과제 #7

2022-29677 한주희

1 Deepspeed

- ZeRO Stage 1은 optimizer state를 분산시켜 메모리 사용을 줄인다. 따라서 optimizer state는 N개의 GPU에 나누어 저장하고, gradient와 parameter는 모든 GPU가 동일하게 갖고 있다. forward 단계에서 parameter는 각 GPU에 복사되어 사용되므로 통신이 발생하지 않으며, backward 단계에서 gradient는 각 GPU에서 계산되고 모든 GPU 간에 All-Reduce 통신을 사용하여 동기화한다. optimizer 단계에서는 optimizer state가 sharded 방식으로 분배되므로, 각 GPU는 자신이 갖고 있는 shard에 대해서만 업데이트를 수행하고, 이 단계에서는 통신이 필요하지 않다.

  Stage 2는 gradient까지 분산시켜 메모리 사용을 줄인다. 따라서 optimizer state를 N개의 GPU에 나누어 저장하고, gradient를 sharded 방식으로 저장하여 각 GPU가 전체 gradient 중 일부만 저장하도록 한다. parameter는 모든 GPU가 동일하게 갖고 있다. Stage 1과 마찬가지로 forward 단계에서는 parameter는 동일하게 복사되어 사용되므로 통신은 없다. backward 단계에서는 sharded 방식으로 gradient를 나누어 계산하며, 필요한 경우 reduce-Scatter를 통해 통신한다. optimizer 단계에서는 sharded 방식으로 저장된 optimizer state에 대해 각 GPU가 자신의 shard를 업데이트하며 통신이 발생하지 않는다.

  Stage 3는 parameter까지 분산시켜 메모리 사용을 줄인다. optimizer, gradient, parameter까지 N개의 GPU에 나누어 저장한다. forward 단계에서 필요한 parameter shard를 각 GPU가 로컬에 유지하거나 Broadcast 통신으로 불러온다. backward 단계에서는 gradient를 로컬 shard에서 계산되며, 다른 shard가 필요할 경우 reduce-Scatter 통신을 사용한다. optimizer 단계에서는 optimizer state와 parameter shard를 GPU마다 독립적으로 업데이트하며, 필요한 경우 all-gather를 통해 동기화한다.

Offload는 optimizer state를 CPU memory로 offload한다. gradient와 parameter는 GPU에 저장되고 stage에 따라 sharded 방식으로 저장될 수 있다. forward 단계에서는 parameter가 GPU에서 관리되며, 필요 시 Broadcast 통신이 발생합니다. backward 단계에서는 gradient를 GPU에서 계산하며 stage 2와 stage 3에서와 동일하게 reduce-Scatter 통신을 사용한다. optimizer 단계에서는 CPU에 저장된 optimizer state를 GPU와 통신하여 업데이트하며, 주로 Point-to-Point 방식의 통신이 사용됩니다.

- `train_batch_size`는 전체 글로벌 배치 사이즈이다.
  `train_micro_batch_size_per_gpu`는 각 GPU가 처리할 microbatch 크기를 저장한다.
  `zero_optimization`에서 stage를 통해 단계를 설정한다. `reduce_scatter` 통신을 true로 설정하여 backward 단계에서 gradient 동기화에 reduce-scatter 통신을 사용하도록 설정한다. `allgather_partitions`은 필요한 optimizer state를 all-gather로 통신해 shard를 GPU간 결합하도록 설정한다. `optimizer`를 통해 학습에 사용할 `adam_optimizer`를 지정한다.

  `zero_optimization`에서 stage 2를 활성화한다. `allgather_partitions`을 true로 설정하여 sharded optimizer state와 gradient를 필요한 시점에 all-gather로 결합하도록 설정한다. `allgather_buck_size`로 all-gather 통신에서 한번에 전송하는 데이터 크기를 설정한다. `reduce_scatter`를 true로 설정하여 backward 단계에서 gradient 동기화에 reduce-scatter 통신을 사용하도록 설정한다. `overlap_comm`을 true로 설정하여 통신과 계산을 동시에 수행할 수 있도록 설정한다.

  `zero_optimization`에서 stage 3를 활성화한다. `contiguous_graindents`를 true로 설정하여 gradient 메모리를 연속적으로 저장한다. `reduce_bucket_size`는 reduce-scatter 통신 시 한번에 전송하는 데이터 크기를 정의한다. `sub_group_size`는 parameter를 서브 그룹 단위로 나누는 크기를 정의한다.

