컨테이너 환경에서 텐서플로의 GPU 메모리 사용 방식에 따른 학습 작업의 성능 분석†

강지훈[1], 길준민[2,‡]
[1]고려대학교 4단계 BK21 컴퓨터학교육연구단
[2]대구가톨릭대학교 컴퓨터소프트웨어학부
k2j23h@korea.ac.kr, jmgil@cu.ac.kr

Analyzing the performance of training tasks based on GPU memory use manner of TensorFlow in Container environments

Jihun Kang[1], Joon-Min Gil[2,‡]
[1]BK21 FOUR R&E Center for Computer Science and Engineering, Korea University
[2]School of Computer Software Engineering, Daegu Catholic University

요 약

인공지능의 학습 작업은 연산량이 많아 고성능 연산 장치인 GPU(Graphics Processing Unit)를 필요로 하며, GPU 장치의 성능은 학습 작업의 실행 성능에 직접적으로 영향을 미치는 요소 중 하나로 작용한다. 인공지능 작업을 처리하기 위해 많이 사용되는 텐서플로의 경우 GPU를 사용해 연산을 수행할 때 기본적으로 거의 모든 GPU 메모리 영역을 단일 학습 작업이 점유하도록 GPU 메모리를 관리한다. 이 방법은 컴퓨팅 자원 중 확장성이 가장 낮은 GPU 메모리의 단편화를 방지하기 위해 사용되는 방법이지만, 하나의 학습 작업이 GPU를 점유하게 되면, 실제 GPU 메모리 사용량과 상관없이 다른 프로세스는 GPU를 사용할 수 없는 문제를 유발한다. 특히, 전이학습, 소규모 학습과 같이 상대적으로 작업 규모가 작은 경우에는 전체 GPU 메모리 용량 중 대부분의 영역이 낭비된다. 본 논문에서는 컨테이너 환경에서 텐서플로의 기본 GPU 메모리 사용 방식으로 인해 다수의 학습 작업을 동시 실행하는 것이 불가능한 문제를 확인하고 GPU 메모리 사용량을 제한한 경우와 하지 않은 경우에 실제 GPU 메모리 사용량과 학습 작업의 실행 시간에 대한 성능 비교를 통해 GPU 메모리의 단편화 방지가 성능에 유의미한 요소인지 검증한다.

1. 서론

일반적으로 학습 작업은 연산량이 많다는 특성으로 인해 GPU와 같은 고성능 연산 장치를 필수적으로 요구한다. 수천 개의 연산 코어가 존재하는 GPU의 고성능 연산을 통해 학습 작업을 상대적으로 빠르게 완료할 수 있으며, 이는 생산성 향상과 비용 절감으로 이어진다.

인공지능 학습과 추론 작업을 구현할 때 대표적으로 사용되는 라이브러리 중 하나인 텐서플로[1]의 경우에도 학습 및 추론 작업을 수행할 때 GPU를 사용할 수 있도록 지원한다. GPU를 사용한 연산은 일반적으로 CUDA[2]나 OpenCL[3]과 같은 별도의 GPGPU 라이브러리를 사용해야 하지만, 텐서플로의 경우 가용 연산 장치에 GPU가 존재한다면 학습 코드를 수정할 필요 없이 GPU를 사용할 수 있도록 투명성을 제공한다.

텐서플로의 경우 프로세스가 GPU 장치를 사용할 때 GPU 메모리의 단편화를 최소화할 수 있도록 단일 학습 작업에게 가용 GPU 메모리의 거의 모든 영역을 할당한다[4]. 이는 CPU나 메모리와 같은 일반적인 컴퓨팅 장치에 비해 상대적으로 확장성이 낮고 용량이 낮은 GPU 메모리를 보다 효율적으로 사용하기 위해 텐서플로 라이브러리에서 사용하는 방식이다. 이로 인해, 학습 작업이 실행되면 가장 먼저 GPU에 접근하는 프로세스가 GPU 메모리의 거의 모든 영역을 할당받으며, 그 뒤에 실행된 프로세스가 GPU를 사용하기 위해서는 먼저 실행된 프로세스가 작업을 끝내고 GPU를 반환해야 한다. 이러한 방식은 GPU 메모리의 단편화를 방지할 수는 있지만 자원 활용률 측면에서는 비효율적으로 작동한다.

† 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.NRF-2022R1A2C1092934)
‡ 교신저자

GPU의 경우 연산 코어와 GPU 자체의 온 보드 메모리가 존재한다. 또한, GPU 코어는 연산 시 사용할 데이터를 GPU 메모리에서만 가져올 수 있다. 이로 인해 텐서플로와 같은 학습 작업뿐만 아니라 일반적인 GPGPU 프로그래밍 모델에서도 GPU 코어를 사용해서 작업을 처리하기 위해서는 먼저 연산에 사용할 데이터를 GPU 메모리에 적재해야지만 GPU 코어를 사용한 연산을 시작할 수 있다.

