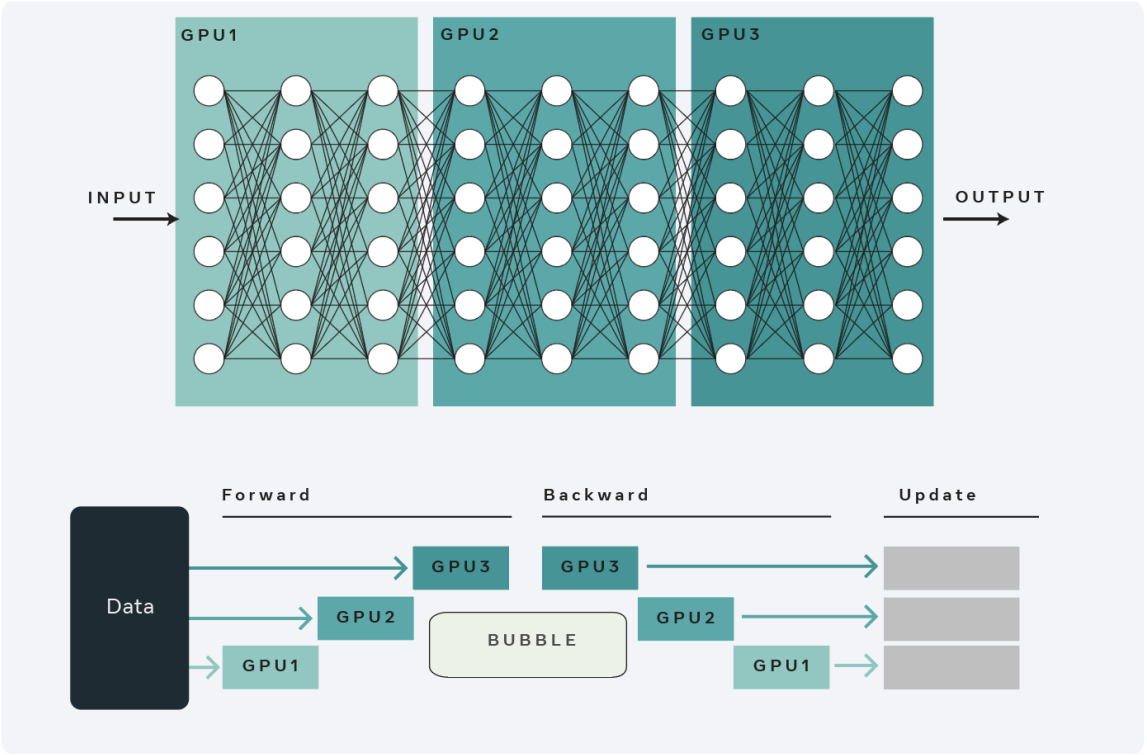


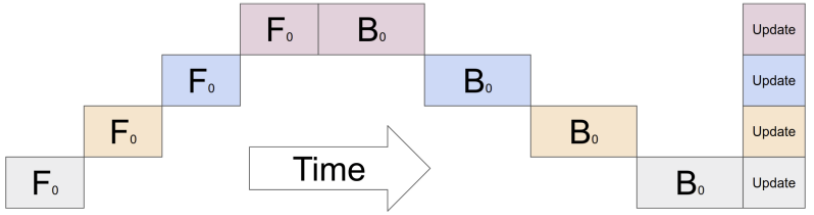
Inter-Layer 병렬화

1. Inter-layer 병렬화의 비효율성

- Inter₩-layer 병렬화는 Layer wise하게 병렬화하기 때문에 Layer 단위로 순차적 연산 진행
- 하지만 이러한 순차적인 종속성 때문에 한 Layer가 연산을 수행중일 때 다른 Layer에 할당된 GPU는 놀 수 밖에 없음

