Machine Translated by Google

Colossal-AI: 통합 딥 러닝 시스템 대규모 병렬 훈련용

리 셩구이1 , Jiarui Fang1 , Zhengda Bian1 , 홍신 리우1 , 율량 리우1 , Haichen Huang1 , Boxiang Wang1 , 그리고 양 유1,2

{lisg, fangjr, bian.zhengda, liuhongxin, liuyuliang, hhc, boxiangw}@hpcaitech.com
youy@comp.nus.edu.sg 1HPC-AI Technology Inc.

2싱가포르국립대학교(NUS)
Yang You는 NUS의 교수진이며 이 작업은 HPC-AI Technology Inc.에서 수행되었습니다.

개요 - Transformer 모델의 성공으로 딥 러닝 모델 규모가 수십억 개의 매개변수로 확대되었습니다. 단일 GPU의 제한된 메모리 리소스로 인해 최적의 병렬 전략을 선택하는 모범 사례는 여전히 부족합니다. 딥 러닝과 병렬 컴퓨팅 모두에 대한 도메인 전문 지식이 필요하기 때문입니다.

Colossal-AI 시스템은 모델 학습의 순차적 코드를 분산 환경으로 확장하는 통합 인터페이스를 도입하여 위의 문제를 해결했습니다. 데이터, 파이프라인, 텐서 및 시퀀스 병렬 처리와 같은 병렬 교육 방법은 물론 제로 중복 최적화 프로그램과 통합된 이기종 교육 방법을 지원합니다. Colossal-AI는 기준 시스템과 비교하여 대규모 모델에서 최대 2.76배의 교육 속도 향상을 달성할 수 있습니다.

색인 용어—머신 러닝, 딥 러닝, 분산 시스템, 병렬 컴퓨팅, 이기종 시스템

I. 서론

딥 러닝은 많은 응용 분야에서 인상적인 성능을 선보였으며 기존 방법으로는 해결하기 어려운 문제에 획기적인 발전을 가져왔습니다. BERT[1], Vision Transformer[2]와 같은 신경망은 많은 양의 데이터를 통해 인간도 경쟁할 수 없는 수준으로 고차원적 특징을 학습하고 예측할 수 있습니다. 강력한 하드웨어를 사용할 수 있게 됨에 따라 신경망에는 더 다양한 아키텍처와 더 많은 수의 매개 변수가 있습니다.

AI 커뮤니티의 분명한 추세는 딥 러닝 모델이 빠르게 커지고 있다는 것입니다. 2020년에 가장 큰 모델의 제목을 BERT-Large에서 GPT 2[3]로 옮기는 데 불과 3개월이 걸렸으며, GPT-2의 매개변수 수는 BERT-Large보다 약 5배 더 큽니다. 또한 GPT-2는 1750억 개의 매개변수를 가진 GPT-3[4]로 진화합니다. 보다 최근에는 GLM[5]이 놀랍게도 1.75조 개의 매개변수로 타이틀을 차지했습니다. 이러한 대규모 모델은 더 많은 데이터와 컴퓨팅 리소스를 사용하며 일반적으로 소규모 모델보다 많은 도메인에서 일반성과 성능이 더 우수합니다. 이러한 추세는 보다 강력한 컴퓨팅 하드웨어와 더 큰 데이터 세트가 등장함에 따라 계속될 것으로 예상됩니다. 결과적으로 기존의 훈련 방법은 더 이상 효과적이지 않으며 대규모 모델 훈련을 위해서는 분산 훈련이 필요합니다.

GPU와 같이 일반적으로 사용되는 가속기 하드웨어의 제한된 고속 메모리는 모델을 수십억 개의 매개변수로 확장하는 병목 현상입니다. 적응형 옵티마이저[6], [7]는 거대한 모델 훈련에 널리 적용되기 때문에 딥 러닝의 메모리 소비는 모델 매개변수, 레이어 활성화, 기울기 및 옵티마이저 상태에서 비롯됩니다. 모델 매개변수, 그래디언트 및 옵티마이저 상태를 모델 데이터로, 레이어 활성화를 비모델 데이터로 참조합니다. 모달 데이터의 총 메모리 소비는 매개변수만 소비하는 것보다 몇 배 더 클 수 있으므로 단일 GPU는 더 이상 대규모 모델 교육에 충분하지 않습니다. Float 16 형식의 100억 개의 매개변수는 이미 20GB의 모델 메모리를 사용할 수 있는 반면 일반적인 GPU에는 16GB 또는 32GB의 메모리만 있습니다. 최적화 없이 하나의 데이터 샘플로 100억 개의 매개변수 모델을 교육하면 일반적인 GPU보다 훨씬 많은 80GB 이상의 메모리가 필요할 수 있습니다.

데이터 병렬 처리는 구현이 쉽기 때문에 딥 러닝 프레임워크에서 지원하는 가장 인기 있는 병렬 처리 방법입니다. 인기에도 불구하고 이 방법은 전체 모델을 단일 장치에 보관해야 하므로 ResNet[8]과 같은 소규모 모델에만 적합합니다. GPU의 메모리 공간을 줄이기 위해 많은 방법이 제안되었습니다. 활성화 체크포인팅[9]은 계산을 메모리와 교환하여 비모델 데이터를 줄이는 데 사용되었습니다. 모델 데이터에 대한 메모리 제약을 해결하기 위해 파이프라인 병렬 처리[10], [11] 및 텐서 병렬 처리[12]와 같은 병렬화 기술을 탐색하여 모델 데이터를 샤딩하고 교육 효율성을 높여 한 번에 모델을 교육할 수 있도록 했습니다. 더 큰 규모. 스케일링 문제에 대한 솔루션을 제공하는 최신 시스템에는 GShard[13], FairScale[14], Megatron-LM[15] 및 DeepSpeed[16]가 포함됩니다. 이러한 시스템 중 Megatron-Lm과 DeepSpeed는 오픈 소스 커뮤니티에서 가장 많이 사용되며 최고의 성능을 제공합니다. 따라서 그들은 실험의 기준선으로 선택됩니다. Megatron-LM은 최적화된 파이프라인과 텐서 병렬성을 활용하여 Transformer 기반 모델을 훈련하기 위해 제안되었습니다. 한편, DeepSpeed 팀의 연구에 따르면 데이터 병렬 처리에는 메모리 중복성 측면에서 본질적인 결함이 있습니다. 그들은 모델 관련 데이터를 완전히 분할하는 효율적인 방법을 제안했습니다.

Machine Translated by Google

데이터 병렬 훈련에서 메모리 중복을 제거합니다. 이 두 가지 효율적인 방법은 모델 훈련을 수백 개의 장치와 수십억 개의 매개변수로 확장할 수 있는 길을 열었습니다.

대부분의 딥 러닝 엔지니어와 연구원은 비분산 코드 작성에 익숙하기 때문에 병렬 및 분산 프로그래밍에 적응하기가 합리적으로 어렵습니다. 그러나 모든 범위의 가속 방법에 쉽게 액세스할 수 있는 시스템이 없습니다. Megatron-LM의 주요 문제 중 하나는 다중 장치 통신, 데이터 일관성 및 확장 효율성에서 추가 엔지니어링 노력을 도입한다는 것입니다. 더 많은 차원에서 병렬성을 탐색하려는 경우 특히 그렇습니다. 한편, 환경 설정 및 무작위성 등 주의를 기울여야 하는 다른 측면이 있습니다. 이러한 기술 장벽은 일반 AI 커뮤니티가 다차원 분산 교육에 대한 수요 증가에 대처하는 데 도움이 되지 않습니다. DeepSpeed의 경우, 새로운 제로 중복 최적화 프로그램에도 불구하고 데이터 병렬 교육에 대한 최적화만 제공하고 하이브리드 병렬 교육을 달성하기 위해 Megatron-LM과의 통합에 크게 의존합니다. 복잡한 구현으로 인해 시스템 성능이 저하되고 확장성이 저하됩니다.

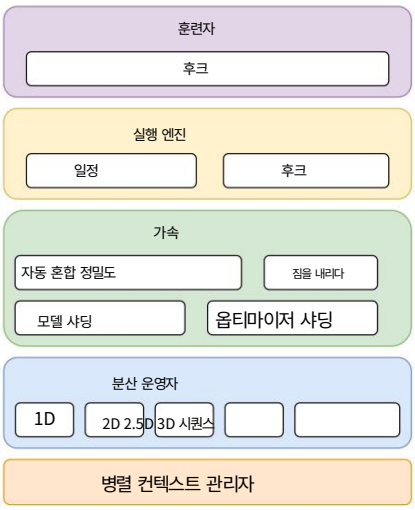


그림 1: Colossal-AI의 아키텍처

이러한 문제를 해결하기 위해 우리는 하나의 딥 러닝 시스템에서 일련의 훈련 가속 기술을 통합하여 AI 커뮤니티에서 복잡한 분산 훈련을 민주화하는 오픈 소스 시스템인 Colossal-AI를 개발했습니다. 이 시스템에는 다차원 텐서 병렬 처리 및 시퀀스 병렬 처리와 같은 새로운 병렬 처리 방법도 포함되었습니다. Colossal-AI는 사용자가 단일 노드 프로그램을 작성하는 코딩 습관을 유지하면서 사용자 친화적인 API를 제공하여 분산 교육을 쉽게 만드는 것을 목표로 합니다. 간단히 말해서 우리는 이 작업에서 대규모 분산 교육에 다음과 같은 주요 기여를 합니다.

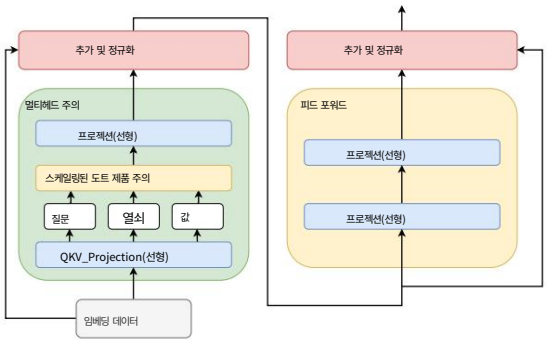


그림 2: Transformer Layer의 아키텍처

1 Colossal-AI는 통합 딥 러닝 시스템이며 AI 커뮤니티를 위한 데이터, 파이프라인과 같은 모든 가속 기술 세트를 제공합니다. 그림 1과 같은 모듈식 설계를 통해 Colossal-AI는 이러한 기술을 자유롭게 조합하여 최고의 교육 속도 향상을 달성할 수 있습니다. 한편, 모듈 설계 시 확장성을 고려했으며 Colossal AI를 통해 사용자 맞춤형 기능을 쉽게 지원할 수 있습니다. 시스템 아키텍처의 세부 사항은 구현 섹션에서 설명합니다.

