Parallelism

딥러닝에서의 분산 처리

김정욱

Classic

다룰 주제 1

Deep Learning and GPU memory

1

1 Introduction

Finetuning large language models (LLMs) is a highly effective way to improve their performance, [40, 62, 43, 61, 59, 37] and to add desirable or remove undesirable behaviors [43, 2, 4]. However, finetuning very large models is prohibitively expensive; regular 16-bit finetuning of a LLaMA 65B parameter model [57] requires more than 780 GB of GPU memory. While recent quantization methods can reduce the memory footprint of LLMs [14, 13, 18, 66], such techniques only work for inference and break down during training [65].

"LLaMA 65B 모델을 일반 FP16으로 훈련 시 GPU 메모리가 780GB 이상 필요하다"

2

Alpaca github 내용 중...

Addressing OOM

Naively, fine-tuning a 7B model requires about 7 x 4 x 4 = 112 GB of VRAM. Commands given above enable parameter sharding, so no redundant model copy is stored on any GPU. If you'd like to further reduce the memory footprint, here are some options:

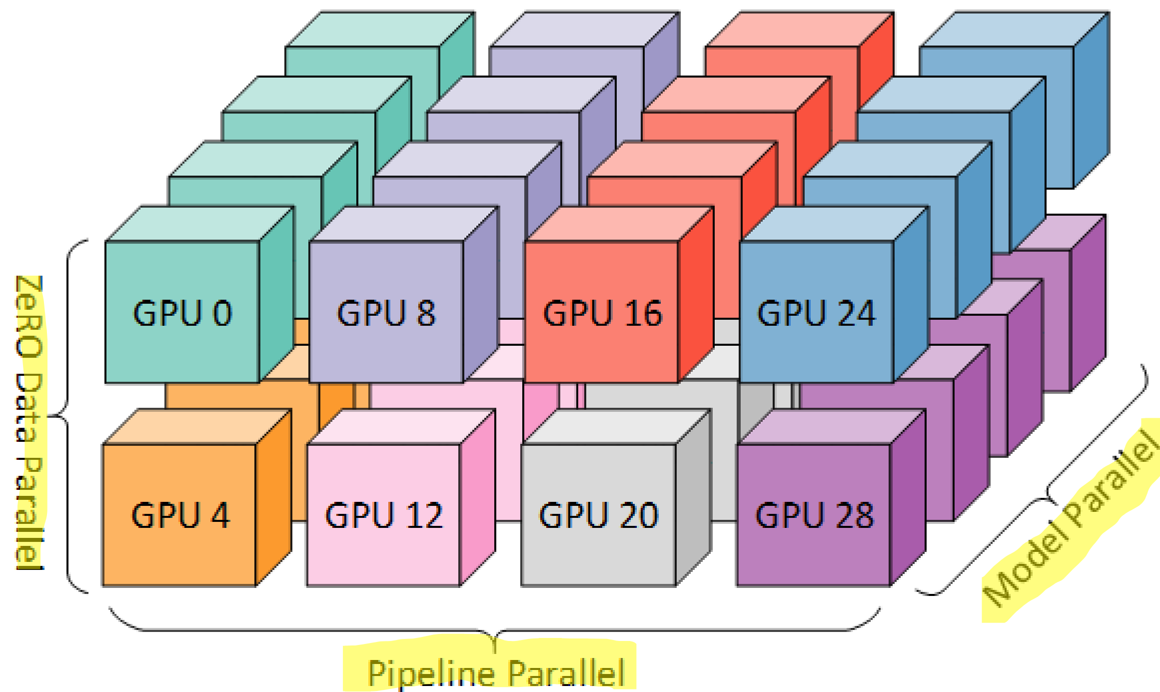
"대충, 7B 모델을 훈련할 때는 7×4×4=112GB 가 필요하다"

(...왜 하필 7×4×4?)

다룰 주제 2

Parallelism의 종류

- Data Parallelism
- Pipeline Parallelism
- Tensor Parallelism
- Model Parallelism
- ZeRO Data Parallelism



DeepSpeed의 3D parallel = ZeRO DP + PP + MP

본 세미나의 효과

- LLM 훈련 시 Multi-GPU 세팅은 필수
- Parallel한 훈련 시 사용하는 DP, PP, TP, MP, ZeRO-DP 정리
- DeepSpeed, FullyShardedDataParallel 등의 개발 문서 이해
- 최근 나오는 PEFT 논문을 이해하기 쉬워짐

(QLoRA에서 발췌)

> PEFT를 위해 디자인된 LoRA에서는 대부분의 메모리가 LoRA parameter가 아닌 activation gradient에서 사용된다. Gradient checkpointing을 사용하면 input gradient는 sequence당 평균 18MB로 감소하기 때문에 모든 LoRA 가중치를 합친 것보다 많다. 반면 제안한 4비트 모델 parameter은 5GB를 사용한다. 그러므로 gradient checkpointing이 중요하고, LoRA parameter를 줄이는 데 집중하는 것은 별로 이득이 없다. 따라서 우리는 큰 메모리 요구 없이 더 많은 어댑터를 사용 가능하다.

- 실제로 LLM을 개발할 때 optimizer, gradient, parameter 중 무엇을 partitioning하고 offload할 건지를 이해하고 선택 가능

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Samyam Rajbhandari*, Jeff Rasley*, Olatunji Ruwase, Yuxiong He
{samyamr, jerasley, olruwase, yuxhe}@microsoft.com

ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, Yuxiong He
{samyamr, olruwase, jerasley, shsmit, yuxhe}@microsoft.com

DeepSpeed ZeRO

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Mohammad Shoeybi [1 2] Mostofa Patwary [1 2] Raul Puri [1 2] Patrick LeGresley [2] Jared Casper [2]
Bryan Catanzaro [2]

Tensor Parallelism

Reference

- HF Model Parallelism: https://huggingface.co/docs/transformers/v4.15.0/parallelism

- HF Efficient Training on Multiple GPU: https://huggingface.co/docs/transformers/main/en/perf_train_gpu_many#tensor-parallelism

- HF Performance and Scalability: https://huggingface.co/docs/transformers/v4.15.0/performance

- Detailed TP overview: https://github.com/huggingface/transformers/issues/10321#issuecomment-783543530

- NVIDIA docs: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

- 계산 그래프를 활용한 편미분 구하기, 역전파법 이해: https://uponthesky.tistory.com/9

- Deepspeed Docs: https://deepspeed.readthedocs.io/en/latest/zero3.html

- PyTorch FSDP docs: https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

\+ ...

Parallelism

-deep dive-

딥러닝에서의 분산 처리

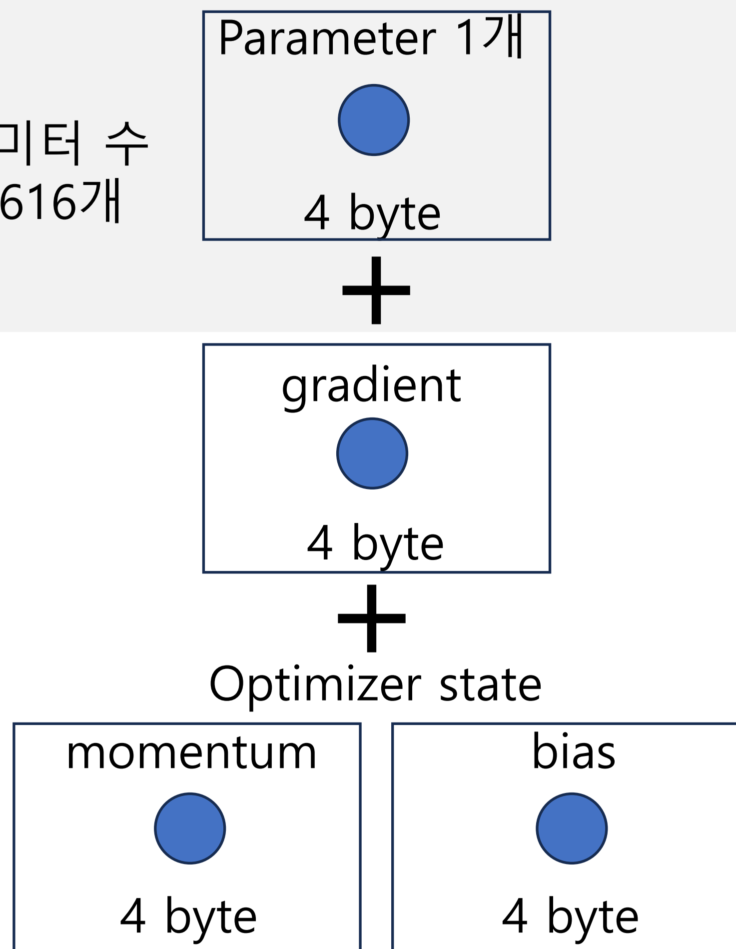
김정욱

다룰 주제 1

Deep Learning and
GPU memory

일반적인 32비트 훈련에 필요한 메모리

LLaMA-7B
전체 파라미터 수
6,738,415,616개



모델을 훈련 없이 GPU에 올리기만 한다면
4 byte * 7 billion = 28GiB

* 앞으로 편의상 기비바이트 대신
뭉뚱그려서 기가바이트 사용

\+

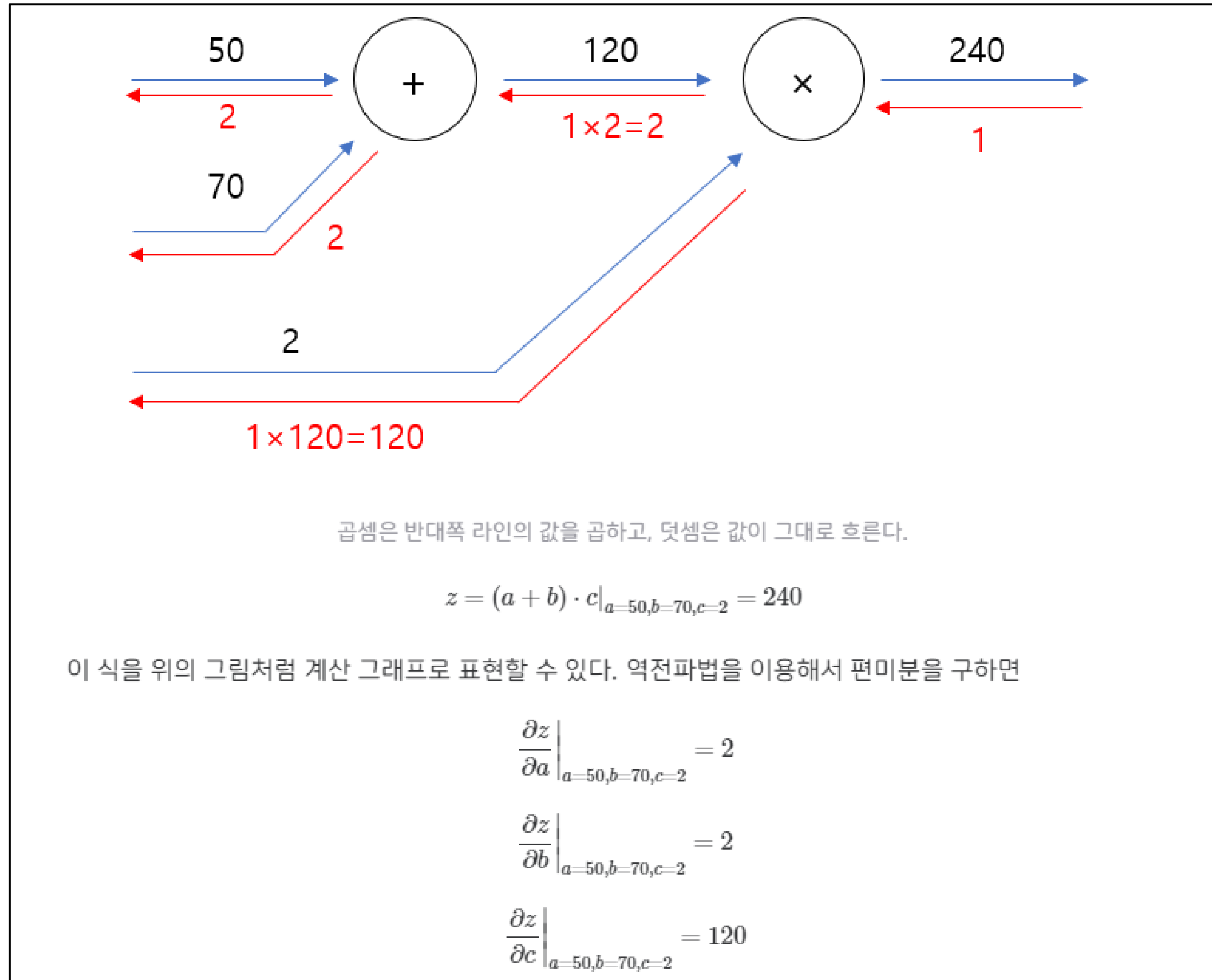
Residual Memory

- Activations (다음 슬라이드에 설명)
- Temporary buffer
- Unusable memory fragments

즉, p+os+g 를 계산하면

16 byte * 7 billion
= 112 GiB

Activation 메모리



Backpropagate로 gradient를 구할 때,

중간에 계산됐던 값들이 모두 필요함!