Pipeline Parallelism

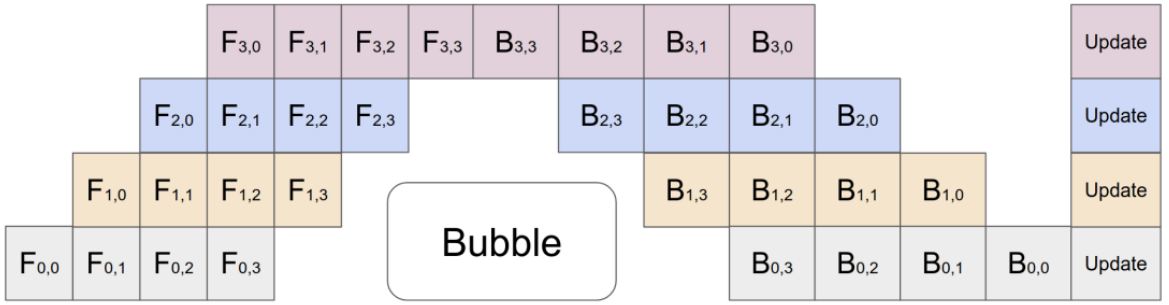


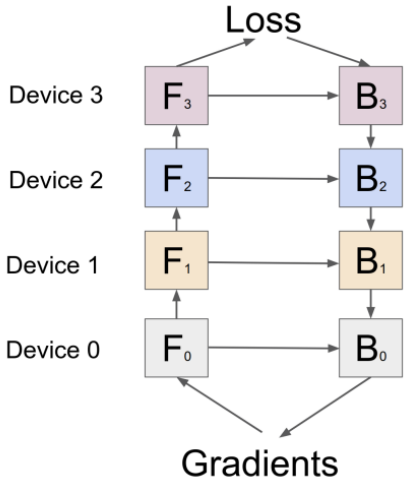


GPipe

1. Inter layer 병렬화의 비효율성 개선

- Idle 상태의 GPU들을 줄일 수 없을까 하는 고민에서 시작
- Mini-batch를 더 잘게 쪼개 micro-batch 단위(batch-splitting)로 연산 과정을 pipelining 함
- 결과적으로 idle 상태 (bubble time)을 줄일 수 있음





GPipe

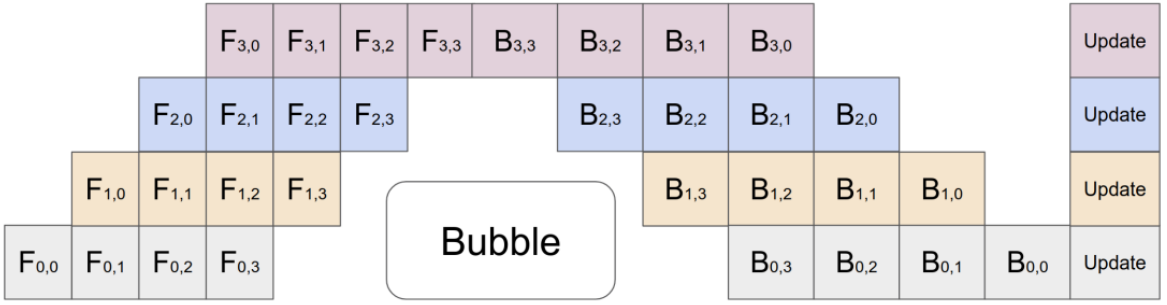
1. Bubble time

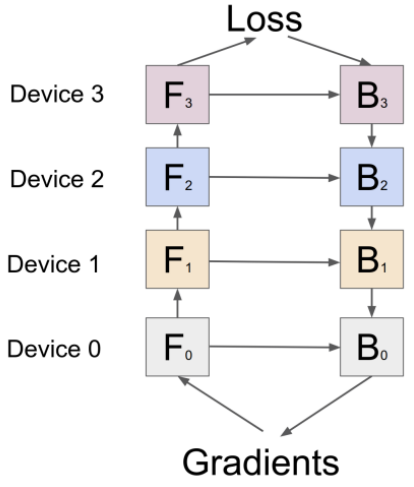
$$O(\frac{K-1}{M+K-1})$$

- M : number of micro-steps
- K : number of model partitions

2. Bubble time의 최소화

- $M \geq 4K$ 일때 최소화





GPipe

1. Training throughput

- 같은 파티션 수에서 micro batch를 늘릴 수 록 학습량이 늘어남

TPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 1$	1	1.13	1.38	1	1.07	1.3
$M = 4$	1.07	1.26	1.72	1.7	3.2	4.8
$M = 32$	1.21	1.84	3.48	1.8	3.4	6.3

GPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 32$	1	1.7	2.7	1	1.8	3.3

- M : number of micro-steps
- K : number of model partitions

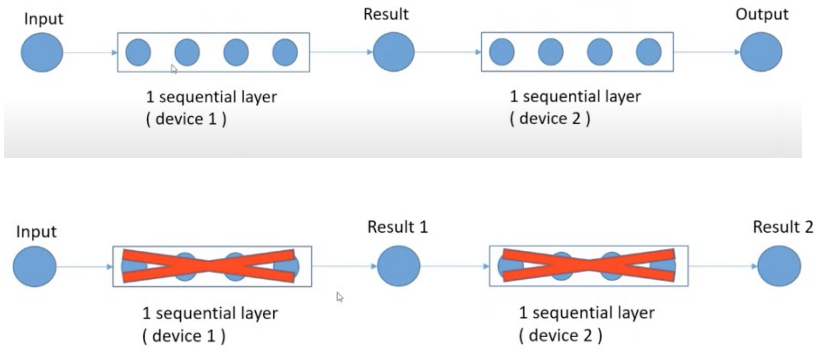
GPipe

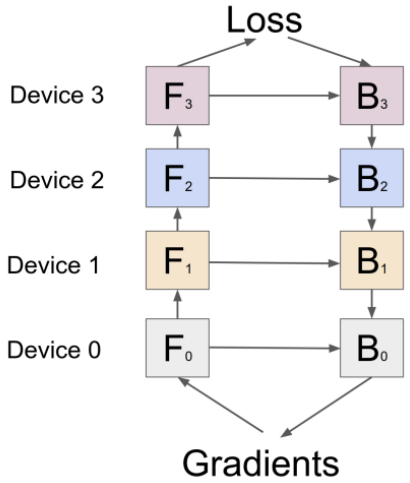
1. Activation Memory

- Neural Network 에서 BP를 계산할 때는 FP 과정에의 Activation function을 미분하기 위해 입력값이 필요
- 일반적으로, N개의 mini batch와 L개의 layer으로 이루어진 Neural에서는 BP를 위해 FP 이후에 N x L개의 activation input 값을 메모리에 유지

2. Gpipe Re-materialization

- GPipe는 하나의 단계(단계 = 여러 Layer들의 그룹)의 마지막 계층에 있는 Activation 값만 유지
- BP가 시작될 때마다 (마지막 계층부터) FP의 activation 값을 다시 계산하여 메모리에 유지 후 완료 되면 삭제
- 이런 방식으로 단계당 하나의 output만 유지하게 되므로 메모리를 획기적으로 절약할 수 있음





GPipe

1. Maximum model size

- Naïve-1은 Gpipe를 사용하지 않음 pipeline-k는 k개의 파티션과 k개의 GPU로 Gpipe를 적용
- Pipeline 파티션이 증가시킴으로써 더 큰 모델 학습 가능

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	**Naive-1**	**Pipeline-1**	**Pipeline-8**	**Pipeline-32**	**Pipeline-128**
Transformer-L	3	13	103	415	1663
# of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