2 Colossal-AI에서는 최적화된 병렬 처리 및 이기종 학습 방법을 제공합니다. 이러한 방법은 기준 시스템보다 더 나은 시스템 성능을 달성합니다.

최소한의 코드 변경으로 친숙한 API를 통해 사용자에게 제공됩니다.

3 서로 다른 하드웨어 조건에서 적합한 병렬 처리 전략을 조사하기 위해 심층 분석을 수행했습니다.

II. 배경

Transformer 아키텍처의 출현 덕분에 딥 러닝 모델은 컴퓨터 비전 및 자연어 처리와 같은 영역에서 전례 없는 성능 향상을 얻었습니다. BERT[1] 및 비전 트랜스포머[2] 각각의 영역에서 기본적인 기본 모델이 되었습니다. 단일 Transformer 계층의 일반적인 아키텍처는 그림 2에 나와 있습니다. Transformer 계층은 Multi-head Attention 블록과 Feed Forward 블록으로 구성됩니다. 파란색 사각형으로 표시되는 일반적인 Transformer 레이어에는 총 4개의 선형 레이어가 있습니다. Transformer 레이어를 사용하면 레이어 수, 선형 레이어의 숨겨진 크기, Multi-head Attention 모듈의 어텐션 헤드 수와 같은 하이퍼 매개변수에 따라 모델 크기를 확장할 수 있습니다. 결과적으로 모델 크기는 수십억 개의 매개변수에 쉽게 도달할 수 있습니다. 모델 크기의 급격한 증가는 인상적인 성능 향상을 가져옵니다. 예를 들어, GPT-3(1,750억 개의 매개변수)[4]은 LAMBADA 언어 작업[17]에서 예측 정확도가 절대적으로 18% 증가하여 최첨단 모델보다 성능이 뛰어납니다.

Transformer 아키텍처를 통해 모델 크기를 쉽게 확장할 수 있고 컴퓨팅 및 통신 하드웨어가 더욱 강력해짐에 따라 AI 엔지니어와 연구원은 시간 비용 절감을 위해 단일 GPU 교육에서 분산 교육으로 교육 패러다임을 전환했습니다. 분산 교육을 통해 딥 러닝 커뮤니티는 허용 가능한 작업 완료 시간을 유지하면서 더 큰 데이터 세트로 더 큰 모델을 교육할 수 있습니다. 다양한 기술들이 있었다

Machine Translated by Google

분산 교육을 가속화하기 위해 제안되었으며 아래에서 논의될 것입니다.

A. 데이터 병렬성

데이터 병렬 처리는 단순성으로 인해 가장 일반적인 형태의 병렬 처리입니다. 데이터 병렬 교육에서 모델은 장치 전체에 복제되고 데이터 세트는 여러 샤드로 분할됩니다. 각 데이터 세트 샤드는 장치에 할당되고 그림 3a와 같이 해당 장치의 모델에 공급됩니다. 각 장치에는 전체 매개변수 세트가 있으므로 모든 훈련 반복에 대해 모델 가중치를 동기화해야 합니다. 따라서 역방향 전파 중에 매개변수의 기울기가 계산되면 기울기가 줄어들고 옵티마이저는 동기화된 기울기로 매개변수를 업데이트합니다. 데이터 병렬 처리를 통해 모델을 쉽게 교육하고 배포할 수 있으며 추가 통신 오버헤드로 인해 GPU 수에 따라 부선형 확장이 가능합니다.

B. 모델 병렬성

데이터 병렬 훈련에서 눈에 띄는 특징 중 하나는 각 GPU가 전체 모델 가중치의 복사본을 보유한다는 것입니다. 그러나 이로 인해 특히 Adam [18]과 같은 상태 저장 최적화 프로그램을 사용할 때 대규모 모델의 메모리 요구 사항을 충족하기가 어렵습니다. GPU 메모리는 모델의 규모가 커지면 병목 현상이 됩니다. 따라서 각 장치가 모델 가중치의 일부만 보유하면 되도록 모델 매개변수를 샤딩해야 합니다. 결과적으로 이 문제를 해결하기 위해 모델 병렬 처리가 제안되었습니다. 일반적으로 모델 병렬 처리에는 텐서 병렬 처리, 파이프라인 병렬 처리 및 시퀀스 병렬 처리의 세 가지 하위 범주가 있습니다. 파이프라인 병렬 처리는 모델을 계층별로 분할하고 각 장치가 계층의 일부를 보유합니다.

1) 텐서 병렬화 텐서 병렬화는
장치 배열에 대해 텐서를 샤딩하고 그림 3b와 같이 산술 계산을 위한 분산 행렬-행렬 곱셈 알고리즘이 필요합니다.

Megatron-LM[12]은 Transformer 아키텍처[19]에 대한 행 또는 열 차원에서 선형 레이어를 분할하는 1D 텐서 병렬 처리를 제안했습니다.

트랜스포머 레이어의 MLP(다층 퍼셉트론) 모듈을 예로 들면 그림 4에서와 같이 Y = W2W1X 의 행렬 곱으로 MLP 모듈을 볼 수 있습니다 . 여기서 X는 숨겨진 상태이고 W1 및 W2 는 선형 계층 가중치이고 Y는 계층 출력입니다. 활성화 함수와 정규화 계층은 그림 4에서 무시됩니다.

일반적인 Transformer 모델에서 숨겨진 상태는 W1 에 의해 더 높은 차원으로 투영된 다음 W2에 의해 원래 차원으로 다시 투영됩니다. 열 차원의 행렬 W1 과 행 차원의 행렬 W2를 분할할 수 있습니다.

이렇게 하면 각 장치에서 부분적으로 Y가 표시됩니다. 행렬 곱셈의 올바른 최종 결과를 얻기 위해 부분 결과에 모두 축소 연산을 적용할 수 있습니다. Transformer 아키텍처의 Self Attention 모듈은 유사한 분산 방식으로 계산됩니다.

이러한 방식으로 각 장치는 N 장치에서 교육할 때 MLP 매개 변수의 1/N을 보유합니다. 이렇게 하면 모델 크기를 산술 정확성을 보장하기 위해 장치 간에 추가 통신이 필요하더라도 장치 수에 따라 확장하고 단일 장치의 메모리 용량을 초과합니다.

현재 방법은 Megatron-LM [12]에서 제안한 1D 텐서 병렬 처리에 의존합니다. 그러나 이 방법에는 통신 및 메모리 효율성을 해치는 몇 가지 단점이 있습니다.

이 방법의 주요 문제점 중 하나는 집단 커뮤니케이션을 위해 모두 축소 작업에 의존한다는 것입니다. 이로 인해 단일 노드의 GPU 간에 완전히 연결된 하이엔드 NVLink가 없는 시스템에는 비우호적입니다.

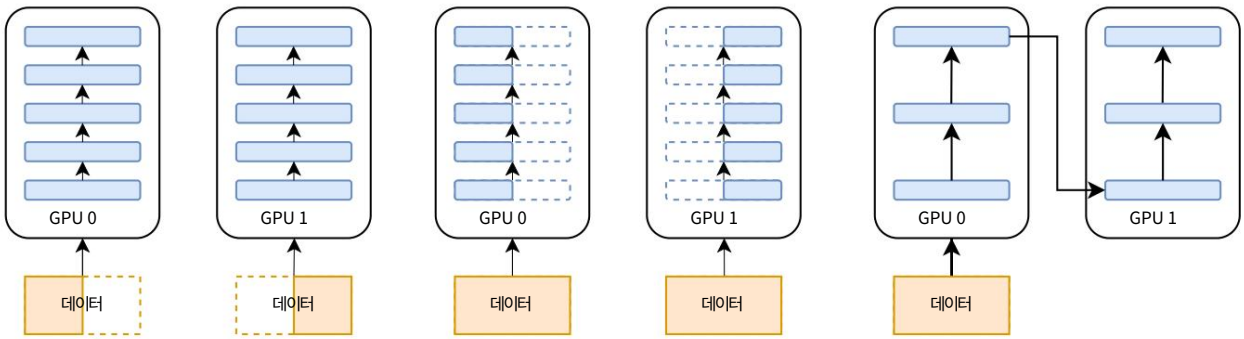
그림 5a와 같이 완전히 연결된 고성능 GPU 상호 연결을 통해 모든 GPU 쌍 간의 통신 대역폭은 지속적으로 높습니다. 그러나 이러한 고급 하드웨어는 비싸고 희소합니다. 일부 슈퍼컴퓨팅 센터에 있는 머신을 포함한 많은 머신에는 그림 5b와 같이 GPU가 부분적으로만 연결되어 있습니다. 이러한 종류의 GPU 토폴로지에서는 통신이 PCIe 버스를 거쳐야 하므로 멀리 떨어진 장치 간의 통신 대역폭이 직접 연결된 GPU보다 훨씬 낮습니다. 따라서 낮은 통신 대역폭은 1D 텐서 병렬 처리에서 전체 축소 작업의 효율성을 방해할 수 있습니다.

또한 1D 텐서 병렬 처리에는 계층 입력 및 출력에서 중복 메모리 사용이 있습니다. 트랜스포머 아키텍처의 MLP 계층을 예로 들면 그림 4와 같이 MLP 계층의 입력 X와 출력 Y가 서로 다른 장치에 걸쳐 복제됩니다. 이러한 메모리 중복성은 제한된 하드웨어 리소스에서 학습할 수 있는 최대 모델 크기를 제한합니다. , 대규모 분산 교육의 민주화에 도움이 되지 않습니다.

1D 텐서 병렬 처리 외에도 그림 6과 같이 2D, 2.5D 및 3D 텐서 병렬 처리[20]-[22]와 같이 대규모 모델 훈련을 위해 보다 발전된 텐서 병렬 처리가 도입되었습니다. 딥러닝 훈련을 통해 기억력과 의사소통 효율에서 장점이 있습니다. 따라서 서로 다른 하드웨어 사양에 더 잘 대처할 수 있습니다. 이는 사용자에게 자신의 기계에 가장 적합한 텐서 병렬 처리 기술을 사용할 수 있는 옵션을 제공합니다.