단일 프로세스가 GPU 메모리의 거의 모든 영역을 점유하는 텐서플로의 GPU 메모리 사용 방식은 GPU 메모리의 낭비뿐만 아니라 GPU 코어의 낭비도 발생시킨다. 가장 먼저 GPU를 점유한 학습 작업이 GPU 메모리 영역 전부를 점유하고 있기 때문에 다른 프로세스는 GPU 메모리를 할당받지 못하고 이로 인해 GPU 메모리에 데이터가 준비되지 않은 작업들은 GPU 코어를 사용하지 못한다.

텐서플로 라이브러리에서도 프로세스가 요구하는 수준의 GPU 메모리만 사용하도록 지원하는 코드 수준의 기능이 존재한다. 하지만, 이 기능은 학습 작업의 코드 작성 시 포함해야 하는 프로그래밍 함수 기능이며, 코드 수준에서 추가해야 사용할 수 있다.

본 논문에서는 컨테이너 환경[5]에서 GPU를 사용한 학습 작업을 수행할 때 GPU 메모리 단편화 방지를 위해 단일 학습 작업이 GPU 메모리 거의 모든 영역을 점유하는 기본 방식과 코드 추가를 통해 GPU 메모리 사용량을 실제 필요한 용량만큼으로 제한하는 방식과의 자원 사용량과 성능의 비교를 통해 GPU 메모리 단편화가 학습 작업의 성능에 미치는 영향을 확인한다.

2. GPU 메모리 사용 방식에 따른 학습 성능

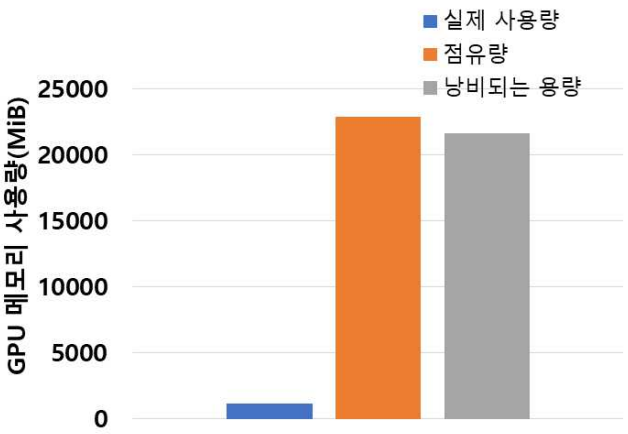
<표 1> 실험 환경

분류	성능
CPU	i9-10920X(3.5GHz, 12 Cores, 24 Threads)
Memory	128GB
GPU	RTX 3090(24GB Memory)
OS	Ubuntu 18.04
Docker	nvidia Docker2[4]

이번 장에서는 실험을 통해 GPU 메모리의 단편화 방지를 위해 단일 프로세스가 GPU 메모리의 거의 모든 영역을 독점하는 기본 사용 방식과 사용량을 학습 작업에 필요한 GPU 메모리 용량만큼만 할당하여 GPU 메모리 할당량을 제한하는 방식 사이의 성능을 비교하여 단편화 방지로 인한 성능 차이가 유의미한 요소인지 확인한다. 실험 환경은 <표 1>과 같다.

이번장의 실험에서는 최대 25개의 컨테이너를 사용하며, 각 컨테이너는 MNIST 데이터 셋[6]을 사용한 학습 작업을 수행하며, 배치사이즈는 32, 에포크는 5로 설정하여 학습 작업을 실행한다. 실험을 통해 단일 학습 작업이 가용 GPU 메모리 대부분을 독점하는 기본 방식과, 학습 작업에서 필요로 하는 실제 GPU 메모리 요구량만큼만 할당받는 제한 방식 사이의 자원 사용량과 성능 차이를 비교한다. 학습 작업의 GPU 메모리 제한은 학습 작업의 코드에 *tf.config.experimental.set_memory_growth*를 추가하여 학습 작업이 GPU 메모리 대부분을 점유하지 못하도록 방지하고 실제 학습 작업에 필요한 용량만큼의 GPU 메모리를 할당받도록 한다.

첫 번째 실험은 텐서플로의 GPU 메모리 사용에 대한 기본 방식을 활용할 때와 제한 방식을 사용했을 때의 자원 사용량을 비교하여 낭비되는 GPU 자원의 용량을 확인한다. 실험 결과는 (그림 1)에서 보여준다.