-> activation memory

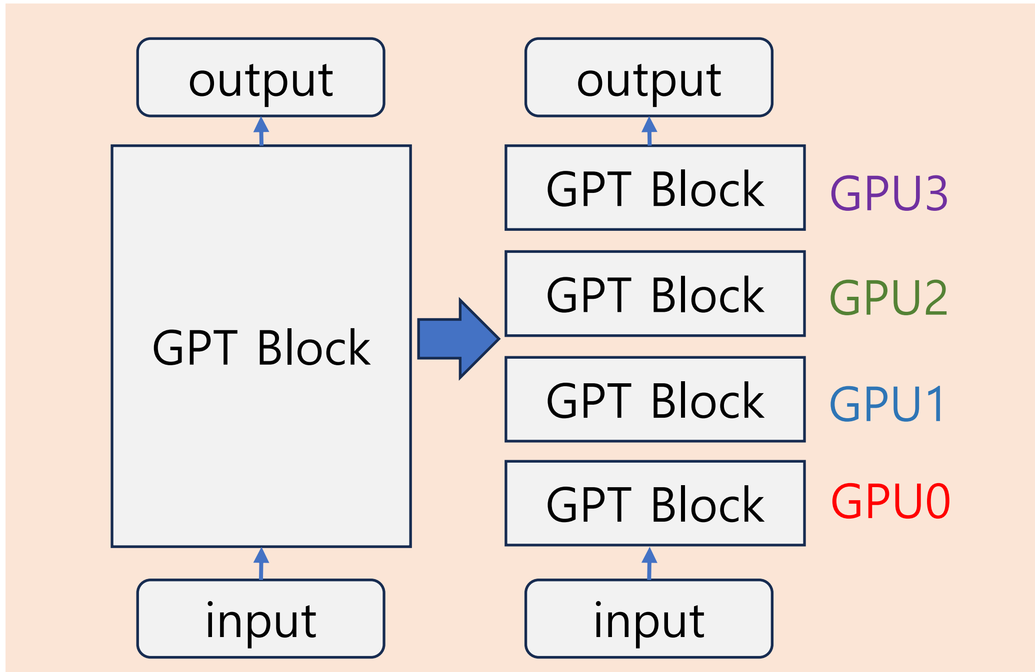
다룰 주제 2

Parallelism의 종류

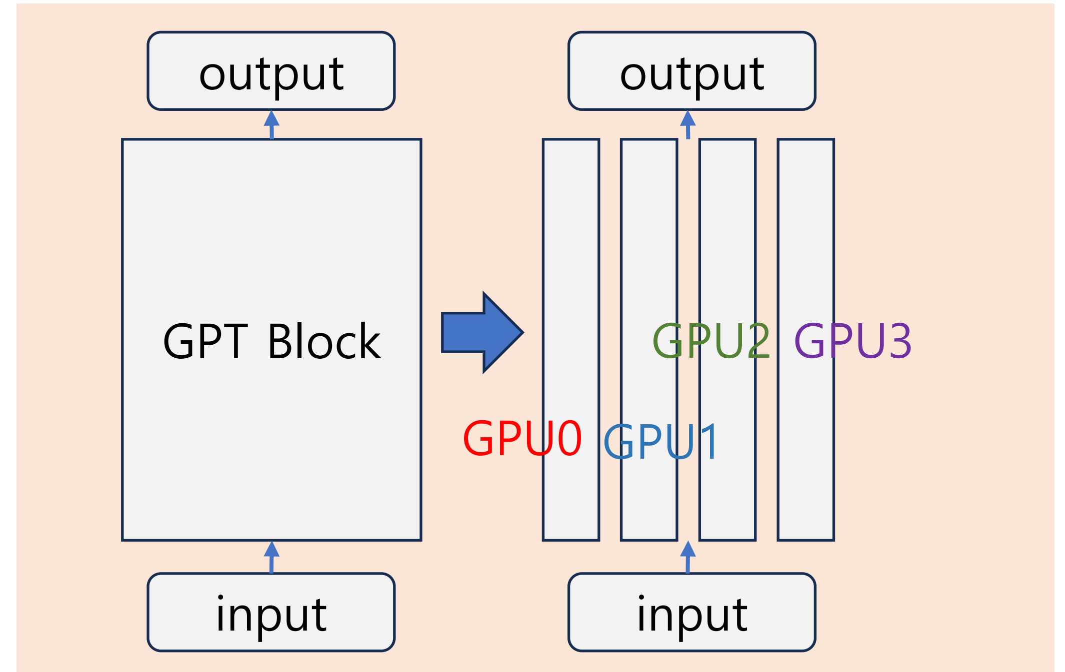
- Data Parallelism
- Pipeline Parallelism
- Tensor Parallelism
- Model Parallelism
- ZeRO Data Parallelism

요약

Pipeline Parallelism (PP)

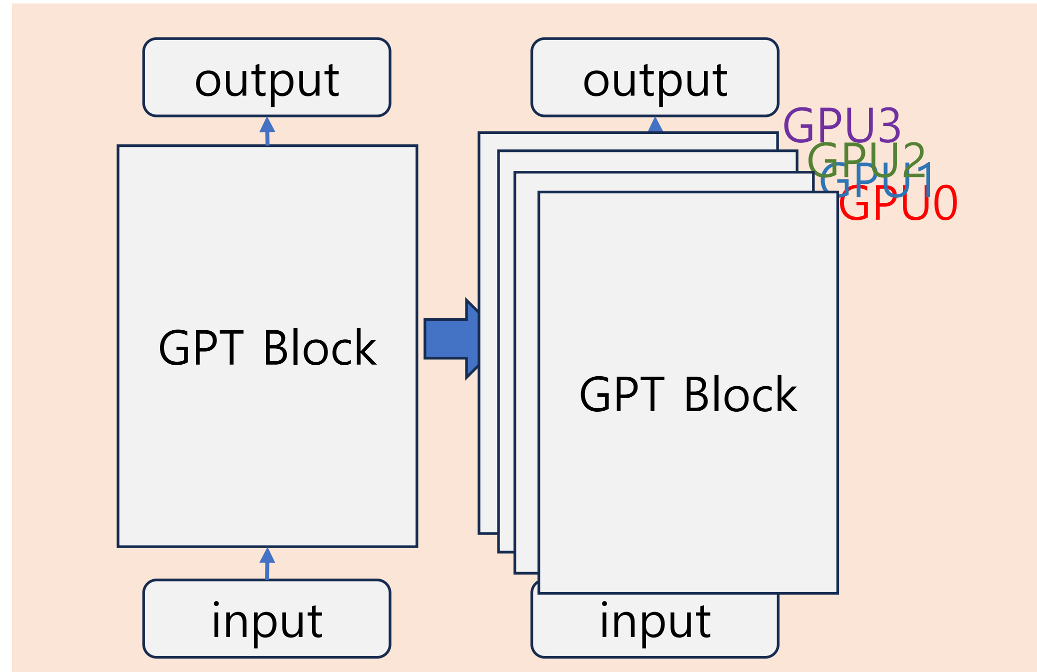


Tensor Parallelism (TP)



요약

Data
Parallelism
(DP)



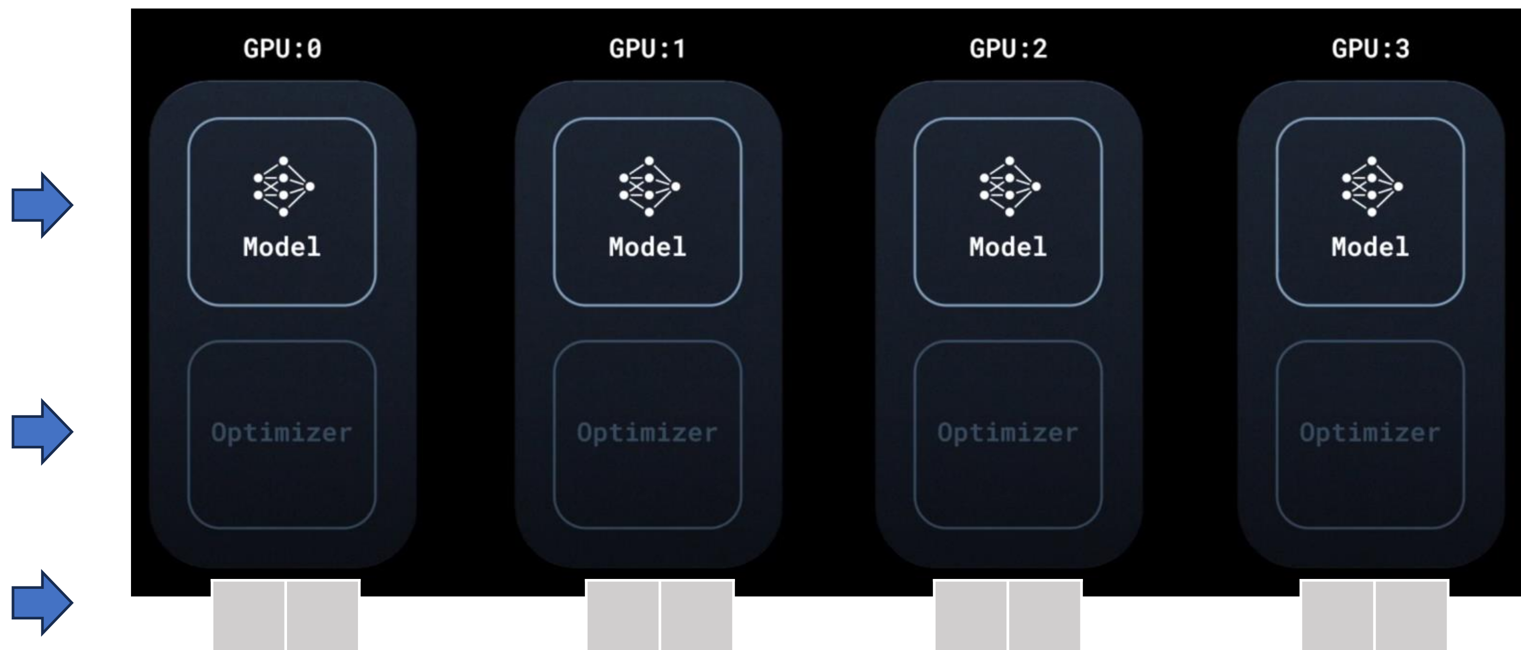
1. Data Parallelism (DP)

1. Data Parallelism (DP)

- 가장 구현이 쉬움

1. 정확히 같은 모델 전체를
각 GPU에 복사

2. 정확히 같은 optimizer state를
각 GPU에 복사

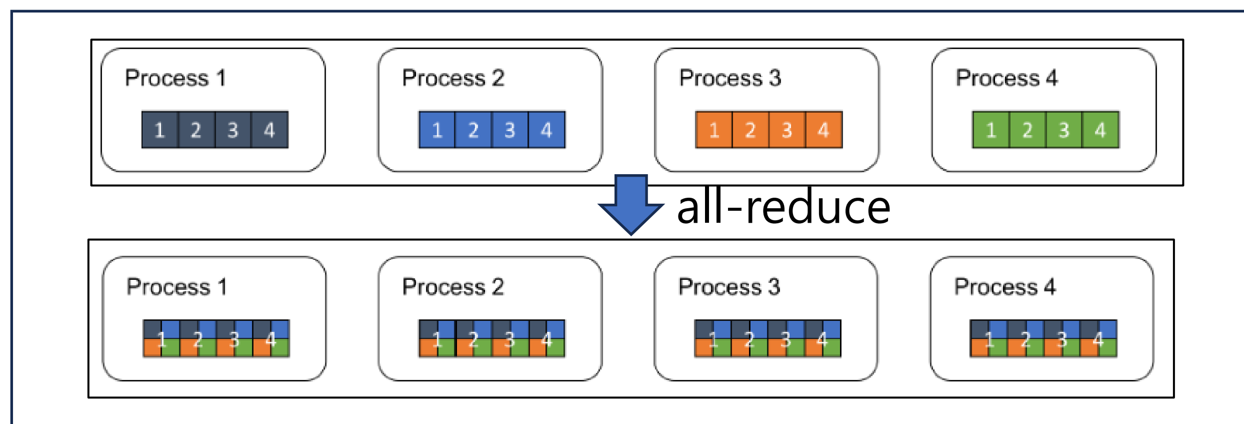


3. 데이터를 GPU끼리 분담 (예를 들어 batch_size=8, WORLD_SIZE=4 인 경우 한 GPU당 batch_size=2)