2D 텐서 병렬 처리[20]는 SUMMA 및 Can 비 행렬 곱셈 알고리즘[23]-[25]에 의존하며 입력 데이터, 모델 가중치 및 레이어 출력을 두 개의 서로 다른 차원으로 분할합니다. N개의 장치가 주어지면 텐서는 N개의 청크로 분할되고 각 청크는 하나의 GPU가 소유합니다. GPU는 그림 6a와 같이 각 행에 √N개의 GPU가 있고 √N개의 행이 있는 정사각형 네트워크 토폴로지로 배열됩니다. 예를 들어 [P, Q] 모양의 텐서는 [P/√N, Q/√N] 모양의 청크 텐서로 분할됩니다. 이 방법은 입력과 출력이 더 이상 중복되지 않기 때문에 제로 메모리 중복성으로 이어져 1D 텐서 병렬 처리에 비해 메모리 소비가 훨씬 적습니다. 산술 정확성을 보장하기 위해 부분 입력과 가중치는 브로드캐스트와 같은 집단 통신을 통해 장치 간에 통신했습니다. 2D 텐서 병렬 처리는 모델 가중치에 대한 추가 통신 오버헤드가 있지만 1D 텐서 병렬 처리는 그렇지 않지만 2D 텐서 병렬 처리는 더 미세한 분할 체계로 인해 전송되는 데이터 양이 적기 때문에 여전히 더 효율적입니다.

Machine Translated by Google



(a) 데이터 병렬 (b) 텐서 병렬 (c) 파이프라인 병렬

그림 3: 분산 훈련을 위한 기존 병렬 처리

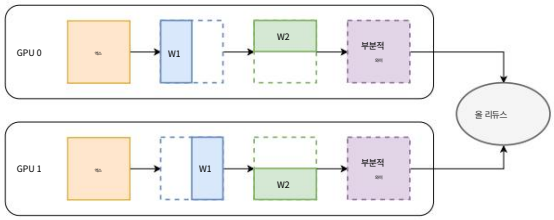
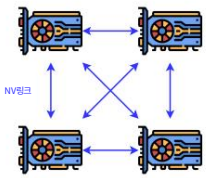
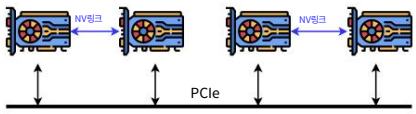


그림 4: Megatron-LM MLP 모듈



(a) 완전히 연결된 GPU



(b) 부분적으로 연결된 GPU

그림 5: GPU 컴퓨팅 노드의 공통 네트워크 토폴로지

많은 수의 장치로 확장됩니다.

2.5D Tesnor Paralleism[21]은 2.5D 행렬 곱셈 알고리즘[26]에서 영감을 받아 2D 텐서 병렬화를 더욱 병렬화하기 위해 제안되었습니다. 병렬화를 위해 행렬의 선택적 깊이 차원을 추가합니다. 깊이 = 1이면 2D 텐서 병렬 처리에 가깝습니다. 깊이 > 1인 경우 행렬을 3번 분할하고 병렬화 수준을 1개 더 추가합니다. N개의 장치가 주어지면 텐서는 N = S D와 같은 방법 2로 분할됩니다. 여기서 S는 정사각형의 한 변의 크기이고 D는 그림 6b에 표시된 큐브의 깊이입니다.

S는 사용자가 D를 지정하면 계산할 수 있습니다. 많은 수의 장치로 확장할 때 2.5D 텐서 병렬 처리는 통신량을 더 줄이기 때문에 2D 텐서 병렬 처리보다 더 효율적입니다.

3D 텐서 병렬화[22]는 3D 행렬 곱셈 알고리즘[27]을 기반으로 제안되었으며 Colossal-AI에도 통합되었습니다. 3D 텐서 병렬화는 그림 6d와 같이 텐서를 큐빅 방식으로 분할합니다. 모든 텐서에 3차원이 있는 것은 아니므로 첫 번째 차원이 두 번 분할되는 첫 번째 차원과 마지막 차원만 분할하도록 선택합니다. 예를 들어 [P, Q] 모양의 텐서는 [P/$\sqrt[3]{N}$, Q/$\sqrt[3]{N}$] 모양의 청크 텐서로 분할됩니다 . 이 방법은 N 장치에서 최적의 O(N1/3) 통신 오버헤드를 달성하는 동시에 계산 및 메모리 사용이 활성화뿐만 아니라 매개 변수의 최적화된 로드 밸런싱을 통해 균등하게 분산됩니다.

고급 텐서 병렬화 방법은 서로 다른 네트워크 토폴로지를 필요로 하므로 사용자는 GPU 수에 따라 방법을 선택해야 합니다. 1D 방법은 여러 GPU에서 작동할 수 있는 반면 2D, 2.5D 및 3D 방법에는 각각 n개의 GPU가 필요합니다. 여기서 a와 n은 양의 정수입니다. GPU 수가 요구 사항을 충족하지 못하는 경우 사용자는 1D 텐서 병렬 처리로 대체할 수 있습니다. 3 과 n

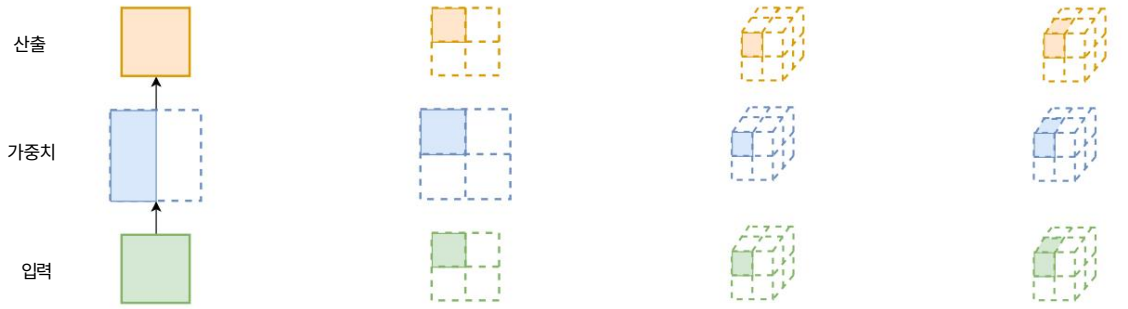
이러한 고급 텐서 병렬 처리 방법은 더 많은 장치[23], [24], [26], [27]로 확장할 때 더 적은 통신량을 제공합니다. 한편, 이러한 방법은 1D 텐서 병렬화에 존재하는 파이프라인 단계 간의 통신 오버헤드를 제거하므로 파이프라인 병렬화와의 호환성이 더 좋습니다. Megatron-LM에서 제안한 1D 텐서 병렬 처리에서는 레이어 활성화가 중복됩니다. 따라서 노드 간 통신 대역폭을 절약하기 위해 텐서는 다음 파이프라인 단계로 넘어가기 전에 청크로 분할하고, 그림 7과 같이 대상 파이프라인 단계에 도달한 후 노드 내 통신으로 청크를 수집합니다.

반대로 고급 텐서 병렬 처리 방법에서는 다음 단계로 넘어갈 텐서가 이미 전체 논리 텐서의 하위 청크입니다. 따라서 분할 수집 작업이 필요하지 않으며 추가 통신 비용이 발생하지 않습니다.

2) 파이프라인 병렬화 텐서 병렬

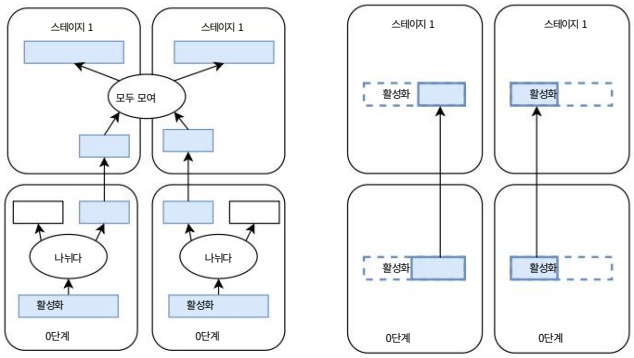
화 외에 또 다른 샤딩 체계는 모델을 계층별로 분할하는 것입니다. 텐서 병렬 처리와 비교할 때 파이프라인 병렬 처리는 계층의 계산 흐름을 변경할 필요가 없기 때문에 특히 더 친숙합니다.

Machine Translated by Google



(a) 1D 텐서 병렬 처리 (b) 2D 텐서 병렬 처리 (c) 2.5D 텐서 병렬성 (d) 3차원 텐서 병렬성

그림 6: 텐서 병렬성



(a) 1D 텐서 병렬 처리 (b) 2//2.5/3D 텐서 병렬성

그림 7: 텐서 병렬 처리를 사용한 파이프라인 단계 간 데이터 전송

PyTorch[28]와 같은 동적 그래프 기반 프레임워크. PipeDream [29] 및 GPipe [30]와 같은 방법은 모델을 여러 청크로 분할하고 각 청크는 그림 3c와 같이 장치에 할당되도록 제안되었습니다. 정방향 통과 동안 각 장치는 중간 활성화를 다음 단계로 전달하고 마지막 파이프라인 단계에서만 손실 값을 계산합니다. 역방향 전달 중에 각 장치는 입력 텐서의 기울기를 이전 파이프라인 단계로 다시 전달합니다. 기울기는 각 파이프라인 단계의 매개변수에 대해 누적되며 매개변수는 파이프라인에 마이크로 배치 데이터가 부족할 때만 업데이트됩니다. 파이프라인 병렬 처리를 통해 여러 장치가 동시에 계산할 수 있으므로 처리량이 높아집니다. 또한 파이프라인 단계 간에 활성화 및 기울기만 전달되므로 노드 간 통신을 최소화합니다. 파이프라인 병렬 훈련의 한 가지 단점은 다른 장치가 계산에 참여할 때 일부 장치가 유휴 상태가 되어 계산 리소스가 낭비되는 버블 타임이 있다는 것입니다. Chimera[31]와 같은 현재 방법은 양방향 파이프라이닝을 사용하여 버블 시간을 줄이고 최첨단 성능을 달성합니다.