PipeDream

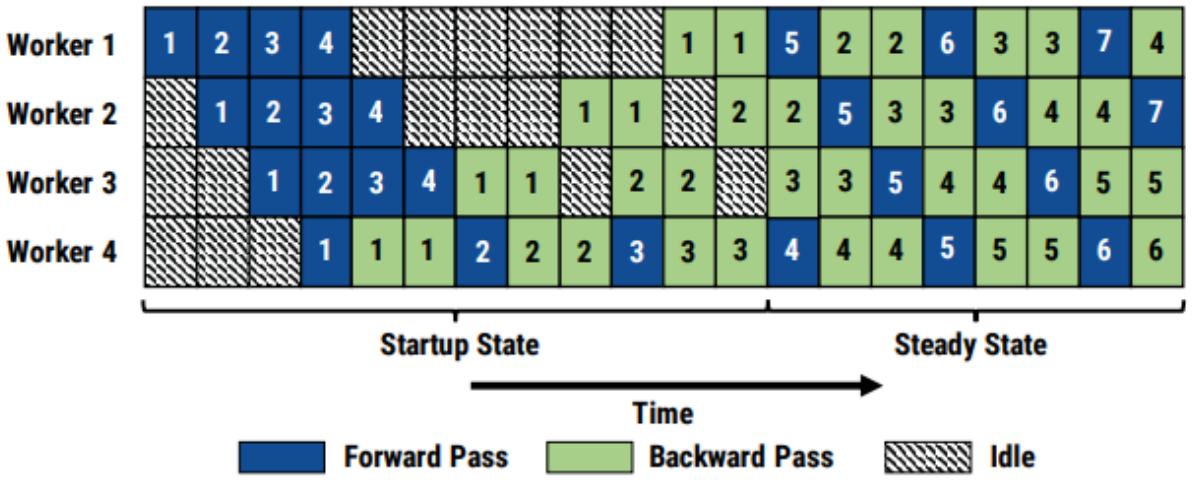
1. Gpipe의 개선점

- Gpipe는 하나의 weight 버전만 유지하며, 주기적으로 Pipeline Flush가 일어남
- Pipeline Flush는 backward 연산을 통해 구해진 Gradient로 weight을 업데이트 하는 과정
- Flush 시간 동안 GPU는 FP, BP 연산을 하지 않아 효율이 떨어짐

PipeDream

1. PipeDream

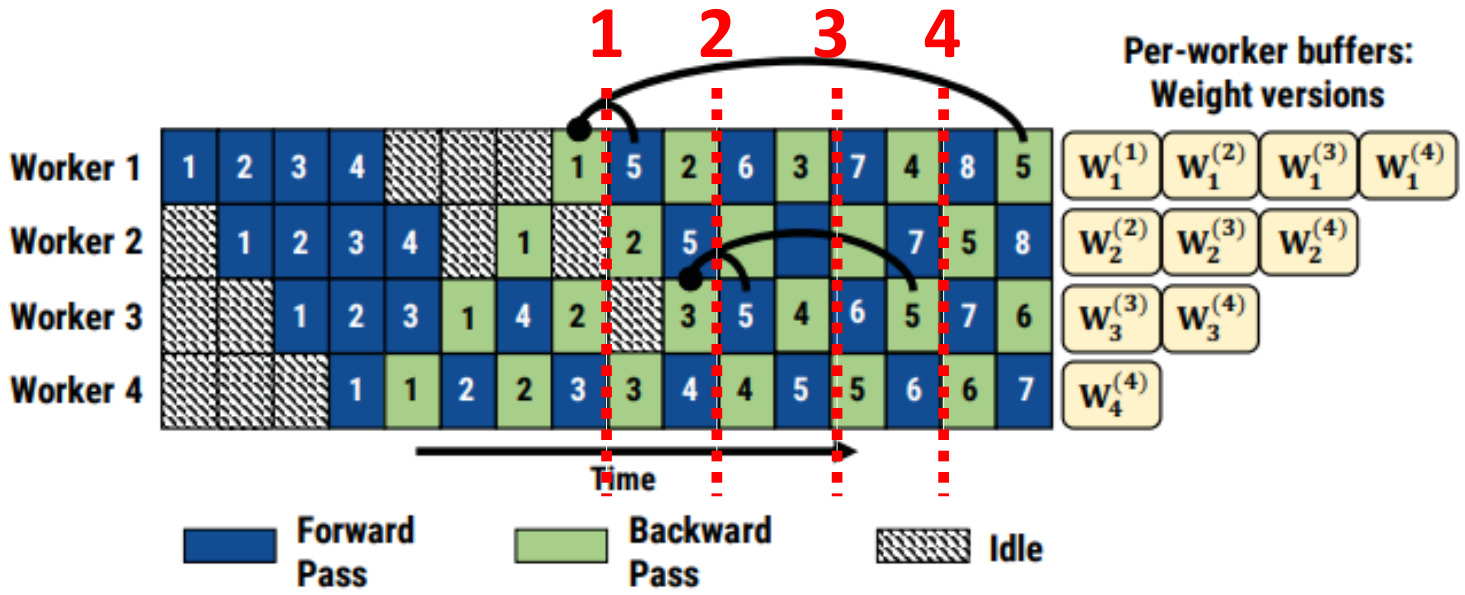
- PipeDream는 여러 버전의 weight를 운용함으로써 전체 flush 과정 없이 연산의 효율을 높임