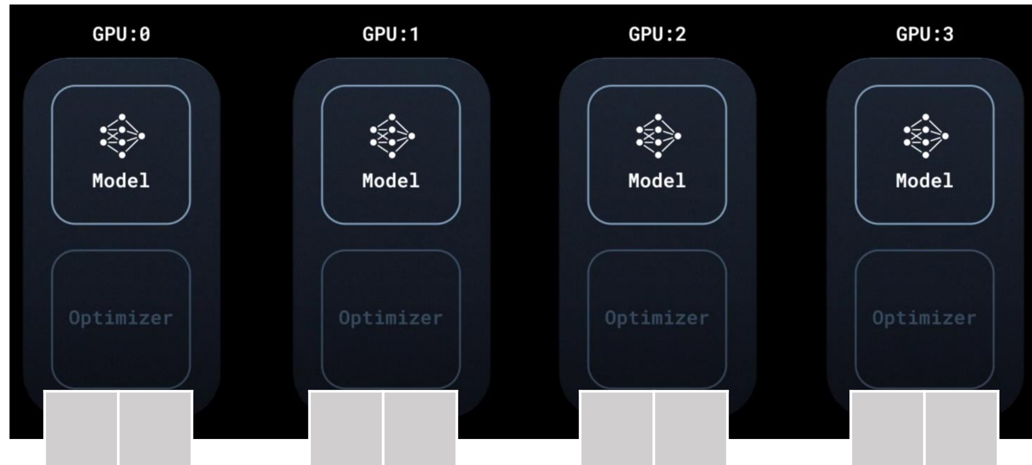
4. 각 GPU가 할당받은 microbatch에 대해
전체 forward와 backward를 수행

5. 모든 parameter의 gradient를 All-Reduce

6. 각 GPU가 알아서 각자 optimizer step



1. Data Parallelism (DP)



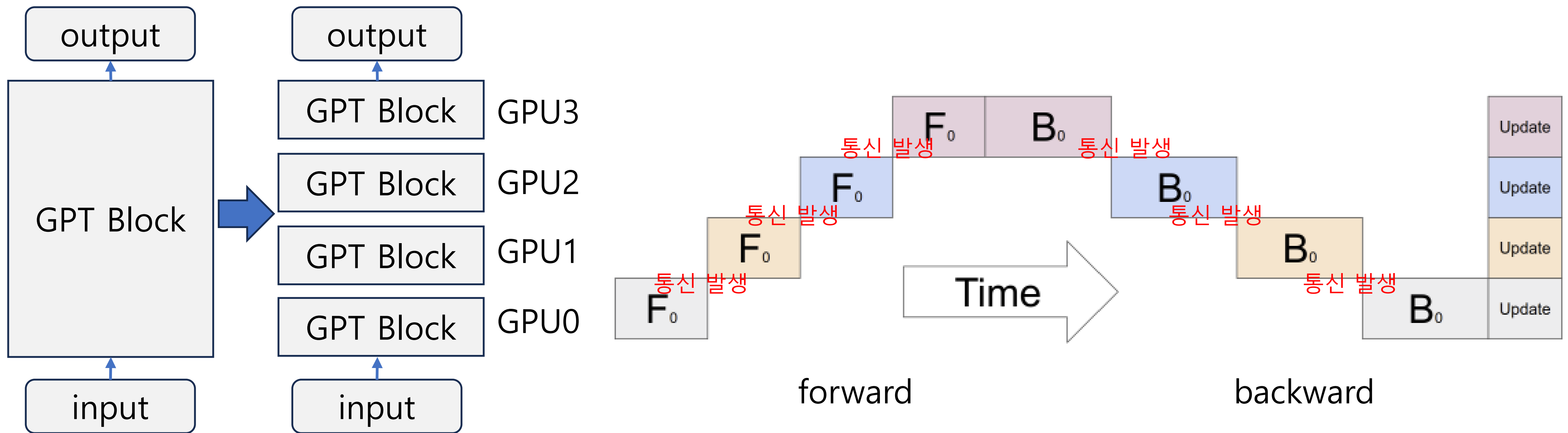
요약

- 메모리 매우 비효율적
  - (model, optimizer state, gradient가 전부 replicate됨)
- GPU간 통신이 많이 필요하지 않아 속도가 가장 빠름
- 구현이 쉽다

2. Pipeline Parallelism (PP)

2. Pipeline Parallelism (PP)

모델이 너무 크다 → 모델을 잘라서 GPU에 각각 나누자!



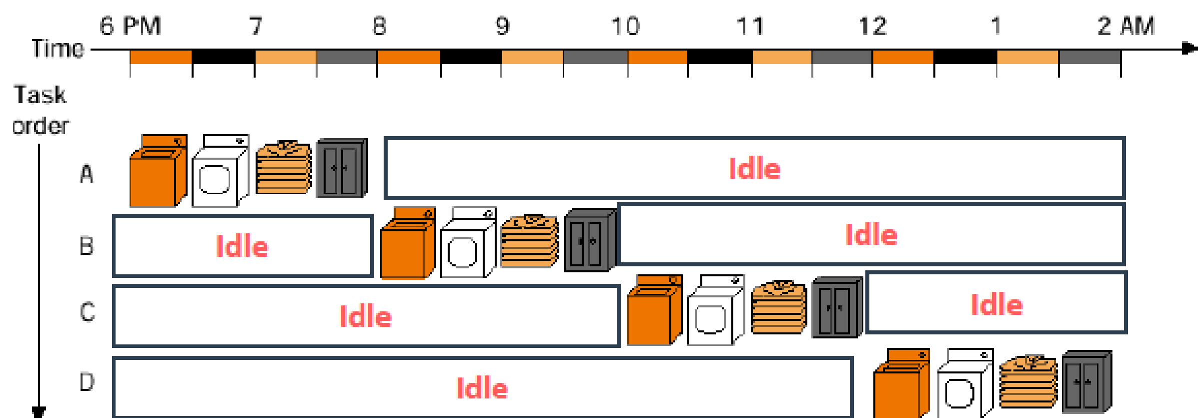
※ 이 방법을 HF에서는 Naïve Model Parallelism이라고 함

2. Pipeline Parallelism (PP)

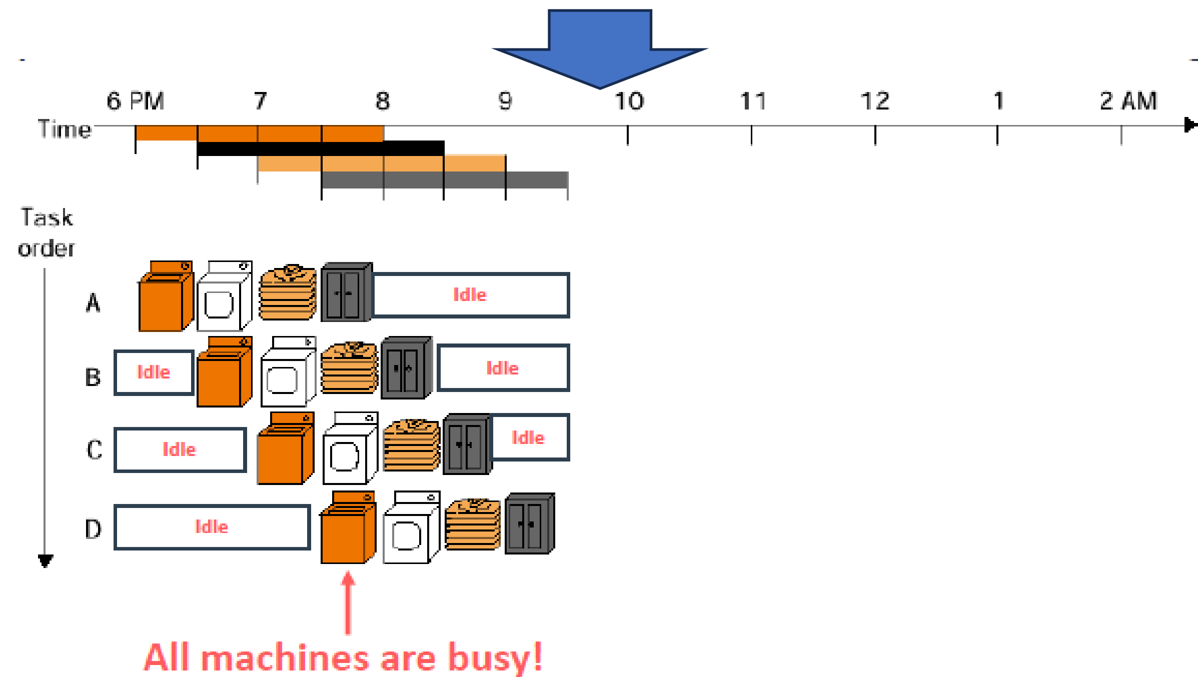
더 최적화할 수 있다!

세탁기-건조기-개기-걸기 의 빨래 4단계 작업
빨래의 batch size=4뭉치

Normal flow
Elapsed time = 16H



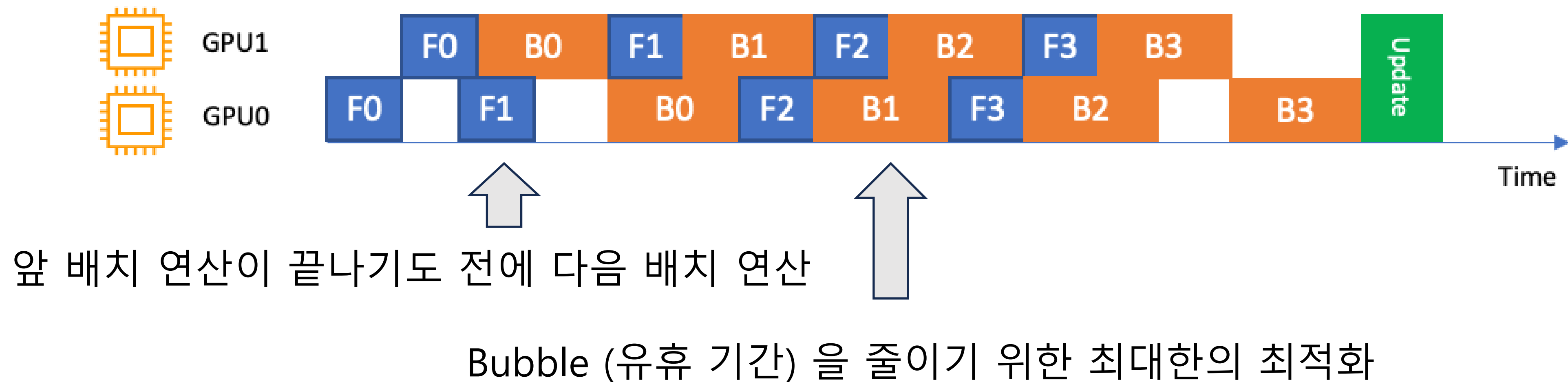
Pipeline Parallelism
Elapsed time = 7H



2. Pipeline Parallelism (PP)

더 더 최적화할 수 있다!