3) 시퀀스 병렬화 Tensor 병렬

화는 주로 모델 데이터로 인한 메모리 병목 현상을 해결합니다. 그러나 비모델 데이터는 AlphaFold 및 문서 수준의 텍스트 이해. 이러한 작업은 시퀀스 길이가 긴 데이터를 사용하기 때문입니다. Transformer 계층의 self-attention 메커니즘은 시퀀스 길이에 대해 2차 복잡도이므로 긴 시퀀스 데이터는 중간 활성화에 소비되는 메모리 사용량을 증가시켜 장치의 교육 기능을 제한합니다. 고급 텐서 병렬 처리 방법이 입력과 활성화를 분할할 수 있지만 모델 매개변수로 인한 추가 통신 오버헤드는 필요하지 않습니다.

시퀀스 병렬 처리[32]는 큰 시퀀스 차원으로 인한 메모리 벽을 허물어 긴 시퀀스 모델링을 가능하게 하기 위해 제안되었습니다. 시퀀스 병렬 처리에서 모델은 데이터 병렬 처리와 마찬가지로 장치 간에 복제됩니다.

그러나 배치 차원으로 분할하는 대신 입력 데이터를 시퀀스 차원으로 분할하고 각 장치는 하위 시퀀스만 유지합니다. 데이터가 장치에 분산되기 때문에 self-attention 계산을 완료하려면 추가 통신이 필요합니다. 시퀀스 병렬 처리에서 부분 쿼리, 키 및 값 임베딩을 교환하기 위해 Ring Self Attention 모듈이 제안됩니다.

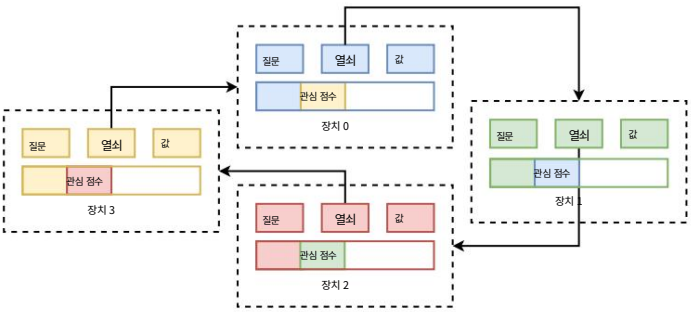


그림 8: Ring Self Attention의 주의 점수 계산

self-attention 메커니즘은 mula에 대해 다음과 같이 작동합니다.

주의(Q, K, V) = 소프트 tmax(QKT / dk)V

여기서 Q는 쿼리 임베딩, K는 키 임베딩, V는 값 임베딩, dk 는 키 임베딩의 차원입니다. 시퀀스 병렬 처리에서와 마찬가지로 쿼리, 키 및

Machine Translated by Google

값 임베딩은 모두 시퀀스 차원으로 분할되므로 계산을 완료하려면 부분 임베딩을 링 방식으로 통신해야 합니다. 어텐션 스코어의 계산을 예로 들면 어텐션 스코어는 A = soft tmax(QKT / √dk)로 구한다. Ring Self Attention 모듈에서 키 임베딩은 먼저 로컬 장치에 임베딩된 쿼리와 곱해진 다음 그림 8과 같이 N-1번 동안 다음 장치로 전송됩니다. 로컬 하위 시퀀스에 대해 얻을 수 있습니다. 최종 주의 출력 AV는 비슷한 방식으로 계산할 수 있습니다.

시퀀스 병렬 처리는 파이프라인 병렬 처리와 직교합니다.
고급 텐서 병렬 처리와 마찬가지로 레이어 활성화가 시퀀스 차원으로 분할되므로 파이프라인 단계 간에 추가 통신이 필요하지 않습니다.

C. 제로 중복성 데이터 병렬성

위의 방법은 주로 모델 매개변수의 양으로 인한 메모리 문제를 해결합니다. 그러나 모델 매개변수는 훈련 중 메모리 소비의 일부만을 형성합니다. 모델 데이터 중에서 옵티마이저 상태는 메모리 소비의 더 많은 부분을 차지할 수 있습니다. Adam[18]과 같은 상태 저장 최적화 프로그램을 사용할 때 최적화 프로그램 상태(예: 운동량 및 분산)는 모델 매개변수[18], [33]가 차지하는 메모리 공간에 비해 3배 더 큰 메모리 공간을 차지할 수 있습니다.

N개의 장치에 대한 데이터 병렬 교육에는 모델 매개변수, 그래디언트 및 최적화 상태의 N개 사본이 있습니다. 이러한 중복성을 제거하기 위해 DeepSpeed[16]에서 Zero Redundancy Optimizer가 제안되어 이러한 중복 모델 데이터를 여러 장치에 분할합니다. Zero Redundancy Optimizer에는 3단계가 있으며, 옵티마이저 상태, 그래디언트 및 모델 매개변수를 파티셔닝 체계에 단계별로 추가합니다. 마지막 단계를 예로 들면 각 장치는 모델 데이터의 1/N만 보유하고 해당 장치가 소유한 매개 변수의 파티션만 업데이트합니다. 모든 모델 데이터가 분할되어 있으므로 정방향 및 역방향 패스에서 집단 통신이 필요합니다. 정방향 단계에서는 중간 활성화 및 모델 출력을 계산하기 위해 분할된 모델 가중치를 수집하기 위해 모두 수집이 필요합니다. 뒤로가는 동안 모델 가중치가 다시 수집되어 그래디언트를 계산합니다. 그런 다음 분할된 그래디언트가 해당 장치로 축소됩니다. 각 장치는 기울기와 매개변수의 파티션만 보유하므로 전체 모델 매개변수 대신 분할된 매개변수만 업데이트합니다. 이 방법은 데이터 병렬 교육의 향상이므로 모델 병렬 처리와 직교하며 더 큰 모델을 지원하기 위해 모델 병렬 처리와 통합될 수 있습니다.

D. 이기종 훈련

GPU의 컴퓨팅 파워를 더 잘 활용하기 위해 노력하고 있지만, 일반적으로 CPU 메모리가 GPU 메모리보다 용량이 크다는 사실을 간과하는 경우가 많습니다. 따라서 CPU 메모리는 딥 러닝 훈련에서 메모리 제약을 깨는 데 필수적일 수 있습니다. 예를 들어 Tesla V100 GPU 8개가 장착된 일반적인 Nvidia DGX1 워크스테이션에서 집계된 GPU 메모리는 노드당 256GB이고 CPU 메모리는 512GB입니다. CPU 메모리를 모델 훈련에 활용할 수 있다면 모델 크기를 간단히 확장할 수 있습니다. 또한 NVMe 솔리드 스테이트 디스크(SSD)는 기존 하드 디스크 드라이브(HDD)에 비해 통신 대역폭이 상대적으로 높기 때문에 NVMe SSD를 사용하여 모델 데이터도 수용할 수 있어 단일 노드의 교육 용량을 더욱 높일 수 있습니다.

DeepSpeed는 더 큰 모델을 위한 공간을 만들기 위해 사용하지 않을 때 GPU에서 CPU 또는 NVMe 디스크로 텐서를 이동하는 제로 오프로드[34]를 제안했습니다. 고성능 이기종 저장 장치를 활용하고 서로 다른 하드웨어 장치 간에 Tensor를 지속적으로 교환함으로써 단일 GPU에서 수십억 개의 매개변수로 모델을 훈련할 수 있게 되었습니다.

이는 컴퓨팅 리소스가 제한된 사용자에게 특히 친숙하며 대규모 모델 교육의 민주화에 필수적입니다.

III. 설계

지난 10년 동안 컴퓨팅 리소스의 급속한 발전으로 하드웨어 사양은 슈퍼컴퓨팅 센터와 연구실에 따라 다양해졌습니다. 이로 인해 하드웨어 조건의 큰 변동으로 인해 딥 러닝 교육에 지속적으로 만족스러운 가속을 제공하는 보편적인 방법을 갖기가 어렵습니다. 이러한 문제는 Megatron-LM 및 DeepSpeed로도 해결할 수 있습니다. 대조적으로, Colossal-AI는 모듈 방식으로 구성된 일련의 가속 기술이 특징입니다. 이 가속 기술 무기고는 최대 성능을 달성하기 위해 광범위한 훈련 설정을 다룰 수 있습니다. 이 섹션에서는 Colossal-AI에 통합된 가속 기술의 구현 및 분석에 대해 논의합니다.

방법	총 통신량 / 전송된 요소 수
1D	2(p - 1) * Sx
2D	3(j - 1) * (Sx + Sw) 3(k
2.5D	- 1) * (Sx/d + Sw) 2(l - 1)/
3D	l * (Sx + Sw + Sy)

표 I: 실험을 위한 시스템 사양

A. 다차원 모델 병렬성

배경 섹션에서 언급했듯이 모델 병렬 처리는 단일 GPU의 메모리 부담을 줄이기 때문에 분산 교육에서 중요한 기술입니다. 따라서 장치 수가 증가함에 따라 모델 크기를 수십억 개의 매개변수로 확장할 수 있습니다. Colossal-AI는 분산 교육의 요구 사항을 충족하기 위해 일련의 모델 병렬 처리 방법을 제공합니다.

Transformer 모델의 주요 기능 중 하나는 아키텍처가 선형 계층에 크게 의존한다는 것입니다. 텐서 병렬화는 주로 행렬-행렬 곱셈에 적용되므로 Transformer 모델 훈련의 가속화에 매우 적합합니다.

Colossal-AI에서 기존의 모든 텐서 병렬 처리 방법은

Machine Translated by Google

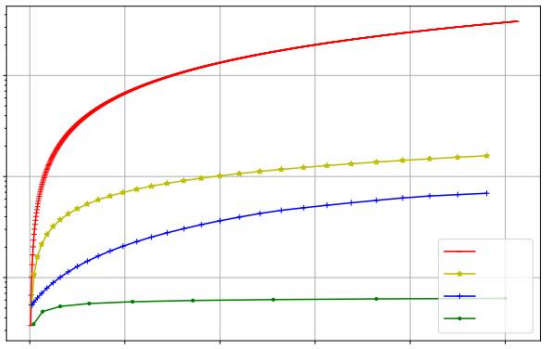


그림 9: 이론적 분석에서 Tensor Parallelism의 스케일링 성능(h = 1024, s = 512, b = 32)

사용자가 교육 요구 사항 및 GPU 수에 따라 하나의 방법을 선택할 수 있도록 지원됩니다.