PipeDream

1. PipeDream

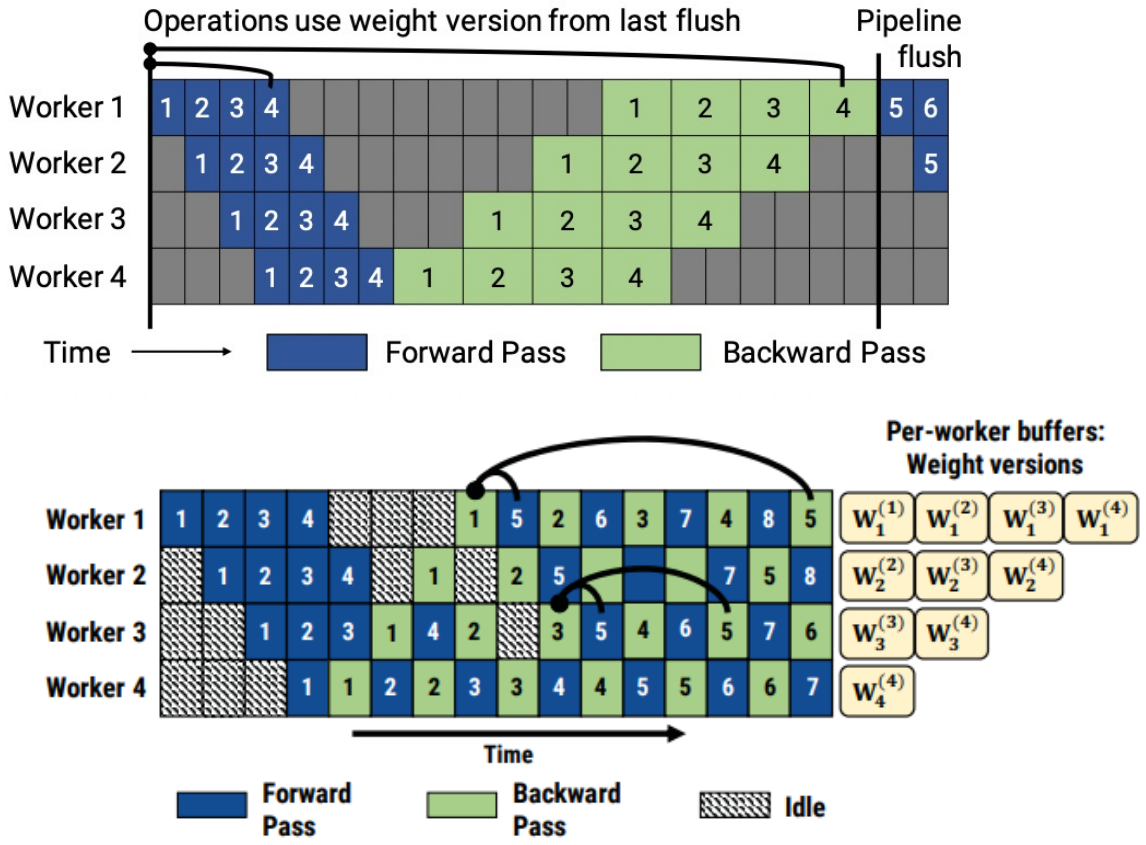
- 그런데 각각의 weight 버전을 따로 운영하면 backward를 거치고 난다음 parameter 업데이트가 일어나게 됨
- 4개 weight 버전을 운영할 때 5번째 microbatch가 연산 될때는 서로 다른 parameter를 통해 연산이 일어나게 되어 학습에 문제가 발생
- 따라서 이렇게 각기 다르게 업데이트된 파라메터들을 저장할 별도의 메모리 공간을 할당



PipeDream vs Gpipe

1. PipeDream과 Gpipe 비교

- Gpipe : Memory 효율적, 연산 비효율적
- PipeDream : Memory 비효율적, 연산 효율적



End of Documents