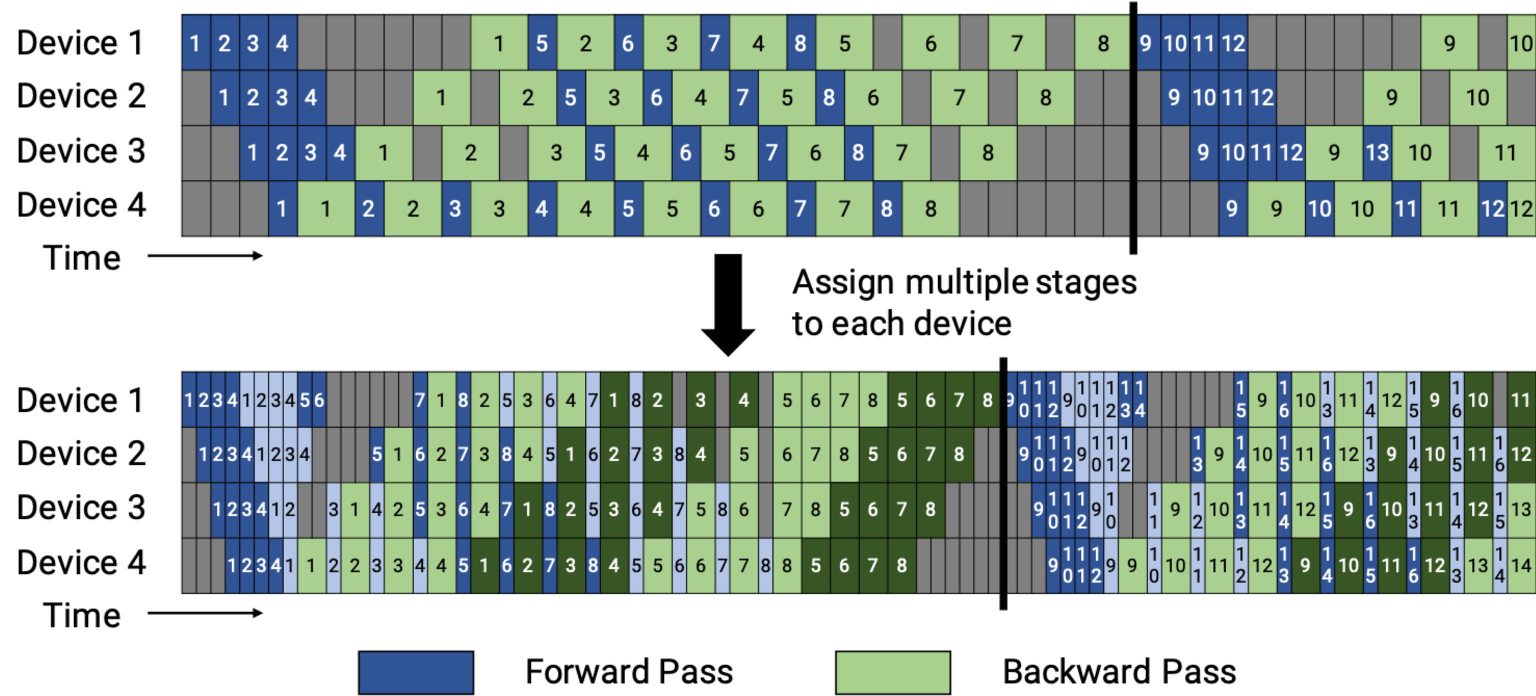
Interleaved Pipeline



Bubble (유휴 기간) 을 줄이기 위한 최대한의 최적화

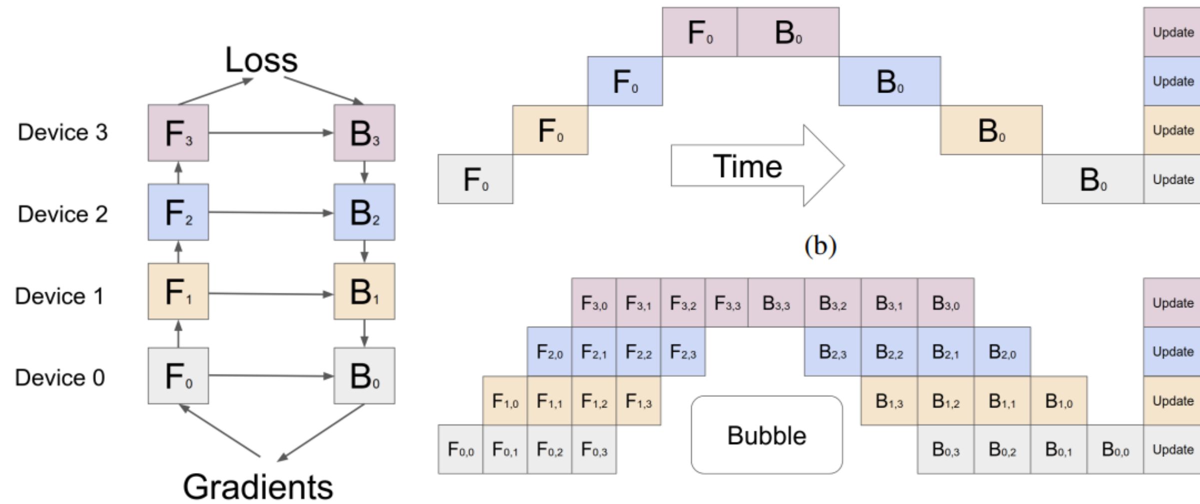
2. Pipeline Parallelism (PP)

더 더 더 최적화할 수 있다!



- Minibatch를 Microbatch로 나눔
- 그걸 중간중간에 삽입
- DeepSpeed에서는 구현되어있지 않음

2. Pipeline Parallelism (PP)



요약

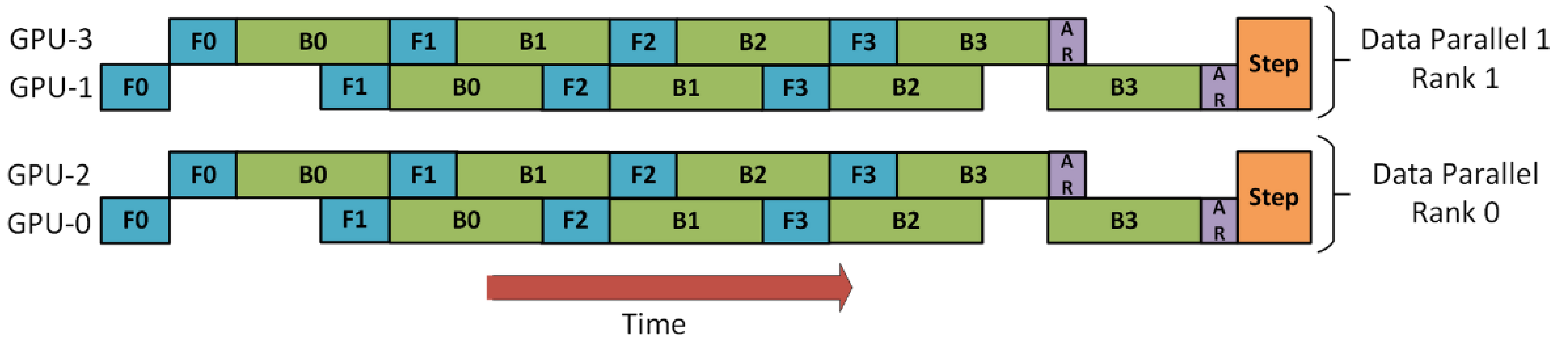
- 모델 구조가 제한됨
  - 모델 레이어가 전부 Sequential해야만 함
  - 모델 input이 항상 텐서가 되어야만 함
  - T5같이 인코더에서 조건부 루틴이 있을 경우 처리하기 힘듦

- 비교적 GPU간 통신이 적어서 빠름

- 메모리 절약 가능

DP + PP

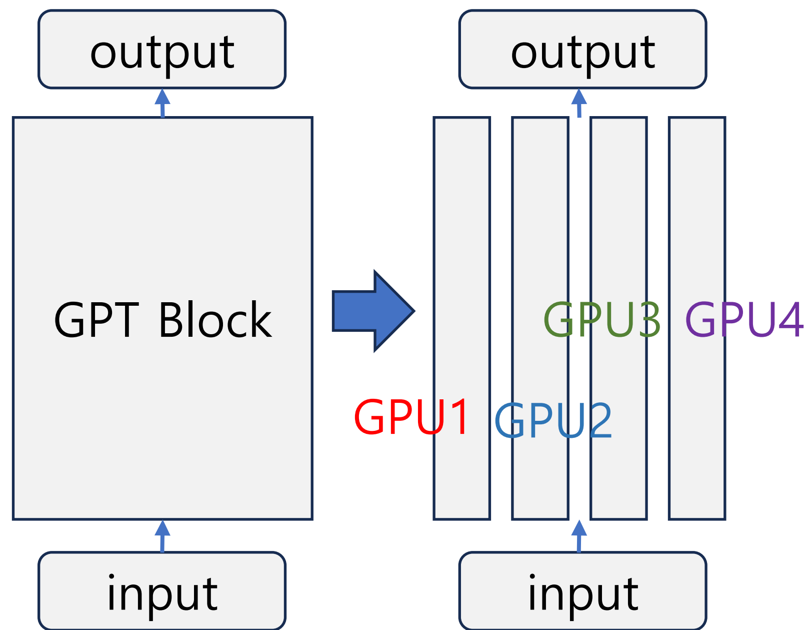
Hybrid solution: DP와 PP를 섞을 수 있음



3. Tensor Parallelism (TP)

3. Tensor Parallelism (TP)

모델을 세로로 자름



그 결과, 레이어 하나를 지날 때마다
매우 많은 통신 발생

3. Tensor Parallelism (TP)

- 행렬곱 연산만 있는 상황 가정

$$\begin{bmatrix} u & v & w \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} =$$

W2 W1 Features

3. Tensor Parallelism (TP)

- 행렬곱 연산만 있는 상황 가정

먼저 연산

$$\begin{bmatrix} u & v & w \\ . & . & . \\ . & . & . \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = \begin{bmatrix} u & v & w \\ . & . & . \\ . & . & . \end{bmatrix} \begin{bmatrix} 1a+2b+3c & \cdot & \cdot \\ 4a+5b+6c & \cdot & \cdot \\ 7a+8b+9c & \cdot & \cdot \end{bmatrix}$$

GPU1 GPU2 GPU3 GPU1 GPU2 GPU3

한번 연산할 때마다 all-reduce 필요

중간에 계산된 값들: **activation**

$$= \begin{bmatrix} u(1a+2b+3c)+ v(\dots)+ w(\dots) & \cdot & \cdot \\ . & \cdot & \cdot \\ . & \cdot & \cdot \end{bmatrix}$$

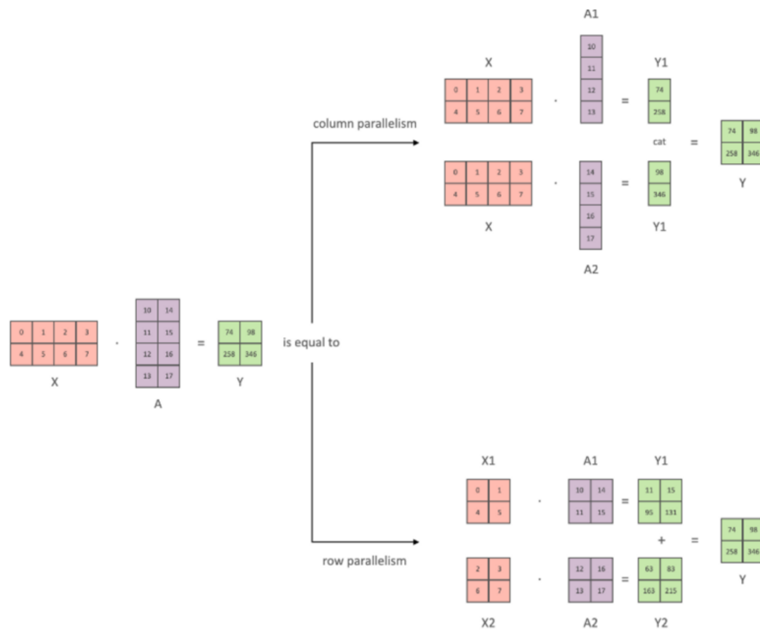
얘를 대상으로 backward

Backward 연산할 때도
'1a+2b+3c' 에는 gradient 'u'가 저장
'4a+5b+6c' 에는 gradient 'v'가 저장
'7a+8b+9c' 에는 gradient 'w'가 저장