모든 텐서 병렬화 방법 중에서 고급 텐서 병렬화, 즉 2D, 2.5D 및 3D 텐서 병렬화의 두드러진 이점 중 하나는 1D 텐서 병렬화에 비해 통신 비용이 저렴하다는 것입니다. 표 1은 행렬 곱셈 Y = WX를 계산할 때 총 통신량을 보여줍니다. 여기서 X는 (b, s, h) 모양, W는 (h, h) 모양, Y는 (b, h) 모양입니다. 표 I에서는 다음과 같은 표기법이 사용됩니다.

• p: 총 GPU 수 • j: 그림 6b와 같은 정
사각형 모양의 네트워크 토폴로지의 한 변에 있는 GPU 수, 여기서 $p = j^2$ •
k: 앞 사각형의 한 변에 있는 GPU 수 그림 6c와 같은 sqaure-shaped
네트워크 토폴로지, 여기서 $p = d * k^2$ • d: 입방체 모양 네트워크 토폴로지의
깊이 차원에서 GPU의 수

그림 6c

• l: 그림 6d와 같이 정육면체 형태의 네트워크 토폴로지의 한쪽에 있는 GPU의 수, 여기서 $p = l^3$입니다.
• Sx : 입력 행렬 X의 요소 수, SW 및 SY 에 동일한 의미가 적용됩니다 .

• b: 입력 행렬 X의 배치 크기. • s: 입력 행렬 X의
시퀀스 길이. • h: 가중치 W의 숨겨진 크기.

그림 9에서 볼 수 있듯이 고급 텐서 병렬 처리의 통신량은 특히 많은 수의 노드를 사용하는 경우 1D 텐서 병렬 처리보다 훨씬 적습니다. 통신 효율성의 근본적인 이유는 고급 텐서 병렬 처리가 컴퓨팅 노드의 하위 그룹에서만 통신을 발생시키기 때문입니다. 예를 들어, 2D 병렬성에서 집단 통신은 사각형 모양의 네트워크의 한 행 또는 한 열의 노드만 포함합니다. 대조적으로, 1D 텐서 병렬 처리는 하나의 집단 통신 호출에 대한 모든 컴퓨팅 노드를 포함합니다. 따라서 고급 텐서 병렬 처리는 하나의 노드 이상으로 확장할 수 있는 반면 1D 텐서 병렬 처리는 종종 노드 내 컴퓨팅으로 제한됩니다.

Colossal-AI는 텐서 병렬 처리 외에도 시퀀스 병렬 처리와 파이프라인 병렬 처리를 포함하여 하이브리드 병렬 처리를 사용하여 대규모 클러스터에서 모델 교육을 가속화할 수 있습니다.

B. 향상된 샤딩 및 오프로딩

DeepSpeed는 제로 중복 교육 및 오프로딩을 지원합니다. 제로 중복 교육은 모델 매개변수, 기울기 및 옵티마이저 상태를 분할하여 메모리 중복을 제거합니다. 오프로딩은 CPU 메모리와 NVMe 디스크를 활용하여 대규모 모델에 사용할 수 있는 저장 공간을 확장합니다. 그러나 불필요하게 복잡한 구현으로 인해 DeepSpeed의 효율성과 확장성이 떨어집니다. 따라서 더 나은 성능을 위해 샤딩 및 오프로딩 모듈을 재설계했습니다. 제로 중복 데이터 병렬 교육을 위해 통합 샤딩 텐서 인터페이스를 정의했습니다.

텐서 분할은 사용자 지정 가능한 샤딩 전략으로 지원할 수 있습니다. 이를 통해 모델 데이터와 옵티마이저 데이터를 느슨한 결합을 통해 일반적인 방식으로 샤딩할 수 있습니다.
교육 워크플로우는 사용자 정의 라이프 사이클 후크를 사용하여 쉽게 수정할 수도 있습니다. 이러한 방식으로 우리는 DeepSpeed의 제로 중복 메커니즘을 구현할 수 있을 뿐만 아니라 다른 세분성에서 더 많은 최적화를 위한 공간을 만들 수 있습니다. Colossal-AI에서는 확장 가능한 설계 덕분에 메모리 공간에 대해 PatrickStar[33]가 제안한 추가 최적화를 통합했습니다.

첫 번째 최적화는 더 큰 모델을 훈련할 수 있도록 메모리의 fp16 저장 공간을 재사용하는 것입니다.
순방향 패스에서는 fp16 매개변수를 유지하고 그라데이션이 없습니다. 역방향 패스에서 fp16 매개변수는 그래디언트를 계산하는 데 사용됩니다. 그래디언트가 완전히 계산되면 fp16 매개변수는 필요하지 않습니다. 그런 다음 순방향 및 역방향 패스 중에 fp16 매개변수를 보유하는 저장 공간에 fp16 그래디언트를 저장할 수 있습니다. 이 프로세스는 하위 모듈의 세분성에서 스트리밍됩니다. 이렇게 하면 그림 10과 같이 중복성과 최대 메모리 사용량을 더욱 줄일 수 있습니다. 이러한 방식으로 CPU 메모리는 더 큰 모델을 수용할 수 있습니다.

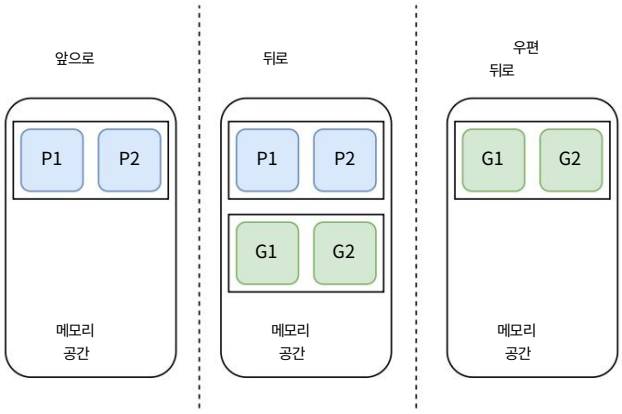


그림 10: 메모리 공간 재사용

두 번째 최적화는 적응형 하이브리드 Adam 옵티마이저입니다.
DeepSpeed는 다음과 같은 경우 옵티마이저 작업에 CPU Adam을 사용합니다.

Machine Translated by Google

플로딩이 활성화되었습니다. 모든 매개변수는 오프로드되고 CPU 메모리에서 업데이트됩니다. 대조적으로 Colossal-AI의 하이브리드 Adam 옵티마이저는 GPU에서 사용 가능한 메모리 공간을 모니터링합니다.
공간이 남아 있는 한 GPU의 일부 매개변수와 그래디언트를 유지합니다. 이러한 방식으로 매개변수가 CPU와 GPU 모두에서 업데이트되어 시간 비용이 절감됩니다.

오프로딩에는 PCIe를 통해 서로 다른 장치 간에 텐서 스와핑이 필요하므로 추가적인 통신 오버헤드가 발생합니다. 그러나 이렇게 하면 모델 학습에 사용할 수 있는 메모리 공간이 더 많이 생성됩니다. 이것은 본질적으로 두 가지 이점을 가져옵니다. 1) 입력 데이터의 크기가 일정하면 더 큰 모델이 GPU에 수용될 수 있고 더 큰 모델이 더 나은 예측 성능을 가져올 수 있습니다. 2) 모델의 크기가 일정하면 PCIe를 통한 추가 통신에도 불구하고 훈련 처리량이 향상될 수 있도록 더 큰 배치 크기를 사용할 수 있습니다.

IV. 구현

Colossal-AI는 PyTorch로 구현된 딥러닝 시스템입니다. Colossal-AI의 전체 아키텍처는 그림 1에 나와 있습니다. Colossal-AI의 기본 구성 요소는 병렬 컨텍스트 관리자입니다. 분산 네트워크 토폴로지와 같은 복잡한 하이브리드 병렬 분산 환경에 대한 메타 정보를 저장하는 역할을 합니다. 병렬 컨텍스트에 따라 분산 연산자는 해당 병렬 모드로 자동 전환됩니다. 예를 들어 하이 레벨 계층의 계산 흐름이 이러한 방식으로 복잡한 계산 및 통신 워크플로우가 사용자에게 숨겨지고 쉽게 텐서 병렬 모델을 구축할 수 있습니다.

```
# 분산 네트워크 시작
colossalai.launch_from_torch(config='./config.py')

# 1D 텐서 병렬 처리를 사용하여 지정
# 병렬 크기 4
병렬 = dict(tensor=dict(size=4, mode='1d'))
```

목록 1: 구성 파일

3) 사용자 친화성: Colossal-AI는 사용자 코드 변경을 최소화하기 위해 모델 학습을 위한 사용자 친화적인 API를 제공합니다. 사용자는 목록 1과 같이 구성 파일만 준비하면 됩니다. 구성 파일은 미리 정의된 스키마에 따라 기능을 지정하는 간단한 Python 파일입니다. Colossal-AI는 목록 1과 같이 'colossalai.initialize'를 사용하여 실행 엔진에 가속 기능을 주입합니다.

Colossal-AI는 또한 사용자가 편리하게 직접 사용할 수 있는 병렬화된 인기 모델 구성 요소를 제공합니다. 이를 위해서는 사용자가 GShard[13]와 같은 병렬 전략을 수동으로 설계할 필요가 없도록 도메인 전문 지식이 필요하지 않습니다.

```
수입 콜로살라이

# 교육 구성 요소 정의
...

옵티마이저, 기준,
트레인로더, 테스트
로더, lr_scheduler)
```

모델 구축 프로세스에서 분리됩니다. 사용자는 훈련을 실행할 수 있습니다 . 기본은 연산자를 사용하여 데이터 에 대해 를 구성하고 train_dataloader : data 는 다양한 가속 engine.zero_grad() 도구가 있습니다. 기본 도구에는 활성화 체크포인팅 engine.criterion(output, label) engine.backward(train_loss)가 대규모 모델 lr_scheduler.step() 교육 프로세스 를 제어 하고 트레이너는 라인에서 교육을 한 교육 일정 및 후크를 정의할 수 있으므로 사용자 맞춤화를 위한 뛰어난 확장성을 제공합니다.