FP16을 활성화하기 위해서 `fp16` argument를 추가해야 한다. enable을 통해 fp16 연산을 활성화한다. `auto_cast`는 fp16 연산에서 pytorch AMP 기능을 사용할지 여부를 설정한다. loss scale를 통해 loss scaling을 설정한다. `initial_scale_power`는 loss scaling 초기값을 설정한다. `loss_scale_window`는 loss scaling 값을 조정하는 배치 수를 설정한다. `hysteresis`는 loss scaling 값을 줄이는 hysteresis를 설정한다. `consecutive_hysteresis`는 hysteresis를 연속적으로 평가할지 설정한다. `min_loss_sclale`는 loss scaling 값의 최소 허용치를 정의한다.
CPU offloading을 활성화하기 위해 `offload_optimizer` argument를 추가해야 한다. `device`로 optimizer state를 offload할 device를 지정한다. `nvme_path`는 nvme 경로이다. pin_memory를 통해 pinned memory를 사용할지 여부를 설정한다. `ratio`는 optimizer state를 offload할 비율을 설정한다. `buffer_count`로 비동기 데이터 전송에 사용할 버퍼 개수를 설정한다. `fast_init`으로 optimizer state를 초기화하는 방식을 설정한다.

-

Stage	Avg Forward Mem Usage (MB)	Avg Backward Mem Usage (MB)	Avg Optimizer Mem Usage (MB)	Steps Executed	문제 발생 여부
Stage 1	9610.41	-9610.32	0	130	None
Stage 2	9610.22	-9610.25	0	40	일무 메모리 효율성 개선
Stage 2 + CPU Offload	9609.57	-9609.53	0	30	CPU로의 offload로 일부 메모리 분산
Stage 3	11998.42	-12476.86	-250.82	40	메모리 압박으로 인해

					PyTorch 캐시 플러시 경고가 빈번하게 발생
Stage 3 + CPU offload	12270.46	-12476.59	-250.45	30	CPU로의 offload로 일부 메모리 분산
Stage 3 + FP16	6455.49	-6329.29	-125.2	90	FP16 활성화로 메모리 사용량 크게 감소

- Stage 1: 정상적으로 실행됨.
- Stage 2: Backward 단계에서 메모리 사용량이 약간 증가한 것으로 보임. Gradient를 각 GPU에 분산 저장하면서 allgather 및 reduce-scatter와 같은 통신 연산이 발생하여, 일시적으로 메모리 사용량이 증가한 것으로 추정됨. 이러한 통신 연산으로 인해 GPU 메모리 부족이 발생해 실행 중단된 것으로 보임.
- Stage 2 + Offload: CPU로 일부 Optimizer State를 offload하면서 GPU 메모리 사용량은 감소했으나, 동기화 지연으로 인해 실행 중단된 것으로 보임. 특히 PCIe 대역폭 한계로 인해 데이터 전송 지연이 발생했을 가능성이 있음.
- Stage 3: 다수의 cache flush 경고가 발생함.메모리 압박으로 PyTorch의 캐시가 자주 플러시되었으며, 이로 인해 성능 저하와 함께 실행 중단된 것으로 보임. Stage 3의 Parameter Partitioning과 Optimizer State 관리가 추가적인 메모리 소비와 연산 부담을 유발한 것이 원인으로 추정됨.
- Stage 3 + Offload: CPU로 일부 Parameter와 Optimizer State를 offload하여 GPU 메모리 사용량은 줄었으나, Backward Pass와 Optimizer 연산 부담이 증가함.

Parameter와 Gradient 동기화 과정에서 CPU와 GPU 간 데이터 전송 속도가 GPU 연산 속도를 따라가지 못하며 bottleneck이 발생, 실행 중단된 것으로 보임.

- Stage 3 + FP16: FP16 사용으로 메모리 사용량이 대폭 감소하였으나 값의 표현 범위가 제한적이므로 Loss scale overflow 발생하여 실행 중단된 것으로 보임. Overflow가 발생할 때마다 Loss Scale을 낮추는 연산이 추가되며 학습 시간이 길어지고, 결국 실행이 중단된 것으로 보임.

-

Stage	Average Forward (s)	Average Backward (s)	Average Optimizer (s)	Average Total (s)
Stage 1	0.3721	0.8553	0.0002	1.2275
Stage 2	0.4270	4.5714	0.0001	4.9985
Stage 2 + Offload	0.4158	4.2478	0.0001	4.6637
Stage 3	1.8912	3.0930	0.1963	5.1805
Stage + Offload	2.6211	4.1413	1.1885	7.9508
Stage 3 + FP16	1.1693	1.2814	0.2075	2.6581