되므로 backward 시에도
한 레이어마다 all-reduce가 필요

3. Tensor Parallelism (TP)

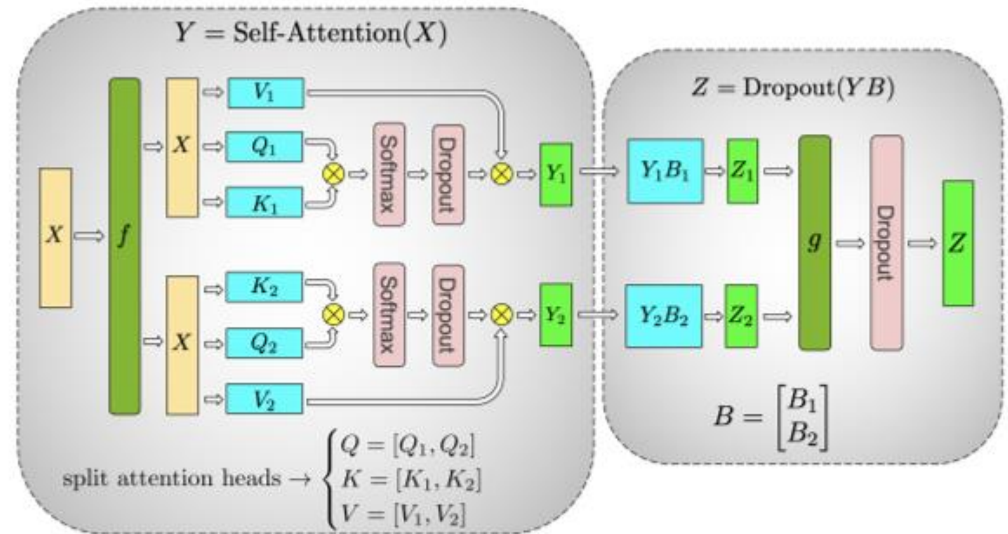
행렬을 세로 말고 가로로 자를 수도 있음



MLP에서는 TP가 행렬을 자르지만

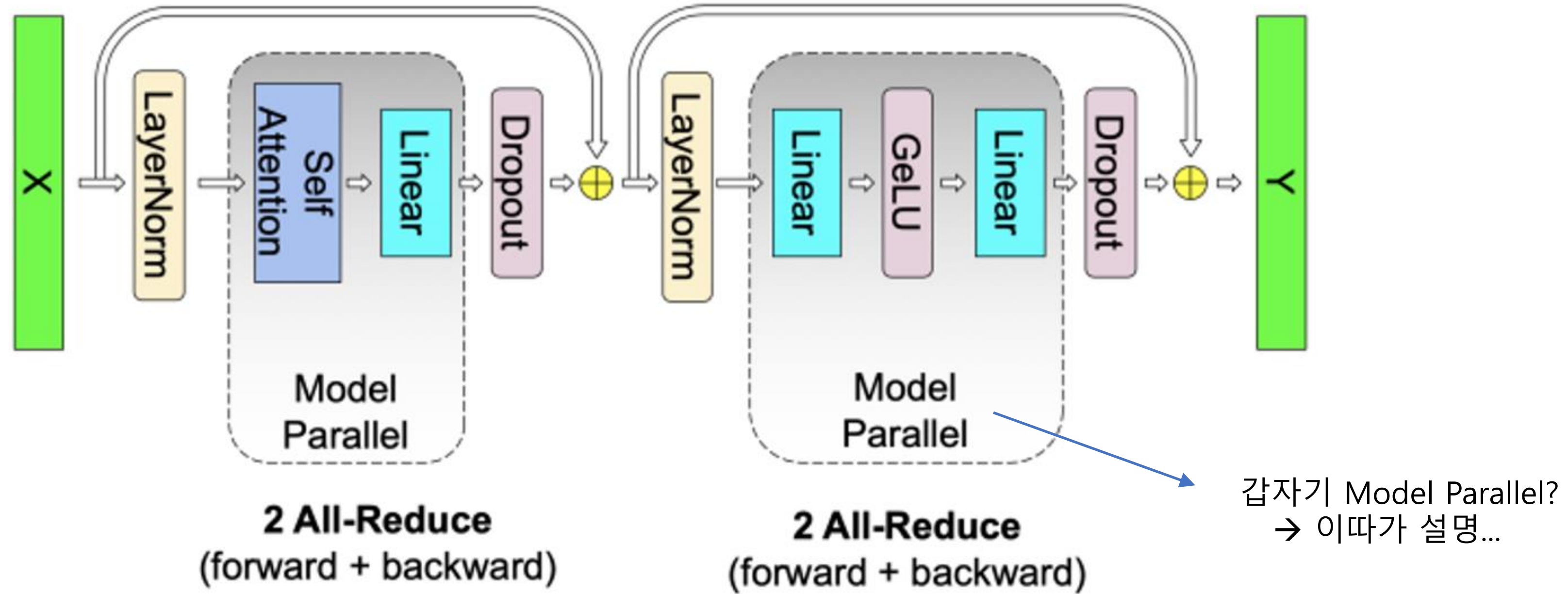
실제 Transformer block에서는 행렬을 자르는 것이 아니라
Multi-head에 대해 Parallel하게 동작함

즉, 각 head가 GPU에 분산됨
-> num_gpu는 num_head의 약수로 설정하면 효율적



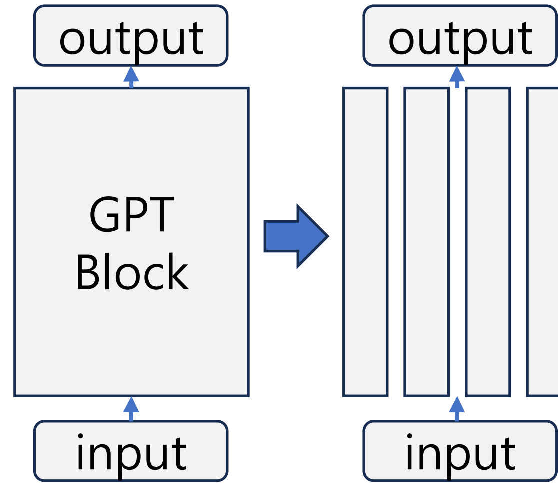
(b) Self-Attention

3. Tensor Parallelism (TP)



하나의 Transformer layer는
GPU간 All-reduce 연산을
4번 발생시킨다.

3. Tensor Parallelism (TP)



특징

- GPU간 통신이 매우 많이 일어남
- 속도는 PP보다 느림
- DP에 비해 메모리를 많이 줄일 수 있음

4. Model Parallelism (MP)

4. Model Parallelism (MP)

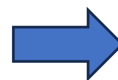
- 용어의 혼용
- 심지어 DeepSpeed ZeRO 논문에서도 혼용함

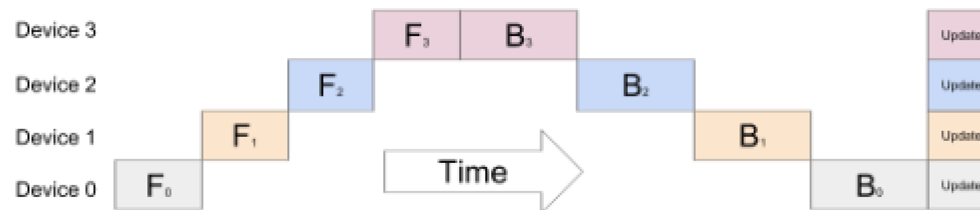
- Model Parallelism은
  - TP를 가리킬 때도,
  - PP를 가리킬 때도,
  - 둘다 가리킬 때도 있음

결론

{ PP, TP } ⊂ Model Parallelism

HF docs에서는
Naïve Model Parallelism이라고 하여,
이 알고리즘을 가리킴





5. ZeRO Data Parallelism (ZeRO-DP)

5. ZeRO-DP (Zero Redundancy Optimizer)

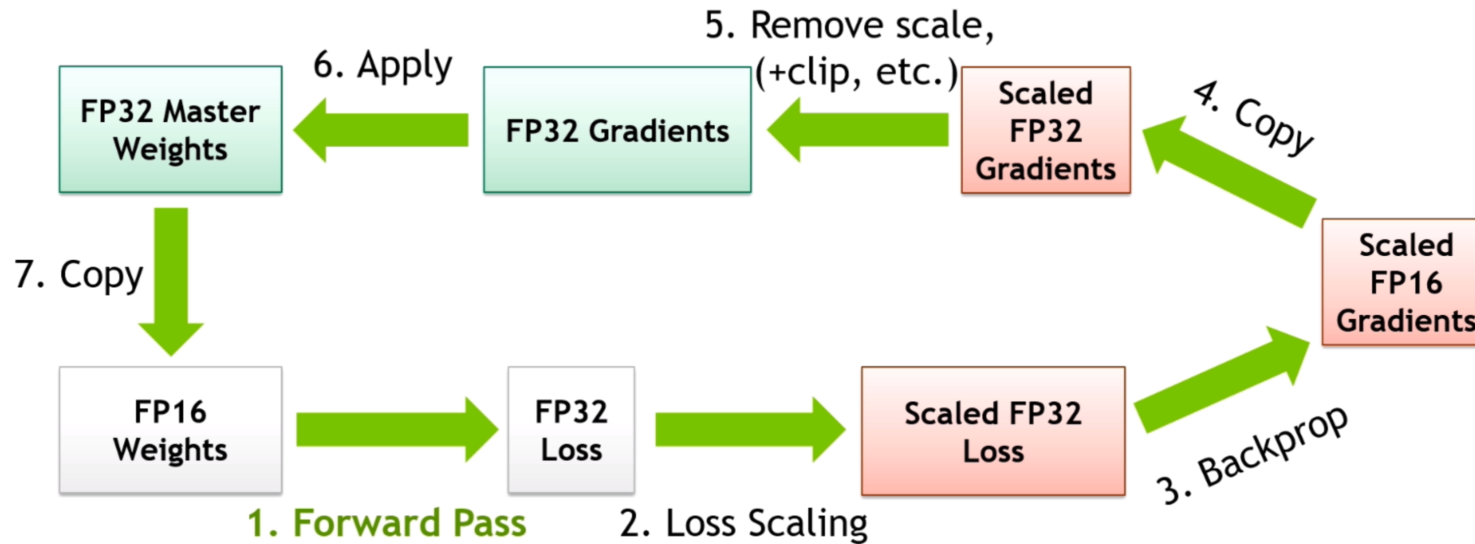
특징

- DataParallel이지만 메모리를 줄일 수 있도록 설계 **(어떻게?)**
- MP와 함께 사용 가능
- PP처럼 모델이 수정될 필요도 없음
- Sharded DDP 라고도 불림
- FP16 Training이 기본
- 3개의 Stage로 나뉨
  - Stage 1: optimizer state partitioning (이라고 하지만 사실은 FP16 parameter도 partitioning함)
  - Stage 2: gradient partitioning
  - Stage 3: parameter partitioning

번외: Mixed Precision Training

번외: Mixed Precision Training

"FP32와 FP16(또는 BF16, TF32) 포맷을 함께 사용하는 것"



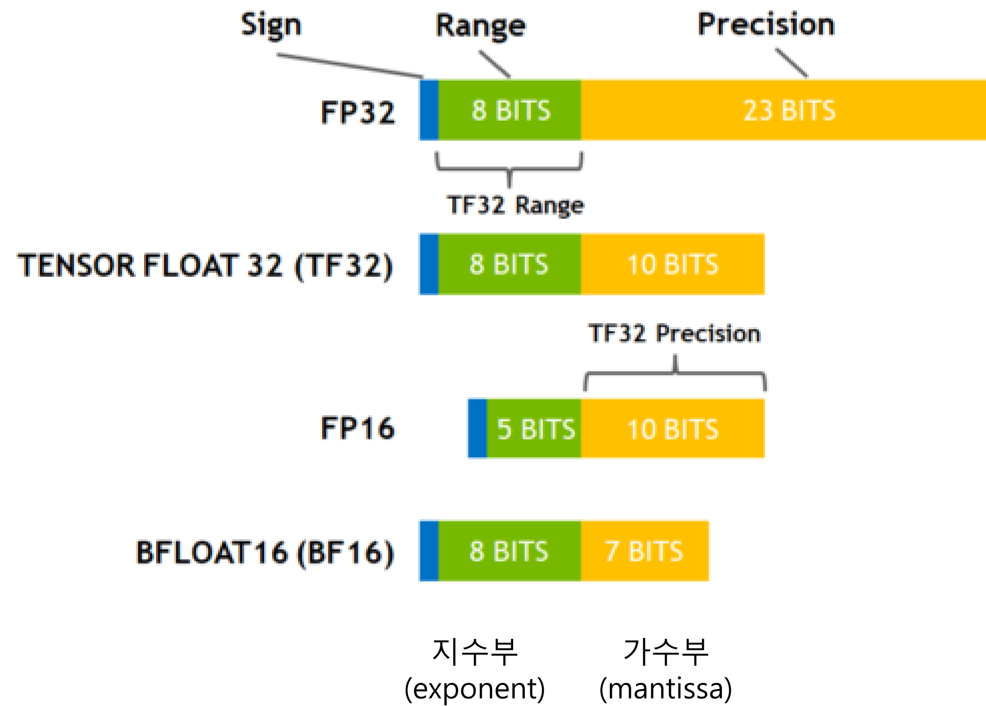
(이걸 이해해야 ZeRO DP를 이해 가능!)