PyTorch 연산자와 같
label = label.cuda() 를 실행
할 모델을 실행할 수 있습니다 . 위의 한 계층에
니다. 고급 기능은 train_loss =
호환성이 됩니다. 마지막으로 실행 엔진은
및 후크를 정의할 수 있으므로

목록 2: Colossal-AI 사용

1) 모듈성: 모듈성 및 확장성의 원칙은 개발 전반에 걸쳐 유지되며 최대 성능을 추구하기 위해 다양한 가속 기술을 쉽게 결합할 수 있습니다.

2) 확장성: Colossal-AI는 지속적으로 개발 중인 시스템으로 향후 확장을 위한 맞춤형 기능을 구현하기 위한 다양한 인터페이스를 제공합니다. 예를 들어, 샤딩 모듈을 통해 사용자는 보다 효율적인 교육 방법을 탐색하기 위해 자신의 샤딩 전략과 수명 주기 후크를 정의할 수 있습니다.

V. 평가

A. 실험 설정

Colossal AI의 시스템 성능을 전체적으로 평가하기 위해 다양한 하드웨어에 대해 다양한 실험을 수행했습니다. 시스템 사양은 표 II에 나열되어 있습니다. 리소스 제약으로 인해 실험 항목 열에 명시된 대로 각 시스템의 주요 기능 중 일부만 테스트했습니다.

우리는 Megatron-LM과 DeepSpeed를 실험의 기준선으로 사용했습니다. Colossal-AI에서 Megatron-LM 텐서 병렬 처리가 구현되었으므로 실험 결과에서 1D 텐서 병렬 처리로 주석 처리됩니다.

Machine Translated by Google

체계 ID	#GPU 노드 당	#Nodes	GPU 모델	GPU 상호 연결	노드 간 상호 연결	실험항목
나	8	1	엔비디아 A100(80GB) NV링크		해당 없음	텐서 병렬성
II	8	1	Nvidia A100(80GB) 인접한 GPU 간 NVlink, 멀리 있는 GPU 간 PCIe		해당 없음	텐서 병렬 처리, Zero
III	4	16	엔비디아 A100(40GB) NV링크		인피니밴드 HDR (200Gb/s) 및 Dragonfly 네트워크 토폴로지	텐서 병행, 시퀀스 병렬성
IV	1	64	엔비디아 P100(16GB) RDMA		Cray Aries 라우팅 및 통신 ASIC 및 Dragonfly 네트워크 토폴로지	텐서 병렬성

표 II: 실험을 위한 시스템 사양

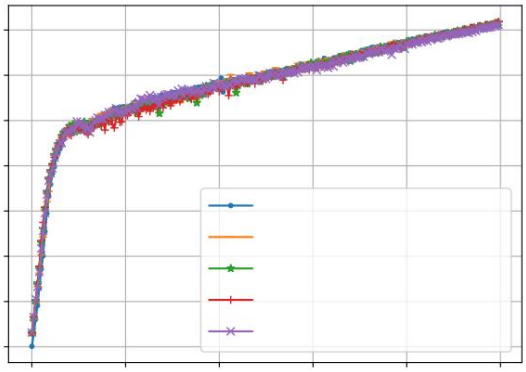


그림 11: ImageNet에서 ViT의 융합 성능

B. 다차원 텐서 병렬성

1) Convergence 우

리는 System III에서 다차원 텐서 병렬성의 산술적 정확성과 수치적 안정성을 검증하기 위해 ImageNet-1k 데이터셋에서 ViT(Vision Transformer)[2]로 실험을 수행했습니다. ViT 모델은 숨겨진 크기가 384인 12개의 트랜스포머 레이어와 6개의 어텐션 헤드를 갖도록 구성됩니다. Jax 초기화와 AdamW 옵티마이저를 0.003 학습률과 0.3 가중치 감쇠로 사용했습니다. 입력 이미지의 모양은 224이고 패치 크기는 16입니다. 전역 배치 크기는 4k이고 모델은 250 epoch 동안 훈련됩니다. 기준선은 각 장치가 완전한 계산 그래프를 유지하므로 데이터 병렬성을 사용한 모델 교육입니다. 그림 11에서 볼 수 있듯이 다차원 텐서 병렬 처리의 테스트 정확도 곡선은 기준선의 정확도 곡선과 정렬되어 다차원 텐서 병렬 처리가 모델 수렴에 영향을 미치지 않음을 보여줍니다.

2) 메모리 효율성 2D, 2.5D,

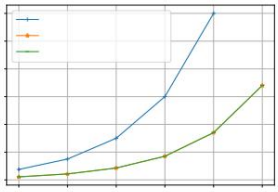
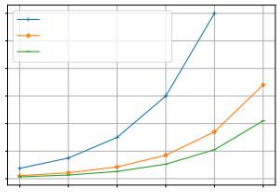
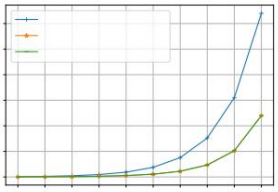
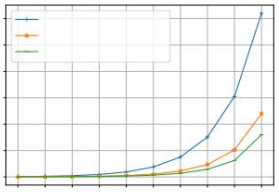
3D 텐서 병렬 처리는 입력 데이터, 레이어 가중치 및 출력 활성화를 분할하는 반면 1D 텐서 병렬 처리는 레이어 가중치만 분할하므로 전자가 메모리 사용량이 적을 것으로 예상됩니다. 그 결과 처음 세 메서드를 사용하면 GPU가 더 큰 모델을 수용할 수 있습니다. 메모리 효율성을 입증하기 위해 시스템 I에서 배치 크기와 숨겨진 크기로 확장되는 두 가지 범위 테스트를 수행했습니다. 이 범위 테스트에서는 두 개의 선형 계층으로 구성된 모델을 만들었습니다. 우리는 4개의 GPU에서 1D, 2D 및 2.5D 실험을 실행하고 8개의 GPU에서 1D, 2.5D(깊이=2) 및 3D 실험을 실행합니다. 순방향 및 역방향 패스 동안 최대 할당 CUDA 메모리를 측정하고 그 결과를 그림 12에 표시했습니다. 1D 텐서 병렬 처리의 메모리 소비는 2D, 2.5D 및 3D 텐서 병렬 처리보다 훨씬 높습니다. 배치 크기가 512 및 8 GPU인 경우 2.5D 및 3D의 메모리 소비는 그림 12b의 1D 텐서 병렬 처리보다 각각 44% 및 65% 낮습니다. 숨겨진 크기가 16384이고 GPU가 8개인 경우 2.5D 및 3D 텐서 병렬 처리의 메모리 성능은 그림 12d의 1D 텐서 병렬 처리보다 각각 62% 및 74.2% 더 좋습니다. 따라서 초대형 모델로 확장할 때는 고급 텐서 병렬 처리가 더 나은 옵션입니다.

3) 하드웨어 호환성 다양한 텐서 병렬

처리 방법의 성능을 더 자세히 알아보기 위해 GPU 상호 연결이 텐서 병렬 처리 성능에 미치는 영향을 조사하기 위해 시스템 I 및 II에 대한 실험을 수행했습니다. 시스템 I과 시스템 II는 그림 5a와 같이 모든 GPU 쌍 간에 NVLink를 완전히 연결했지만 후자는 그림 5b와 같이 4쌍의 인접한 GPU 사이에 NVLink만 있기 때문에 실험을 위해 선택되었습니다. System I의 통신 대역폭은 그림 13과 같이 GPU 쌍으로 측정하든 GPU 그룹으로 측정하든 상관없이 지속적으로 높습니다. 그러나 통신 대역폭은 184GB/s에서 15GB/s로 크게 떨어집니다 인접한 GPU에만 고성능 NVLink가 있으므로 인접하지 않은 GPU 간에 통신이 발생합니다.

따라서 System II의 GPU 토폴로지는 PCIe를 통해 모든 GPU에서 전체 축소 작업에 의존하는 1D 텐서 병렬 처리에 적합하지 않습니다. 대신 2D와 2.5D는 모든 GPU가 아닌 한 쌍의 GPU 사이에서만 통신합니다. 이렇게 하면 통신의 일부가 인접한 GPU 사이의 높은 NVLink 대역폭을 계속 활용할 수 있습니다.

Machine Translated by Google

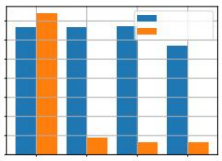
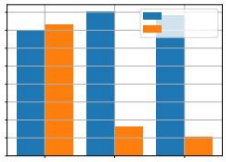





(a) 배치 크기별 GPU 4개 (b) 배치 크기별 GPU 8개 (c) 숨겨진 크기별 GPU 4개 (d) 숨겨진 크기별 GPU 8개

그림 12: 4/8 GPU로 Tensor Parallelism의 메모리 소비에 대한 범위 테스트

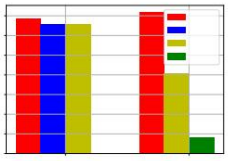
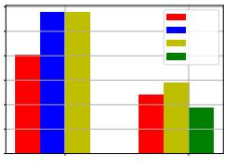
#GPU 모드		#Transformer Layer Hidden 크기		#주의 헤드 배치 크기		처리량(이미지/초)	1D 이상의 속도 향상(%)
4	1D	24	2048년	32	128	5.06	-
	2D				256	6.18	22.1
	2.5D(깊이=1)				256	6.73	33.0
8	1D	24	2048년	32	256	7.46	-
	2.5D(깊이=2)				384	6.57	-11.9
	3D				512	8.38	12.3
16	1D	32	4096	64	64	3.42	-
	2D				256	5.33	55.8
	2.5D(깊이=1)				256	5.46	59.6
32	1D	32	4096	64	128	4.22	-
	2.5D(깊이=2)				256	5.46	50.6
64	1D	32	4096	64	128	4.63	-
	2D				512	12.76	275.5
	2.5D(깊이=4)				512	4.93	6.5
	3D				512	8.63	86.4

표 III: GPU 수가 다른 Tensor Parallelism의 성능

(a) GPU 쌍 간의 통신 대역폭 (b) 집단 통신을 위한 통신 대역폭

그림 13: 시스템 I 및 II의 통신 대역폭(NCCL 대역폭 테스트 도구를 사용하여 125MB 데이터 브로드캐스팅)

(a) 시스템 I (b) 시스템 II

그림 14: 시스템 I 및 II에서 ViT 교육의 처리량

우리는 System I과 II 모두에서 GPU 4개와 GPU 8개에 대해 서로 다른 구성으로 ImageNet-1k 데이터 세트에서 ViT를 교육했습니다. 4개의 GPU에서 ViT 모델에는 숨겨진 크기가 3072인 64개의 트랜스포머 레이어와 48개의 어텐션 헤드가 있습니다. 8개의 GPU에서 숨겨진 크기와 어텐션 헤드의 수는 사용 가능한 메모리가 더 많기 때문에 각각 4096과 64로 조정됩니다.