(그림 1) GPU 메모리 사용 방식에 따른 자원 낭비 규모

(그림 1)에서 보여주는 것과 같이 실험에서 사용한 학습 작업은 실제로 1,205 MiB의 GPU 메모리를 사용한다. 하지만 텐서플로의 GPU 메모리 기본 사용 방식을 사용하면 단일 학습 작업이 실험에서 사용한 GPU 장치의 메모리 전체 용량인 24,265 MiB 중에 약 94%인 22,885 MiB를 점유하며, 학습 작업에서 필요로 하는 실제 GPU 메모리 사용량 보다

약 19배 많은 용량을 할당 받게 된다.

단일 학습 작업이 GPU 메모리의 거의 모든 영역을 점유하는 특성은 GPU 메모리를 점유 중인 학습 작업이 작업을 완료하고 GPU 메모리를 해제하기 전 까지 다른 프로세스가 사용할 수 없게 되는 문제를 발생 시킨다. 이로 인해, 결과적으로 텐서 플로의 GPU 메모리 기본 사용 방식을 사용하게 되면 학습 작업의 GPU 메모리 사용량과 상관없이 동시에 하나의 학습 작업만 실행할 수 있게 된다. 또한, 본 논문의 실험에서 사용한 학습 작업의 경우 학습 작업을 수행 하는 동안 GPU 코어는 4%만 사용되며, 매우 작은 수준의 GPU 코어를 사용한다. GPU 코어의 가용 용량만 보면 실험에서 사용한 학습 작업을 약 24개 더 실행할 수 있을 정도의 가용 GPU 코어가 존재하지만 하나의 학습 작업이 GPU 메모리 거의 대부분을 점유하고 있기 때문에 다른 학습 작업이 실행될 수 없어서 가용 GPU 코어도 활용되지 못하고 낭비된다.

다음 실험은 텐서플로의 GPU 메모리 기본 사용 방식과 제한 방식간의 학습 작업의 실행 시간을 측정하여 비교한다. 실험에서는 앞서 수행한 실험과 동일한 학습 작업을 수행하며, epoch만 10회로 증가시켰다. 텐서플로의 GPU 메모리 기본 사용 방식과 GPU 메모리 사용량을 제한한 제한 사용 방식을 사용했을 때의 학습 작업 실행 시간을 측정한다. 실험 결과는 (그림 2)에서 보여준다.

(그림 2) GPU 메모리 사용 방식에 따른 학습 시간

(그림 2)에서 보여주는 것과 같이 GPU 메모리 사용 방식에 따른 학습 작업의 실행 시간에 대한 성능 차이는 크지 않다. GPU 메모리 사용 방식에 따른 성능은 약 0.5초의 실행시간 차이만 발생하였으며, 이는 0.6%의 성능 차이로 전체 성능에 매우 작은 부분을 차지한다. 이는 GPU 메모리 사용 방식의 차이에 따라 유의미한 성능 차이가 발생하지 않는다는 것을 뜻하며, 이를 통해 GPU 메모리 단편화 방지로 인한 영향이 학습 작업의 성능에 큰 영향을 끼치지 않는다는 것을 확인할 수 있다.

3. 결론 및 향후 연구

이전 장에서 수행한 실험 결과에서 보여주는 것과 같이 텐서 플로가 GPU를 사용할 때 단일 작업이 GPU 메모리 대부분을 점유하는 방식은 GPU 메모리 낭비가 매우 심했다. 특히 실험에서 사용한 실행 규모가 상대적으로 작은 학습의 경우 할당받은 GPU 메모리 중 실제로는 4%만 사용해서 학습 작업을 수행한다. 또한, GPU 메모리 점유로 인해 다른 학습 작업을 추가 실행 하지 못하며, 이는 자원 낭비로 이어진다. 또한, 본 논문의 실험에서 사용한 MNIST 데이터 셋과 같이 학습 데이터 규모가 상대적으로 작은 학습 작업의 경우 GPU 메모리 대부분을 점유한 방식과 요구량만큼만 점유하는 방식 사이에 학습 작업의 실행 시간도 0.6%의 매우 작은 차이만 발생했다.

본 논문의 향후 연구는 전이학습이나 소규모 학습과 같이 학습 규모가 상대적으로 작은 학습 작업을 운용하는 환경에서 단일 GPU에 최대한 많은 학습 작업을 실행하고 GPU 자원을 효율적으로 사용하기 위한 학습 작업 및 GPU 자원 관리 기법에 대한 연구를 수행할 계획이다.

참고문헌

[1] TensorFlow, https://www.tensorflow.org/?hl=ko
[2] CUDA, https://docs.nvidia.com/cuda/cuda-c-programming-guide/
[3] OpenCL, https://www.khronos.org/opencl/
[4] TensorFlow GPU, https://www.tensorflow.org/guide/gpu?hl=ko
[5] Docker, https://www.docker.com/
[6] MNIST dataset, http://yann.lecun.com/exdb/mnist/