번외: Mixed Precision Training

부동소수점 표현

NVIDIA Ampere 아키텍쳐 이상에서 지원

즉, 6, 7, 8번 서버에서는 사용 불가

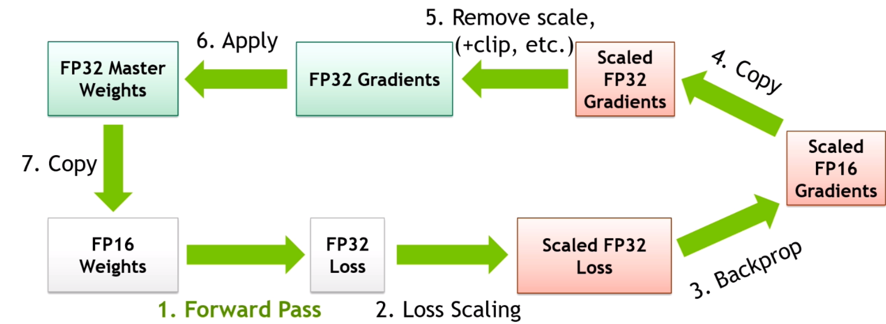


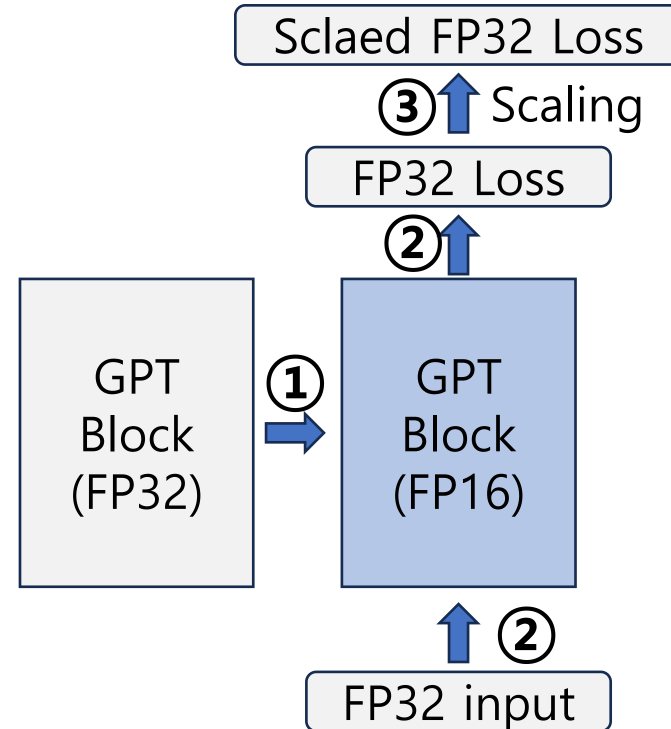
번외: Mixed Precision Training

1. FP32 모델을 통째로 FP16으로 복사

2. FP32 input을 FP16 model에 넣으면 FP32 Loss가 나옴

3. Loss Scaling





번외: Mixed Precision Training

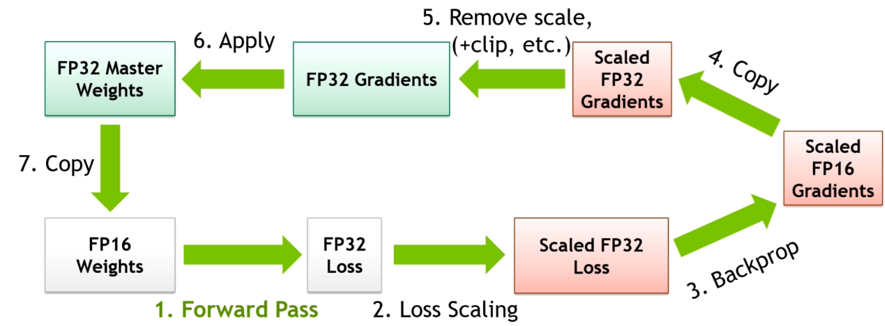
4. Backpropagate 결과 FP16 gradient가 나옴

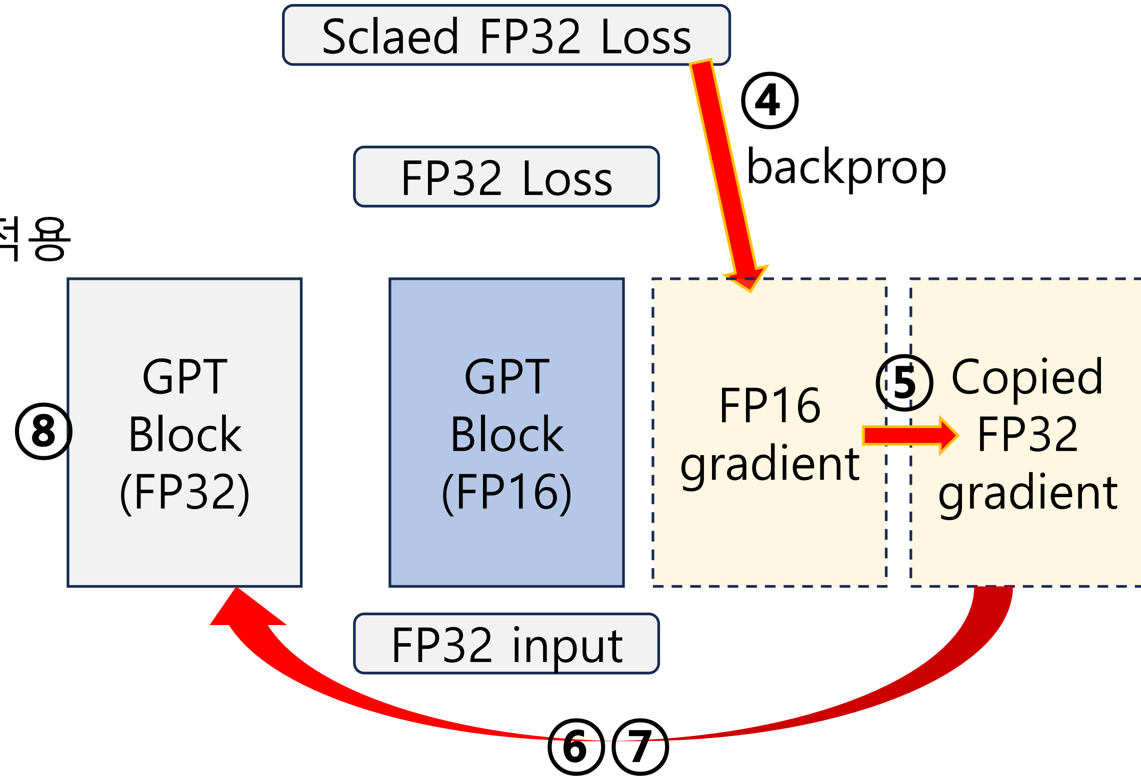
5. FP16 gradient를 copy해서 FP32 gradient로 만듦

6. 생성된 FP32 gradient에 Remove scale (+Clipping 등)

7. 최종 FP32 gradient를 맨 처음의 FP32 Master weight에 적용

8. Master weight에 Optimizer step





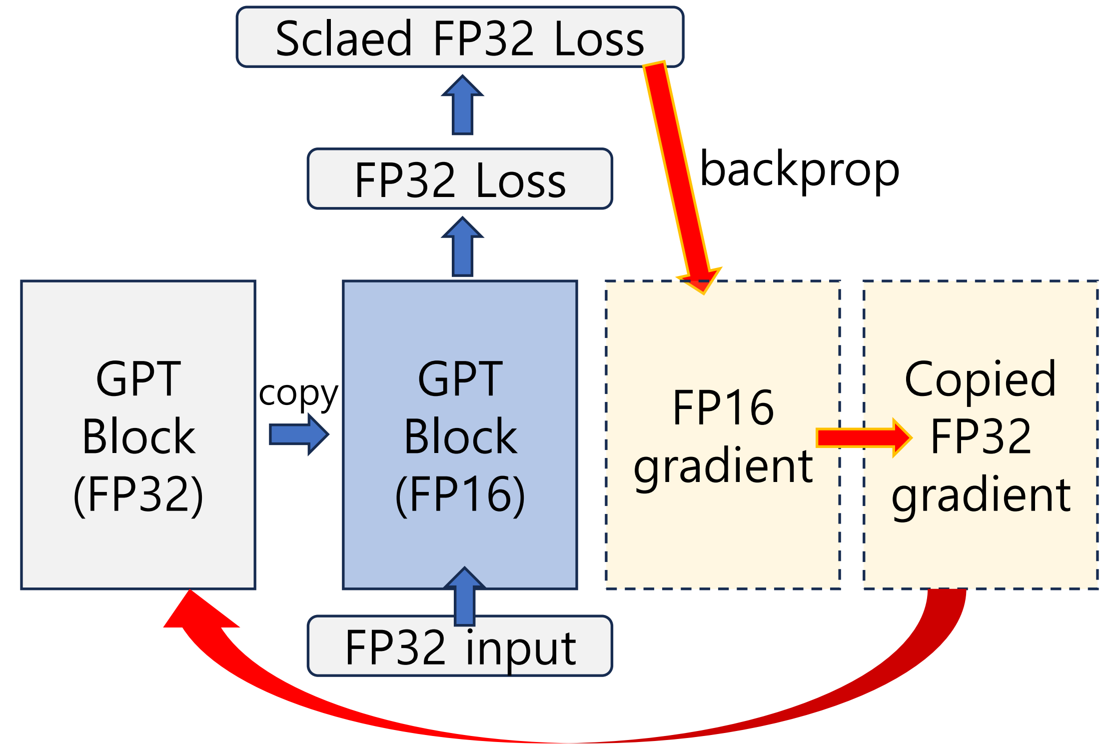
5. ZeRO-DP (Zero Redundancy Optimizer)

FP16 Training에 필요한 parameter 당 메모리

(Parameter의 개수) = x 라고 하면...

FP32 model weight : **4x**
FP32 optimizer status: **4x + 4x**

FP16 model weight : **2x**
FP16 gradient: **2x**

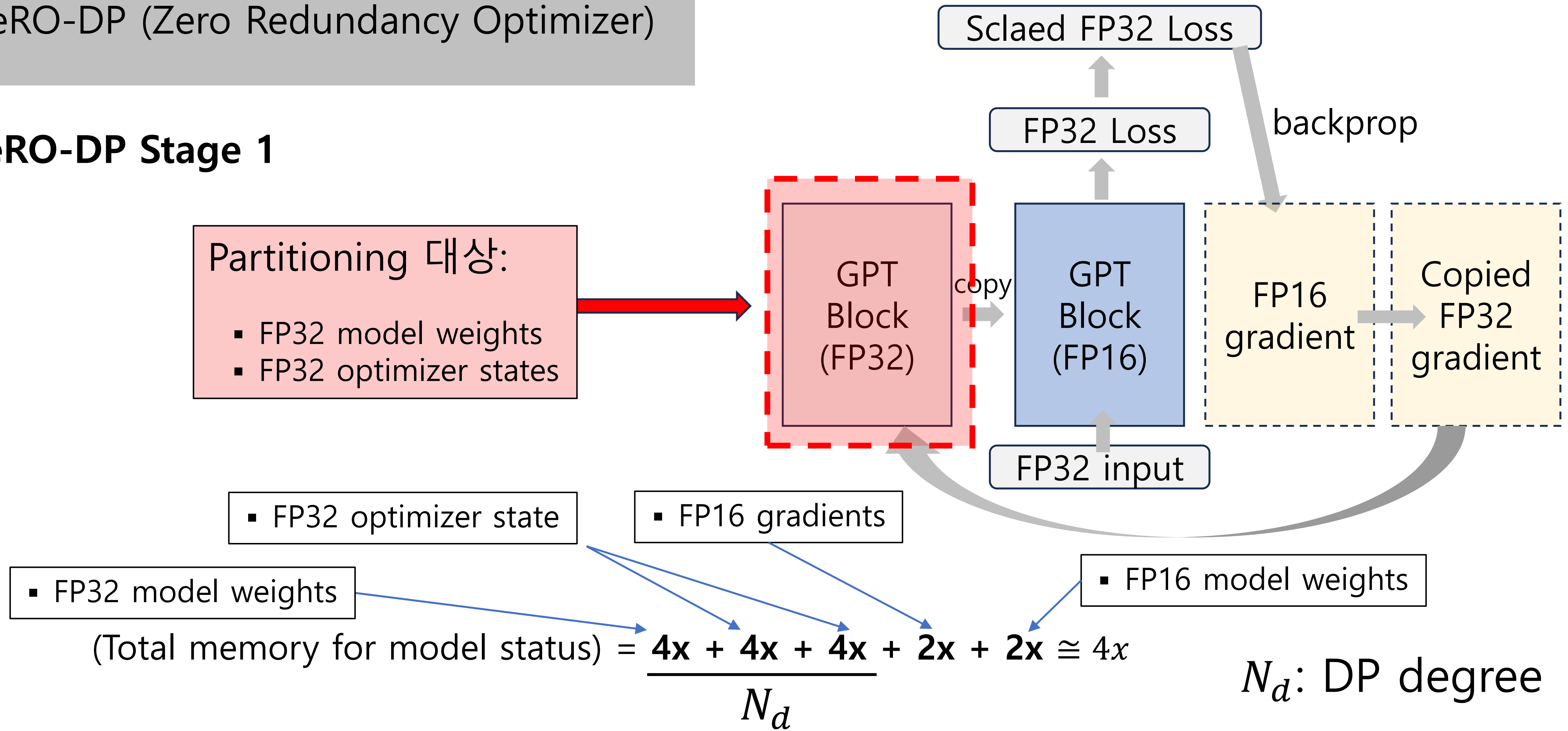


(Total memory for model status) = **4x + 4x + 4x + 2x + 2x** = 16x

그러면 FP16 Training은 FP32에 비교해서 메모리 이득이 없네요?
→ model state 만 보면 FP16이 오히려 메모리를 더 많이 쓴다.
하지만 forward 과정에서 activation이 FP16으로 저장되므로
batch size를 늘릴수록 FP16이 메모리를 절약할 수 있다.