모델은 메모리 부족 문제가 발생할 때까지 다양한 배치 크기 설정으로 학습됩니다. 이와 같이 각 텐서 병렬 처리 방법에 대해 최상의 처리량을 제시합니다. 그림 14a에서 2D, 2.5D 및 3D 텐서 병렬 처리의 처리량은 GPU 4개와 GPU 8개 모두에서 1D 텐서 병렬 처리와 경쟁할 수 없습니다. 이는 두 가지 이유로 예상됩니다.

첫 번째 이유는 1D 텐서 병렬 처리가 System I에 관련된 모든 GPU와 높은 통신 대역폭을 활용할 수 있기 때문입니다. 두 번째 이유는 2D, 2.5D 및 3D 텐서 병렬 처리가 적은 수의 프로세서로 더 많은 통신량을 가지며 이를 능가할 수 밖에 없다는 것입니다. 프로세서 수가 증가할 때 1D 텐서 병렬 처리.

그러나 실험이 그림 14b에서 시스템 II로 전환되면 1D 텐서 병렬 처리는 4개 및 8개의 GPU에 걸친 집단 통신에서 낮은 통신 대역폭으로 인해 병목 현상이 발생합니다. 한편, 2D 및 2.5D는 4개의 GPU를 사용하는 1D 텐서 병렬 처리보다 40% 더 높은 처리량을 제공할 수 있습니다. 8개의 GPU를 사용하면 2.5D 텐서 병렬 처리가 여전히 1D 텐서 병렬 처리보다 20.6% 더 성능이 뛰어납니다. 3D 텐서 병렬 처리는 낮은 스케일링으로 인해 여전히 1D 텐서 병렬 처리보다 낮은 성능을 제공합니다.

4) 처리량 비교 더 많은 GPU로 텐서

병렬 성능을 테스트하기 위해 우리는 System IV에서 ViT를 교육했습니다. System IV에는 16GB GPU 메모리만 있으므로

Machine Translated by Google

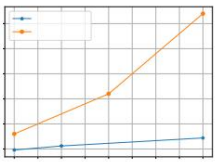
그에 따라 ViT 모델의 구성. 이 모델은 숨겨진 크기가 2048인 24개의 레이어와 4개 및 8개의 GPU에 대한 32개의 어텐션 헤드를 갖도록 설정됩니다. 16개의 GPU부터 이 모델은 숨겨진 크기가 4096인 32개의 레이어와 64개의 어텐션 헤드를 갖도록 설정됩니다.

4~64개의 GPU에 대한 결과는 표 III에 나와 있습니다.
GPU 수가 증가함에 따라 1D 텐서 병렬 처리에 비해 고급 텐서 병렬 처리 속도가 최대 2.76까지 증가하는 것을 관찰할 수 있습니다. 이는 더 많은 프로세서로 확장할 때 고급 텐서 병렬 처리 방법의 통신량이 적기 때문일 수 있습니다. 메모리 효율성 및 낮은 통신량과 함께 2D, 2.5D 및 3D 텐서 병렬 처리는 대규모 분산 교육을 위한 더 나은 옵션입니다.

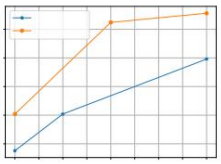
C. 시퀀스 병렬성

이 섹션에서는 시퀀스 병렬 처리를 1D 텐서 병렬 처리와 비교하여 메모리 효율성 및 교육 처리량 측면에서 이점을 보여줍니다. Sequence Parallelism은 활성화에 의해 소비되는 메모리가 모델 데이터에 의해 소비되는 것보다 더 큰 상황을 위해 설계되었으므로 BERT-Base가 실험 모델로 선택되고 Wikipedia 데이터 세트에서 훈련됩니다 [35]. 우리는 시스템 III에서 실험을 수행했습니다. 1D 텐서 병렬 처리에는 어텐션 헤드의 수(12)가 병렬 크기로 나눌 수 있어야 한다는 점에 유의해야 합니다. 6-GPU 실험에서는 각 노드에서 2개의 노드와 3개의 GPU를 사용하는 경우 4, 6 및 12개의 GPU만 사용할 수 있습니다. .

한편, Sequence Parallelism은 어텐션 헤드의 수에 제한을 받지 않으므로 4, 8, 12개의 GPU에서 실험을 수행했습니다.



(a) 최대 배치 크기



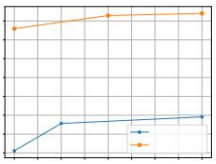
(b) 최대 시퀀스 길이

그림 15: 1D 텐서 병렬 처리에 대한 시퀀스 병렬 처리의 메모리 효율성

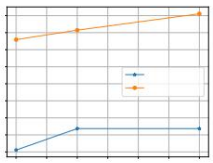
1) 메모리 효율성 BERT 교육
에서 시퀀스 병렬 처리의 메모리 효율성을 설명하기 위해 1D 텐서 병렬 처리와 시퀀스 병렬 처리 모두에서 메모리 부족 문제가 발생할 때까지 배치 크기와 시퀀스 길이를 늘립니다. 시퀀스 길이는 배치 크기 테스트의 경우 512로 고정되고 배치 크기는 시퀀스 길이 테스트의 경우 64로 고정됩니다.

그림 15a에서 볼 수 있듯이 시퀀스 병렬 처리는 1D 텐서 병렬 처리보다 지속적으로 더 큰 배치 크기에 도달할 수 있습니다. 이는 1D 텐서 병렬 처리가 시퀀스 병렬 처리에서 시퀀스 차원을 따라 활성화가 분할되는 중복 활성화에서 메모리 병목 현상이 있기 때문입니다. Sequence Parallelism의 최대 배치 크기는 4.44배입니다.

12개의 GPU가 있는 1D 텐서 병렬 처리의 것입니다. 최대 시퀀스 길이 테스트에서도 동일한 패턴을 관찰할 수 있으며 그림 15b와 같이 메모리 효율성 측면에서 시퀀스 병렬 처리의 우수성을 재확인합니다. Sequence Parallelism의 최대 시퀀스 길이는 BERT-Base 모델의 self-attention 모듈이 시퀀스 길이에 대한 2차 복잡도이므로 최대 시퀀스 길이는 GPU. Linear-complexity self-attention module[36], [37]을 사용하면 Sequence Parallelism은 GPU 수에 따라 최대 시퀀스 길이의 선형 확장을 달성하여 문서 수준의 텍스트 이해를 더 잘 지원할 수 있습니다.



(a) 파이프라인 없음



(b) 파이프라인 포함

그림 16: BERT-Base의 교육 처리량

2) 처리량 비교 Sequence
Parallelism의 훈련 속도를 평가하기 위해 시퀀스 길이 512와 최대 배치 크기로 BERT-Base를 훈련했습니다. 처리량 메트릭은 초당 토큰입니다. 그림 16a에서 볼 수 있듯이 시퀀스 병렬 처리는 지속적으로 더 높은 훈련 처리량을 제공하며 시퀀스 병렬 처리 처리량은 1D 텐서 병렬 처리보다 1.43배 더 높습니다.

디자인 섹션에서 언급했듯이 시퀀스 병렬 처리는 파이프라인 병렬 처리와 자연스럽게 호환되며 파이프라인 단계 간에 추가 통신이 필요하지 않습니다. 우리는 파이프라인 병렬 처리를 통해 훈련을 더욱 확장했습니다. 시퀀스 및 1D 텐서 병렬 처리의 병렬 크기는 4로 고정되며 파이프라인 단계의 수를 1에서 4로 조정합니다.

그림 16b에 표시된 대로 시퀀스 병렬 처리는 4개의 파이프라인 단계를 사용하여 1D 텐서 병렬 처리보다 1.55배 더 빠르게 훈련할 수 있으며, 이는 비파이프라인 결과에 12% 추가 속도를 제공합니다.

D. 샤딩 및 오프로딩

이 섹션에서는 DeepSpeed에 대해 자체 샤딩 및 오프로드 모듈을 평가했습니다. DeepSpeed 3단계를 기준선으로 사용하여 데이터 병렬 교육에서 모델 매개변수, 기울기 및 최적화 상태를 분할했습니다. Colossal-AI와 DeepSpeed 모두 옵티마이저 데이터를 CPU 메모리로 오프로드합니다.

우리는 System II의 Wikipedia 데이터 세트에서 100억 개의 매개변수로 GPT-2 모델을 교육했습니다. 데이터 병렬 교육을 1 GPU에서 8 GPU로 확장하고 배치 크기가 4인 교육 처리량을 관찰합니다.

섹션 III-B에서 논의된 추가 최적화를 통해 Colossal-AI는 그림 17과 같이 GPU 8개에서 DeepSpeed보다 2.33배 더 빠릅니다.

Machine Translated by Google

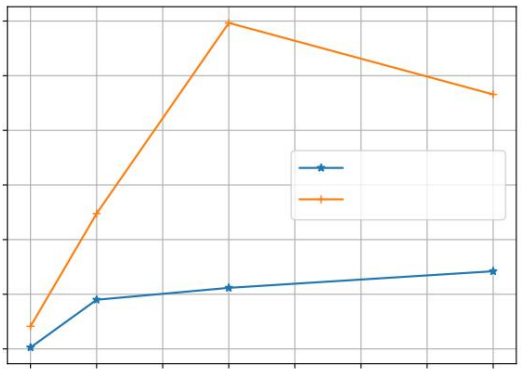


그림 17: 샤딩 및 오프로딩을 사용한 GPT 교육의 처리량

VI. 결론

이 작업에서 우리는 대규모 분산 훈련을 위해 방대한 수의 고급 가속 기술을 하나의 통합 시스템으로 통합한 Colossal-AI를 설계하고 구현했습니다. Colossal-AI는 다양한 병렬 방식의 쉬운 조합을 지원하고 개발자와 사용자가 쉽게 확장할 수 있는 유연한 시스템 설계와 함께 제공됩니다.

