Hw7

2024-27759 인문대학 언어학과 장효형

• (40점) DeepSpeed의 ZeRO Stage 1, 2, 3, Offload에 대해 설명하라. 각각의 Stage에서 optimizer state, gradient, parameter 값들이 forward, backward, optimizer step 단계에서 어떻게 관리되는지, 그리고 N 개의 GPU가 동일하게 갖고 있는지 또는 나누어 갖고 있는지를 기술해야 한다. 또한, forward, backward, optimizer step 단계에서 어떤 데이터(optimizer state, gradient, parameter)가 통신되고, 이 통신이 어떤 방식(예: collective communication 종류)으로 이루어지는지를 설명하라.

1) Stage 1 optimizer state만 나누어 가짐. 그러므로 forward할 때 동일한 parameter로 트레이닝되고, backward할 때 동일한 gradient를 계산 및 보관하고, step에서 분산된 optimizer state를 allgather communcation로 모아 업데이트를 시행.
2) Stage 2 optimizer state, gradient만 나누어 가짐. 그러므로 forward할 때 동일한 parameter로 트레이닝 되고, backward할 때 각 GPU가 서로 다른 gradient를 게산하고 동일한 값을 퍼뜨림(reduce-scatter), 분산된 gradient가 allgather로 모인 optimizer에 적용되어 step 단계에 파라미터를 업데이트.
3) Stage 3 optimizer state, gradient, parameter 모두를 나누어 가짐. 각각의 모델은 자신이 가지고 있는 parameter의 gradient와 optimizer state만을 가짐. Forward and backward 모두에서 파라미터를 가져와야 하므로 allgather 작업 수행, gradient를 계산하고, 각각의 GPU가 각각의 파라미터만 업데이트
4) Offload: CPU 메모리를 이용. CPU 메모리에서 parameter 혹은 optimizer state를 관리.

• (40점) ZeRO Stage 1, 2, 3 각각에 맞는 Deepspeed Configuration 값을 확인하고, 각 설정값이 의미하는 바를 설명하라. 또한, FP16 및 CPU Offloading을 활성화하기 위해 필요한 추가 argument들을 명시하고, 각 설정이 어떤 역할을 하는지 설명하라.

1) Zero_optimization: {“stage”: i} for i in ZeRO Stage [1, 2, 3]. Stage: i 는 ZeRO Stage i 옵션을 실행시킨다.
2) "fp16": { "enabled": true, "loss_scale": 0, "loss_scale_window": 1000, "hysteresis": 2, "min_loss_scale": 1 }. Enabled=True를 제외하고는 loss_scaling관련 파라미터이며, 각각 loss_scale: ‘자동 = 0’, loss_scale_window: window마다 재조정, hysteresis: 민감도, min_loss_scale: 최소 scaling의 값을 의미한다.
3) "zero_optimization": { "stage": 3,
"offload_optimizer": { "device": "cpu", "pin_memory": true },

"offload_param": { "device": "cpu", "pin_memory": true } ₩

}. Zero stage 3을 기준으로 했으며, offload_optimizer는 optimizer를, offload_param,은 parameter를 cpu로 넘김.(device : cpu). Pin_memory: whether to use pinned memory

• (40점) 실행 파일(run.sh)의 1번부터 6번까지의 command에 따라 ZeRO Stage에 따른 GPU 메모리 사용 량을 확인하라. 각 실행 단계에서 forward, backward, optimizer step 동안의 메모리 사용량을 체크하고, 만약 중간에 문제가 발생한다면 그 이유를 설명하라.
torch.cuda.memory allocated()를 사용하면 GPU 메모리 사용량을 확인할 수 있다.

1) 1번(zero 1)
   forward 전: 10,179,905,024
   forward 후: 20,256,247,568
   backward 후: 10,179,905,024,
   optimizer step 후: 10,179,905,024
2) 2번(zero 2)
   forward 전: 5,477,366,272
   forward 후: 15,553,698,816
   backward 후: 5,477,366,272
   optimizer step 후: 5,477,366,272
3) 3번(zero 2 – offload)
   forward 전: 4,953,077,760
   forward 후: 15,029,409,792
   backward 후: 4,953,077,760
   optimizer step 후: 4,953,077,760
4) 4번(zero 3)
   forward 전: 5,540,048,384
   forward 후: 18,918,346,752
   backward 후: 5,803,241,472
   optimizer step 후: 5,540,048,384
5) 5번(zero 3 offload)
   forward 전: 3,602,097,152
   forward 후: 16,946,578,432
   backward 후: 3,865,513,472
   optimizer step 후: 3,602,097,152
6) 6번(zero 3 fp16)

forward 전: 2,617,984,000
forward 후: 9,385,241,600
backward 후: 2,749,418,496
optimizer step 후: 3,394,085,376

• (40점) ZeRO Stage에 따른 20 iteration 동안의 학습 시간을 측정하라. 실행 파일(run.sh)의 1번부터 6 번까지의 command에 따라 학습시간을 측정하고, 중간에 문제가 발생하면 그 원인을 기술하라.

```
parser.add_argument("--epochs", default=20, type=int)
```

Argument 수정(default = 3 -> default = 20). 총 20 Epoch 돌게끔 시행. 결과는 다음과 같다.

1) 1번: Epoch 1, Step 140, Loss: 0.1813080757856369 이후 시간초과로 job cancelled.
2) 2번: Epoch 1, Step 40, Loss: 1.3389266729354858 이후 시간초과로 job cancelled.
3) 3번: Epoch 1, Step 40, Loss: 1.8668826818466187 이후 시간초과로 job cancelled.
4) 4번: Epoch 1, Step 50, Loss: 2.495577096939087 이후 시간초과로 job cancelled.
5) 5번: Epoch 1, Step 30, Loss: 0.06567217409610748 이후 시간초과로 job cancelled.
6) 6번: Epoch 1, Step 90, Loss: 0.09478759765625 이후 시간초과로 job cancelled.

Zero가 메모리를 분산하는 데는 효율적이지만 학습시간이 더 소요된다는 것을 알 수 있다.