왜 FP32 gradient의 메모리는 고려 안하나요?
→ peak 메모리를 고려했기 때문이다.
FP32 gradient가 쓰일 때면 다른 메모리들이 반환된다.

5. ZeRO-DP (Zero Redundancy Optimizer)

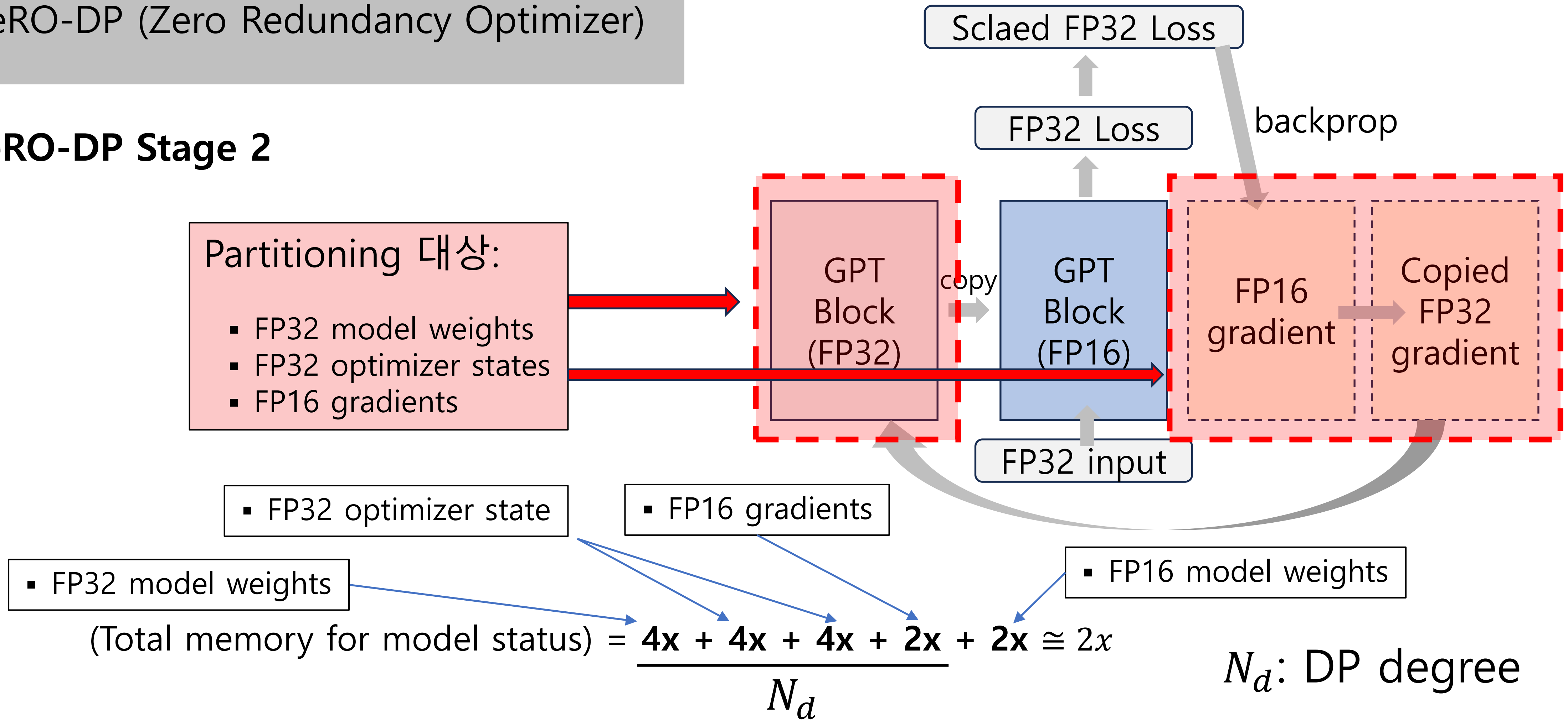


논문의 표현: memory 4배 절약.

$\because TotMem \rightarrow 4x \text{ as } N_d \rightarrow \infty$

5. ZeRO-DP (Zero Redundancy Optimizer)

ZeRO-DP Stage 2



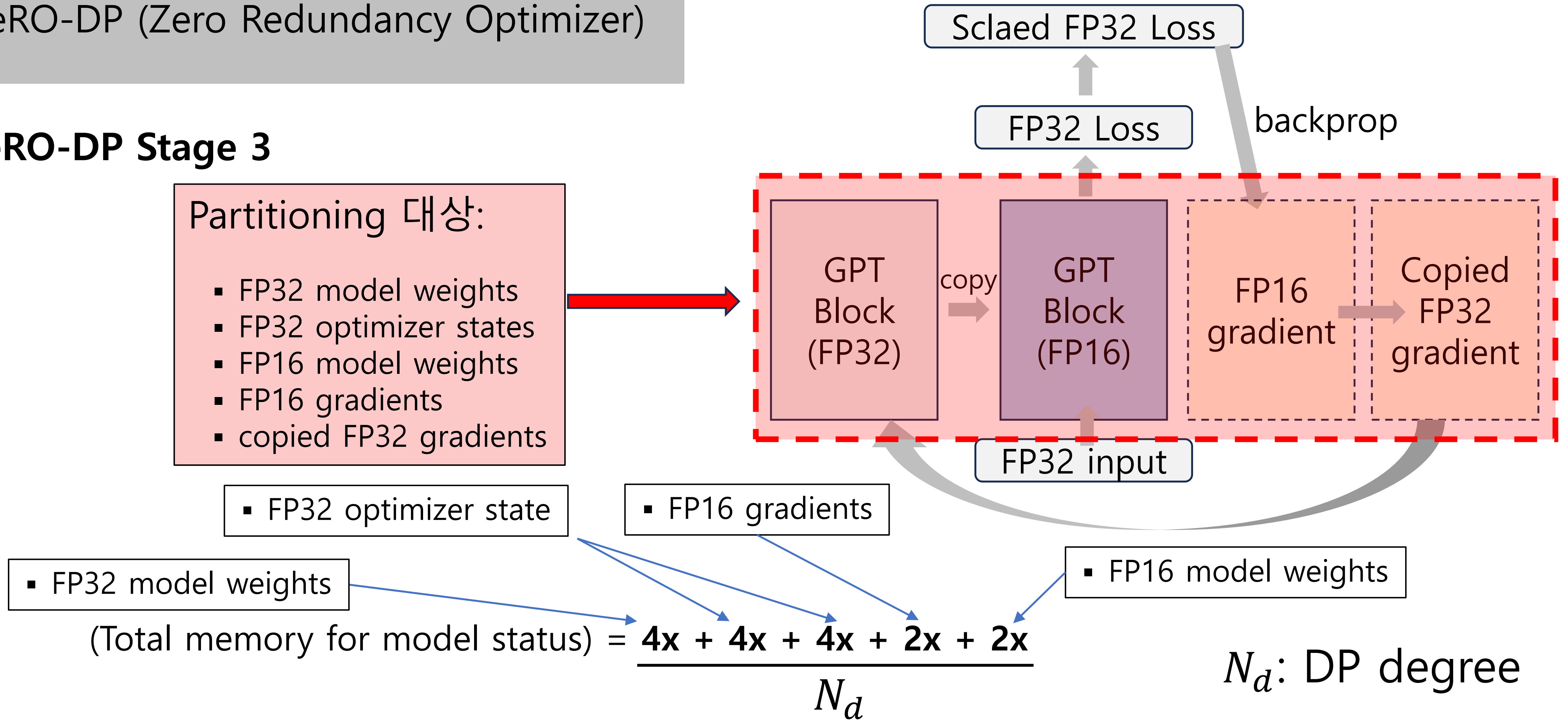
$$(\text{Total memory for model status}) = \frac{\mathbf{4x + 4x + 4x + 2x}}{N_d} + \mathbf{2x} \cong 2x$$

N_d: DP degree

논문의 표현: memory 8배 절약.

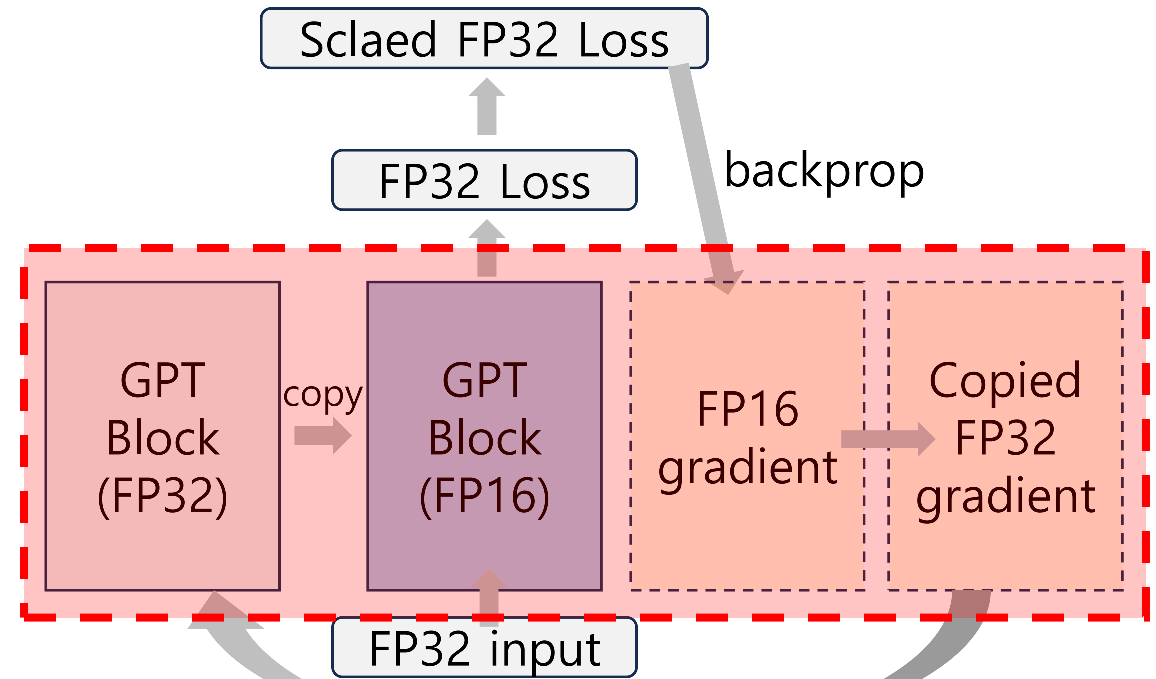
$\because TotMem \rightarrow 2x \ as \ N_d \rightarrow \infty$

5. ZeRO-DP (Zero Redundancy Optimizer)



특징
- FP16 model weights를 TP처럼 partitioning함.
- 그러므로 Stage 2에 비해 GPU간 통신이 많아져서 속도 차이가 꽤 남
- 논문에서는, 최적화 기술 덕분에 속도 차이가 크지 않다고 주장함
- 그러므로 매우 큰 모델이 아니라면 Stage 3 대신 Stage 2도 고려할만 함 (실제로 해보면 20% 정도 차이)

5. ZeRO-DP (Zero Redundancy Optimizer)

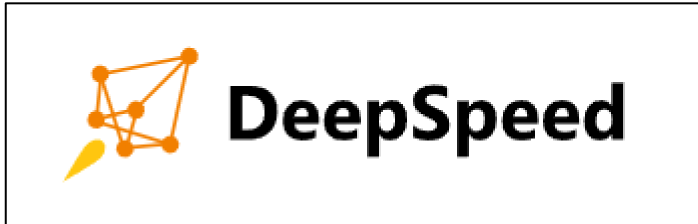


ZeRO-DP

요약

- Mixed Precision이 기본 (FP32도 할 수 있긴 한데 설계 근본이 Mixed Precision)
- Stage1, Stage2, Stage3 순으로 속도를 희생, 메모리를 아낌
- ZeRO-Offload 기능 존재
  - CPU 메모리 또는 NVMe SSD에 os, p를 offload 가능
- MP와 같이 사용 가능하지만 Multi Node가 아닌 경우 ZeRO-DP만 사용해도 충분

6. DeepSpeed VS pytorch FSDP



- ZeRO 논문의 원조
- 업데이트가 활발함
- RLHF 플랫폼에도 적용 가능
- 최근 ZeRO++ 방법론으로 GPU 간 통신이 더 최적화됨
- Alpaca 연구진은 같은 조건에서 DeepSpeed가 더 메모리를 적게 먹었다고 함

FULLYSHARDEDDATAPARALLEL

CLASS torch.distributed.fsdp.FullyShardedDataParallel(*module, process_group=None, sharding_strategy=None, cpu_offload=None, auto_wrap_policy=None, backward_prefetch=BackwardPrefetch.BACKWARD_PRE, mixed_precision=None, ignored_modules=None, param_init_fn=None, device_id=None, sync_module_states=False, forward_prefetch=False, limit_all_gathers=False, use_orig_params=False, ignored_parameters=None*) [SOURCE]

A wrapper for sharding Module parameters across data parallel workers. This is inspired by Xu et al. as well as the ZeRO Stage 3 from DeepSpeed. FullyShardedDataParallel is commonly shortened to FSDP.

- 같은 ZeRO 방법론 사용
- pytorch 네이티브이므로 속도가 더 빠를 수 있다고 홍보
- 디펜던시 이슈에서 비교적 자유로움
- torch.distributed 안에 있으므로 코드가 간결
- 더 자유로운 optimizer, LR scheduler 커스터마이징

3D Parallelism

- 180 billion 모델
- DP는 ZeRO Stage 1 사용
- 3D Parallelism이 필요함을 보여 줌

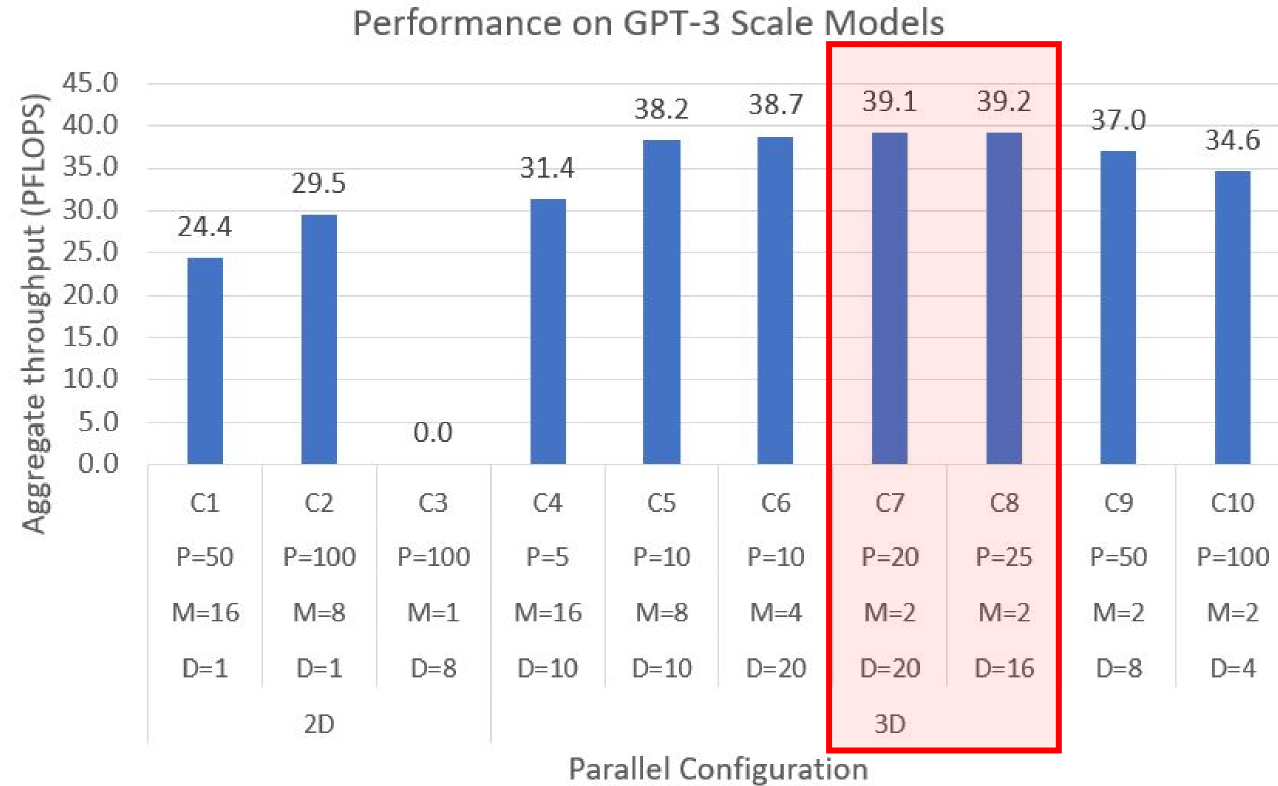


Figure 4: System performance using 800 GPUs to train a GPT-3 scale model with 180 billion parameters using 2D and 3D parallelism. The model has 100 Transformer layers with hidden dimension 12,288 and 96 attention heads. The model is trained with batch size 2,048 and sequence length 2,048. ZeRO-1 is enabled alongside data parallelism. P, M, and D denote the pipeline, model, and data parallel dimensions, respectively.

실무 적용

1. 여기서 말하는 Naïve PP는 무엇인가?

v0.4.1

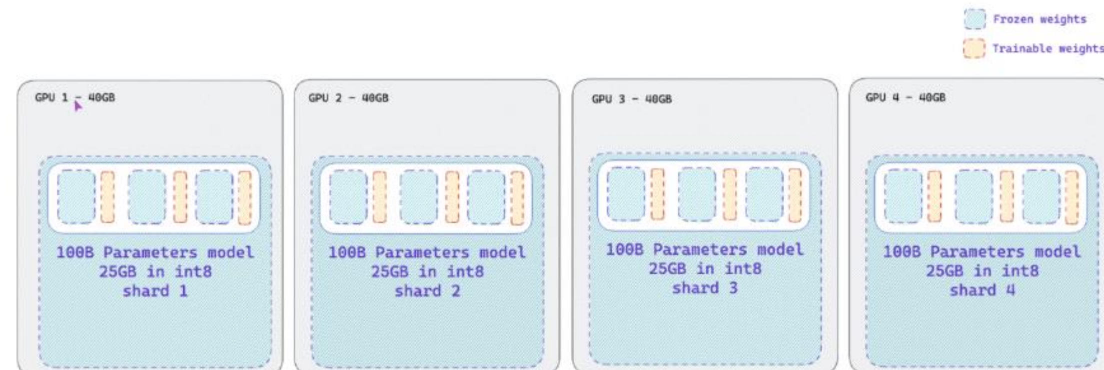
Large models training, Naive Pipeline Parallelism, `peft` Data Parallelism support and distributed training bug fixes

This release includes a set of features and bug fixes to scale up your RLHF experiments for much larger models leveraging `peft` and `bitsandbytes`.

Naive Pipeline Parallelism support

- Let's support naive Pipeline Parallelism by **@younesbelkada** in #210

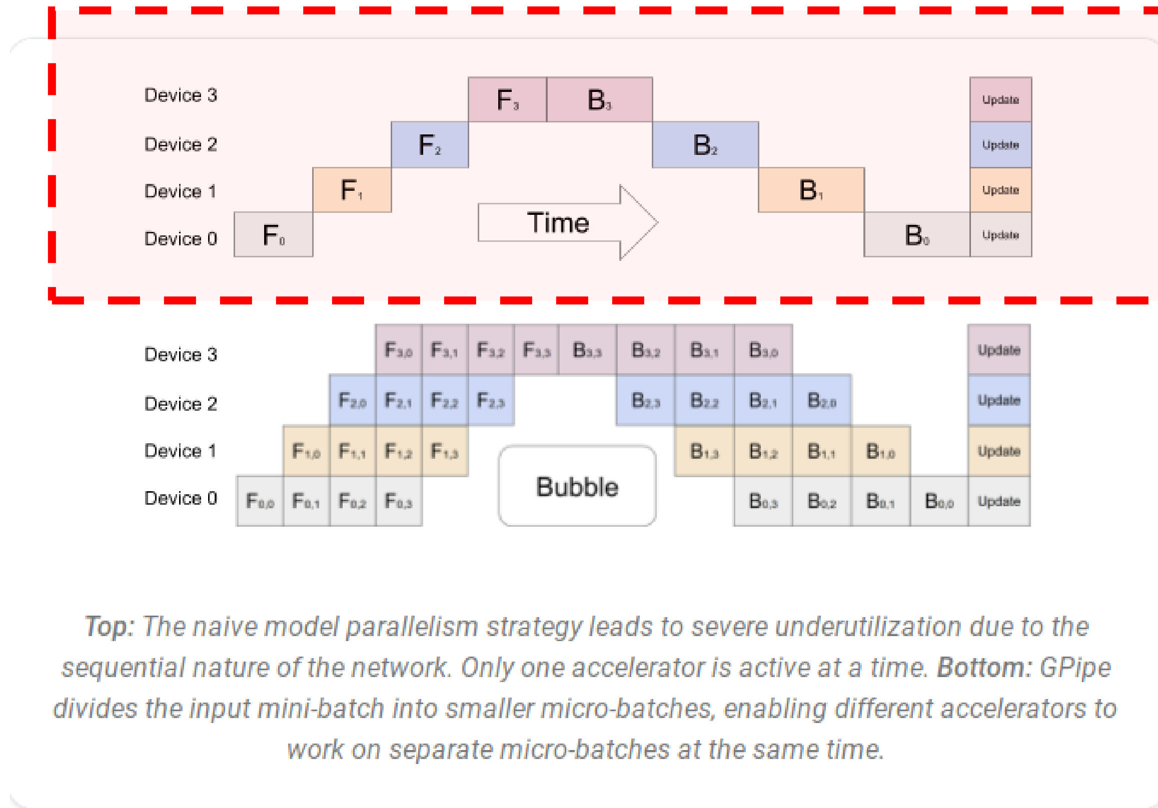
We introduce a new paradigm in `trl`, termed as Naive Pipeline Parallelism, to fit large scale models on your training setup and apply RLHF on them. This feature uses `peft` to train adapters and `bitsandbytes` to reduce the memory foot print of your active model



(출처: trl github)

실무 적용

1. 여기서 말하는 Naïve PP는 무엇인가?



Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. ***Bottom:*** GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.

같은 batch_size라면,

속도:
GPU 1개 > GPU 8개

매우 느려질 것이 예상.

실무 적용

2. DeepSpeed를 쓸 때, 가장 많은 속도를 희생해서 메모리를 조금 줄이는 단계는?

① DDP -> ZeRO Stage 1

② ZeRO Stage 1 -> ZeRO Stage 2

③ ZeRO Stage 2 -> ZeRO Stage 3

실무 적용

2. DeepSpeed를 쓸 때, 가장 많은 속도를 희생해서 메모리를 조금 줄이는 단계는?

① DDP -> ZeRO Stage 1

② ZeRO Stage 1 -> ZeRO Stage 2

③ ZeRO Stage 2 -> ZeRO Stage 3

DS-2, DS-3은 메모리, 속도에서 얼마나 차이?

- DS-2, acc=8, batch_size=2 (42.9GiB) (42.7s/it) (총3시간45분)
- DS-2, acc=4, batch_size=4 (45.9GiB) (39.5s/it) (총3시간30분)
- DS-3, acc=8, batch_size=2 (38.8GiB) (49.7s/it) (총4시간23분)
- DS-3, acc=8, batch_size=4 (42.6GiB) (85s/it) (총3시간42분)
- DS-3, acc=8, batch_size=8 (실패)
- DS-3, acc=4, batch_size=8 (실패)
- DS-3, acc=16, batch_size=4 (42.9GiB) (170s/it) (총3시간42분)
- DS-3, acc=2, batch_size=8 (실패)
- DS-3, acc=1, batch_size=8 (실패)

∵ GPU간 통신이 매우 많이 필요해짐

메모리 여유가 있을 경우 Stage 2를 쓰는 것이 효율적

더 읽을거리

1. ZeRO-R
    - Model state (p, os, g) 제외하고도
      Residual memory (activations, temporary buffers, unusable memory) 에 대한 최적화

2. Loss Scaling
    - FP16로 변환 시 비트가 부족해서 매우 작은 값을 표현하지 못하는 문제를 해결

3. Activation Checkpointing
    - HF에서는 gradient checkpointing라고 함. 같은 말임
    - 무지막지한 Activation 메모리를 최적화

4. Partitioned Activation Checkpointing
    - ZeRO에서도 MP-degree만큼 activation을 partitioning 가능

끝