또한 가속 기술은 다양한 하드웨어 조건에서 강력한 성능을 제공하며 기본 시스템에 비해 우수한 성능을 제공합니다. 실험에서 Megatron-LM 및 DeepSpeed에 비해 Colossal-AI의 장점을 입증했습니다. 2.5D 및 3D 텐서 병렬 처리는 2-layer MLP의 숨겨진 크기를 16384로 확장할 때 Megatron-LM 1D 텐서 병렬 처리에 비해 최대 62% 및 74.2% 메모리 사용량을 절약할 수 있습니다. 64 GPU로 확장할 때 2D 텐서 병렬 처리는 Megatron을 능가할 수 있습니다. -LM 1D 텐서 병렬성 276%. BERT Base와 같이 많은 양의 활성화가 있는 중간 크기 모델의 경우 시퀀스 병렬 처리가 1D 텐서 병렬 처리보다 1.43배 더 높은 훈련 처리량을 제공할 수 있습니다. 재설계된 샤딩 및 오프로딩 모듈은 GPT-2의 DeepSpeed보다 2.33배 빠릅니다.

참조

[1] J. Devlin, M.-W. Chang, K. Lee 및 K. Toutanova, "Bert: 언어 이해를 위한 깊은 양방향 변환기 사전 훈련", arXiv preprint arXiv:1810.04805, 2018.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly 외, "이미지는 16x16 단어 가치: 대규모 이미지 인식을 위한 변환기," arXiv preprint arXiv:2010.11929, 2020.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "언어 모델은 감독되지 않은 멀티태스크 학습자", 2019.

[4] TB Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell 등, "언어 모델은 거의 없습니다. -샷 학습자," arXiv preprint arXiv:2005.14165, 2020.

[5] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, J. Tang, "All nlp task are generation tasks: A general pretraining framework," arXiv preprint arXiv: 2103.10360, 2021.

[6] J. Duchi, E. Hazan 및 Y. Singer, "온라인 학습 및 확률적 최적화를 위한 적응형 하위 그라데이션 방법." 기계 학습 연구 저널, vol. 12, 아니. 2011년 7월 7일.

[7] DP Kingma 및 J. Ba, "Adam: 확률적 최적화를 위한 방법", arXiv 사전 인쇄 arXiv:1412.6980, 2014.

[8] K. He, X. Zhang, S. Ren, J. Sun, "이미지 인식을 위한 심층 잔차 학습", 2016 IEEE 컴퓨터 비전 및 패턴 인식 회의(CVPR), 2016, pp. 770–778 .

[9] T. Chen, B. Xu, C. Zhang 및 C. Guestrin, "하위 선형 메모리 비용으로 심층 네트워크 교육", 2016. [온라인]. 이용 가능: https://arxiv.org/abs/1604.06174 [10] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, QV Le , Y. 우, 및 z. Chen, "Gpipe: Pipeline Parallelism을 사용한 거대한 신경망의 효율적인 훈련", Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc, E. Fox, and R Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [온라인]. 이용 가능: https:// proceedings.neurips.cc/paper/ 2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf [11] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, NR Devanur, GR Ganger, PB Gibbons , M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," 운영 체제 원칙에 관한 27회 ACM 심포지엄 절차, ser. SOSP '19. 미국 뉴욕주 뉴욕: 컴퓨팅 기계 협회, 2019, p. 1–15. [온라인]. 이용 가능: https://doi.org/10.1145/3341301.3359646 [12] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catan zaro, "Megatron-lm: Training multi- 모델 병렬 처리를 사용하는 10억 매개변수 언어 모델", arXiv preprint arXiv:1909.08053, 2019.

[13] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer 및 Z. Chen, "{GS}hard: Scaling giant models with 조건부 계산 및 자동 샤딩," in International Conference on Learning Representations, 2021. [온라인]. 이용 가능: https://openreview.net/forum?id=qrwe7XHTmYb [14] M. Baines, S. Bhosale, V. Caggiano, N. Goyal, S. Goyal, M. Ott, B. Lefaudeux, V. Liptchinsky, M. Rabbat, S. Sheiffer, A. Sridhar 및 M. Xu, "Fairscale: 고성능 및 대규모 훈련을 위한 범용 모듈식 pytorch 라이브러리", https://github.com/facebookresearch/fairscale , 2021 .

[15] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee 및 M. Zaharia, "megatron-lm을 사용하여 gpu 클러스터에서 효율적인 대규모 언어 모델 교육", 고성능 컴퓨팅, 네트워킹, 스토리지 및 분석을 위한 국제 회의 절차, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [온라인]. 이용 가능: https://doi.org/10.1145/3458817.3476209

[16] 제26회 ACM SIGKDD International Conference on Knowledge Discovery & 데이터 마이닝, 2020, pp. 3505–3506.

[17] D. Paperno, G. Kruszewski, A. Lazaridou, Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernandez, "람바다 데이터 세트: 광범위한 데이터 예측이 필요한 단어 예측 담화 맥락", 2016년 6월, pp. 1525–1534.

[18] D. Kingma 및 J. Ba, "Adam: 확률적 최적화 방법", 학습 표현에 관한 국제 회의, 2014년 12월.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, AN 고메즈, L. 유. Kaiser 및 I. Polosukhin, "Attention is all you need", 신경 정보 처리 시스템의 발전, I. Guyon, UV Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan 및 R. Garnett , Eds., vol. 30. Curran Associates, Inc., 2017. [온라인]. 이용 가능: https://proceedings.neurips.cc/paper/ 2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf [20] Q. Xu, S. Li, C. Gong 및 Y. You, "훈련을 위한 효율적인 2D 방법 초대형 딥 러닝 모델", arXiv preprint arXiv:2104.05343, 2021.

[21] B. Wang, Q. Xu, Z. Bian 및 Y. You, "2.5차원 분산 모델 교육", arXiv preprint arXiv:2105.14500, 2021.

[22] Z. Bian, Q. Xu, B. Wang 및 Y. You, "거대한 신경망을 위한 분산 훈련에서 병렬 처리 극대화", arXiv preprint arXiv:2105.14450, 2021.

[23] LE Cannon, Kalman 필터 알고리즘을 구현하는 셀룰러 컴퓨터. 몬타나 주립대학교, 1969.

Machine Translated by Google

[24] J. Berntsen, "하이퍼 큐브의 효율적인 통신 행렬 곱셈" 병렬 컴퓨팅, vol. 12, 아니. 3, pp. 335–342, 1989.

[25] RA van de Geijn 및 J. Watts, "Summa: 확장 가능한 범용 행렬 곱셈 알고리즘" USA, Tech. 의원, 1995.

[26] E. Solomonik 및 J. Demmel, "통신 최적화 병렬 2.5d 행렬 곱셈 및 lu 분해 알고리즘", Euro-Par, 2011.

[27] RC Agarwal, SM Balle, FG Gustavson, M. Joshi 및 P. Palkar, "병렬 행렬 곱셈에 대한 3차원 접근", IBM Journal of Research and Development, vol. 39, 아니. 5, pp. 575–582, 1995.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf , E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai 및 S. Chintala, "Pytorch: 명령형 스타일, 고성능 딥 러닝 라이브러리" in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [온라인]. 이용 가능: http://papers.neurips.cc/paper/9015-pytorch an-imperative-style-high-performance-deep-learning-library.pdf [29] D. Narayanan, A. Harlap, A. Phanishayee, V . Seshadri, NR Devanur, GR Ganger, PB Gibbons 및 M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," 운영 체제 원리에 관한 27회 ACM 심포지엄 절차, ser. SOSP '19. 미국 뉴욕주 뉴욕: 컴퓨팅 기계 협회, 2019, p. 1–15.

[온라인]. 이용 가능: https://doi.org/10.1145/3341301.3359646 [30] Y. Huang, Y. Cheng, A. Bapna, O. Firat, MX Chen, D. Chen, H. Lee, J. Ngiam, QV Le, Y. Wu, Z. Chen, GPipe: 파이프라인 병렬 처리를 사용한 거대한 신경망의 효율적인 훈련. Red Hook, NY, USA: Curran Associates Inc., 2019.

[31] S. Li 및 T. Hoefler, "Chimera: 양방향 파이프라인으로 대규모 신경망을 효율적으로 교육", 고성능 컴퓨팅, 네트워킹, 스토리지 및 분석을 위한 국제 회의 절차, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [온라인]. 이용 가능: https://doi.org/10.1145/3458817.3476145

[32] S. Li, F. Xue, Y. Li, Y. You, "시퀀스 병렬성: 시스템 관점에서 본 긴 시퀀스 트레이닝", 2021. [온라인]. 이용 가능: https://arxiv.org/abs/2105.13120 [33] J. Fang, Y. Yu, Z. Zhu, S. Li, Y. You 및 J. Zhou, "Patrickstar: 사전 훈련된 청크 기반 메모리 관리를 통한 모델,"

2021. [온라인]. 이용 가능: https://arxiv.org/abs/2108.05818 [34] J. Ren, S. Rajbhandari, RY Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, Y. He, "제로 -오프로드: 10억 규모의 모델 교육 민주화," 2021년.

[35] W. 재단. 위키미디어 다운로드. [온라인]. 가능: https://dumps.wikimedia.org

[36] S. Wang, B. Li, M. Khabsa, H. Fang 및 H. Ma, "Linformer: 선형 복잡성을 통한 자기 주의", arXiv preprint arXiv:2006.04768, 2020.

[37] M. Zaheer, G. Guruganesh, KA Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, A. Ahmed, "빅 버드 : 더 긴 시퀀스를 위한 변환기", 신경 정보 처리 시스템의 발전, H. Larochelle, M. Ranzato, R. Hadsell, MF Balcan 및 H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 283–17 297. [온라인]. 이용 가능 : https://proceedings.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf