ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 31, No. 1, pp. 25-33, 2025. 1
https://doi.org/10.5626/KTCP.2025.31.1.25

이기종 클러스터 환경의 딥러닝 응용을 위한 GPU 공유 기법 분석

(An Analysis of GPU Sharing Methods for Deep Learning Applications in a Heterogeneous Cluster Environment)

하 지 원 † 김 서 영 †† 테오도라 아두푸 †††
(Jiwon Ha) (Seoyoung Kim) (Theodora Adufu)
엄 현 상 †††† 김 윤 희 †††††
(Hyeonsang Eom) (Yoonhee Kim)

요 약 이기종 GPU 클러스터 환경에서 딥러닝 계산 작업들을 실행할 때 발생할 수 있는 활용률 저하 문제를 GPU 공유 기술을 통해 해결하고 최적의 성능을 내기 위한 연구가 진행 중이다. 그러나 time-slicing, Multi-Process Service(MPS), Multi-Instance GPU(MIG) 공유 기술을 딥러닝 응용의 특성에 맞춰 적절히 사용하는 연구는 제시되지 않았다. 본 논문은 딥러닝 모델 및 파라미터에 따른 작업의 실행 규모, 자원 민감도, 자원 활용 특징을 분석하여 time-slicing, MPS, MIG 공유 기술에 대한 최적의 작업 배치 기준을 제안한다. 서로 다른 공유 기술을 지원하는 이기종 GPU 클러스터 환경에서 최적의 작업 배치를 수행하고 실험을 통해 실행 시간과 처리량이 향상됨을 보였다.

키워드: 이기종 GPU, 응용 프로파일링, 쿠버네티스, MPS, MIG, time-slicing

Abstract GPU sharing technologies are being implemented to address the low resource utilization often seen in deep learning applications and to enhance performance in heterogeneous GPU clusters. However, there has been considerably less research on the effective use of time-slicing, Multi-Process Service (MPS), and Multi-Instance GPU (MIG) sharing technologies in relation to the specific characteristics of deep learning applications across different GPU architectures. This paper proposes optimal task placement criteria for these GPU sharing technologies by analyzing the execution scale, resource sensitivity, and resource utilization of tasks based on various deep learning models and parameters. In a heterogeneous GPU cluster environment that supports multiple sharing technologies, this study tested these optimal task placement criteria, resulting in reduced execution time and improved throughput.

Keywords: heterogeneous GPUs, application profiling, Kubernetes, MPS, MIG, time-slicing

· We appreciate the high-performance GPU computing support of HPC-AI Open Infrastructure via GIST SCENT.
· 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2021R1A2C1003379)

† 비 회 원 : 서울대학교 컴퓨터공학부 학생
jwh0245@snu.ac.kr
†† 비 회 원 : 숙명여자대학교 소프트웨어학부 연구원
sssyyy77@gmail.com
††† 비 회 원 : 숙명여자대학교 소프트웨어학부 학생
theoadufu@sookmyung.ac.kr
†††† 비 회 원 : 서울대학교 컴퓨터공학부 교수
hseom@snu.ac.kr
††††† 종신회원 : 숙명여자대학교 소프트웨어학부 교수
(Sookmyung Women's Univ.)
yulan@sookmyung.ac.kr
(Corresponding author임)

논문접수 : 2024년 6월 19일
(Received 19 June 2024)
논문수정 : 2024년 10월 29일
(Revised 29 October 2024)
심사완료 : 2024년 11월 2일
(Accepted 2 November 2024)

정보과학회 컴퓨팅의 실제 논문지 제31권 제1호(2025. 1)

1. 서 론

머신러닝(Machine Learning)이 보편화됨에 따라 관련 응용 계산 요구를 만족시키기 위해 데이터센터 내에서의 하드웨어 가속기인 GPU의 수요가 증가하고 있는 추세이다. 동시에 GPU 성능의 빠른 발전으로 새로운 아키텍처의 고성능 GPU들과 해당 아키텍처 상에서 적용할 수 있는 최신 기술들이 함께 개발되었다. 따라서 Google[1], Meta[2], AWS[3], NVIDIA[4]를 포함하는 대다수의 데이터 센터는 이러한 최신 기술을 사용할 수 있는 고성능 GPU 자원들과 기존 자원들이 다양하게 공존하는 이기종(heterogeneous) 환경이다. 예를 들어 Google Cloud Platform의 경우, A100, V100, P100, K80, T4, P4, L4의 7가지 종류의 GPU들이 공존하는 이기종 환경이다[5]. 이러한 이기종 GPU 클러스터 환경에서 딥러닝 계산 작업들을 실행할 때 최적의 성능을 내기 위한 연구가 활발히 진행되고 있다[6,7].

한편, 이러한 딥러닝 계산 작업은 동적인 실행 패턴을 가지며, 이로 인해 일부 구간을 제외한 대부분의 구간에서는 낮은 GPU 자원 활용률을 보이는 GPU 자원 활용률 저하(underutilization) 문제가 발생한다. 이를 위해 Time-slicing[8], Multi-Process Service(MPS)[9], Multi-Instance GPU(MIG)[10] 등 다양한 공유 기술들이 제안되었다. 하지만 이러한 기술들이 데이터센터의 모든 GPU 자원에 적용 가능한 것은 아니다. MPS의 경우 Volta 이전 아키텍처에는 제한적인 기능을 제공하므로 실질적으로 Volta 아키텍처부터 사용 가능하며, MIG의 경우 비교적 최신 아키텍처인 Ampere, Hopper GPU에서만 지원된다. 즉, Volta 버전 이전의 가속기들의 경우 상당히 좋은 성능을 제공함에도 불구하고 머신러닝 및 딥러닝 작업을 수행할 때 상대적으로 활용률이 낮다. 기존 자원을 잘 활용하기 위해서는 자원 민감도(sensitivity)가 낮아 자원의 성능에 덜 영향을 받는 응용을 배치함으로써 활용도를 높일 수 있다. 자원 민감도는 응용의 모델 및 파라미터 조합에 따라 다양하며 응용 분석을 통해 파악한다. 본 논문에서는 딥러닝 응용의 모델 및 파라미터 조합을 고려하여 작업의 메모리 사용량, Streaming Multiprocessor(SM) 활용률, 자원 민감도 및 자원 활용 특징을 분석한다. 이어서, 공유 기술에 따른 최적의 응용 배치 및 조합을 실험을 통해 확인한다. 분석 결과를 통해 각 자원과 GPU 공유 방식에 최적화된 작업 배치를 수행하여 성능 평가를 통해 효율성을 증명한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서 본 논문과 관련된 연구들과 배경 지식에 대해 설명하고, 3장에서는 딥러닝 응용 분석하고 특징을 설명한다. 4장에서는 3장의 분석을 기반으로 실제 GPU 자원들 및 공유 기술의 조합에 따른 성능 변화를 보이고, 본 논문에서 제안한 방법이 성능 향상을 이끌 수 있음을 증명한다. 5장에서는 결론을 도출하고 향후 연구 계획을 설명한다.

2. 관련 연구 및 연구 동기

본 장에서는 관련된 연구를 소개하고 본 연구의 필요성을 제시한다. 이어 GPU 자원 활용률 저하 문제, 공유 기술들을 설명 및 비교한다.

2.1 관련 연구

Schedtune[6]은 다양한 DL 작업이 GPU 간에 어떻게 다른 동작을 보이는지를 조사하고 GPU 간 평균 추론 시간에 큰 차이가 있음을 보인다. 작업에 할당된 GPU가 작업의 완료 시간에 영향을 미쳐 클러스터 환경에서 전체 작업 처리량과 대기 시간에 영향을 미칠 수 있음을 증명한다. 그러나 이 논문은 GPU 성능에 영향을 줄 수 있는 GPU 특성을 식별하는 데만 초점을 맞추고 있으며, 응용 특성이 이기종 GPU 성능에 미치는 영향은 분석하지 않는다. 해당 논문은 Volcano[11]를 사용하여 딥러닝 응용에 대한 메모리를 추정하고 할당하므로, MIG 및 MPS와 같은 공유 기술을 고려하지 않는다.

MLaaS[7]는 이기종 클러스터에서 작업을 스케줄링하는 데서 발생하는 GPU 활용률 저하, 특정 GPU에 대한 높은 수요, 긴 대기 시간, 이기종 머신 간의 불균형한 부하 등의 문제를 제기한다. 또한 자원 할당 및 작업 스케줄링을 지원하기 위해 이질적인 GPU에서 응용의 특성을 분석한다. 그러나 MLaaS가 제안한 시뮬레이션은 단일 GPU에서의 동시 실행되는 작업을 고려하지 않는다.

2.2 연구 동기

GPU 자원의 활용률 저하 문제는 대다수의 딥러닝 응용들이 할당받은 GPU 자원을 효율적으로 활용하지 못

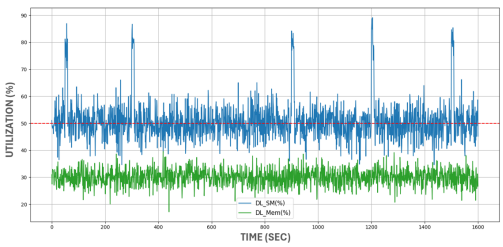


그림 1. GoogLeNet의 메모리 및 SM 활용률

Fig. 1 Memory and SM utilization during GoogLeNet execution

표 1 주요 공유 기술들의 특징 비교
Table 1 Comparison of sharing technologies

Features	Time-slicing	MPS	MIG
Available architectures	All	Volta and above	Ampere and above
Workload isolation	×	△	○
Memory isolation	×	△	○

하는 문제를 의미한다. 그림 1은 NVIDIA A30 PCIe 머신에서 딥러닝 응용인 GoogLeNet의 시간에 따른 메모리(초록색)와 SM(파란색) 활용률을 보인다. 메모리 활용률의 경우 대체로 25-40%의 낮은 활용률을 보이며, SM 활용률의 경우 일부 구간에서 80-90%의 활용률을 보이나 대부분의 구간에서는 40-60%의 낮은 활용률을 보인다. 이처럼 딥러닝 응용의 동적인 자원 활용 패턴에 의해 심각한 자원 활용 저하 문제가 발생한다.

쿠버네티스[12] 환경에서의 GPU 공유 기술로는 time-slicing, MPS, MIG가 활용되고 있으며, 이들의 특징은 표 1에서 비교한다. Time-slicing은 가장 간단한 GPU 시간 공유(temporal sharing) 방식으로 동시에 여러 CUDA 프로세스를 실행함으로써 Pod들이 GPU를 초과 구독(oversubscription)하도록 허용한다. 모든 GPU에서 지원되며 최대 파티션에 제한이 없다는 장점이 존재하지만, 메모리 격리 및 할당 제한이 없어 오류 발생률이 높으며, 빈번한 컨텍스트 전환(context switching)으로 인해 상당한 오버헤드가 발생한다.

MPS는 SM에 대한 완전한 격리를 제공하며, time-slicing에 발생했던 오버헤드를 제거한다. 하지만 Volta 이전 아키텍처의 자원에는 한정적인 기능을 제공하기 때문에 실질적으로는 Volta 아키텍처부터 사용 가능하며, 완전한 메모리 및 오류 격리를 제공하지 못하므로 동일 GPU 상에 실행되는 작업들 사이에 서로 성능 간섭이 발생할 수 있다.

MIG의 경우 최신 아키텍처인 Ampere, Hopper 아키텍처의 GPU에서만 지원되는 기술이지만, 하드웨어 레벨의 자원 격리를 지원하여 GPU를 각각 독립된 메모리, 캐시, 계산 코어를 가진 여러 개의 독립된 인스턴스로 분할한다. 작업 간의 강력한 격리를 보장함으로써 동시 실행 시 간섭을 최소화할 수 있다. 일부 GPU들의 경우 높은 계산 성능을 제공함에도 불구하고 공유 기술의 제약으로 인해 머신러닝 및 딥러닝 작업들을 수행할 때에 자원 활용률이 떨어지는 문제가 있다.

3. 딥러닝 응용 분석

GPU 공유 기술을 통해 GPU 자원의 활용도 및 에너지 효율을 극대화하기 위해서는 워크로드를 구성하는 응용의 특징 분석 및 특성화(characterization)가 필요하다[13]. 따라서 본 장에서는 실험 대상으로 사용될 딥러닝 응용들을 분석한다. 쿠버네티스 환경에서 주로 사용하는 공유 기술들(time-slicing, MPS, MIG)은 워크로드 격리, 메모리 격리 수준 등에서 서로 다른 특징을 가지므로, 동일한 응용을 실행하더라도 어떤 공유 기술을 통해 GPU 자원을 공유하는가에 따라 동시 실행 성능이 달라질 수 있다. 따라서 각 응용의 특징을 분석하여 최적의 성능을 낼 수 있는 공유 기술을 선택하는 것이 중요하다.

표 2 대상 딥러닝 응용 모델 및 종류
Table 2 Target deep learning application model types

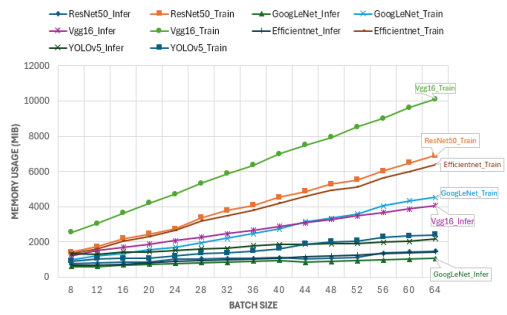
Deep learning model	Type
ResNet50[14], VGG16[15], GoogLeNet[16], EfficientNet[17]	Image classification
YOLOv5[18]	Image recognition

3.1절에서는 각 공유 기술과 응용의 실행 규모 간의 적합성을 파악하기 위해 모델 및 파라미터 조합에 따른 응용의 실행 규모를 분석한다. 이어 3.2절에서는 서로 다른 성능의 GPU 자원에 대한 응용들의 자원 민감도를 관찰하며, 3.3절에서는 최적의 동시 실행 조합을 찾기 위해 응용들을 계산 집약 응용과 메모리 집약 응용으로 분류한다. 표 2는 본 실험에서 사용된 딥러닝 응용 모델들의 정보이다. 해당 딥러닝 모델들은 주로 이미지 분류와 객체 인식에 널리 사용되는 대표적인 Convolutional Neural Network(CNN) 모델이다.

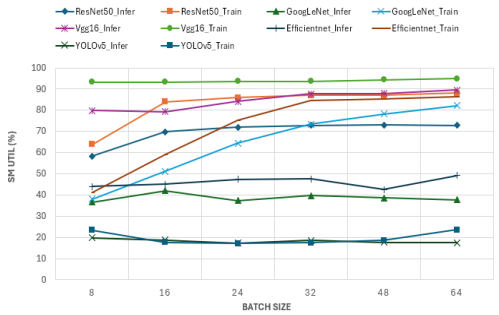
3.1 응용 규모와 파라미터와의 관계

딥러닝 응용은 응용 시나리오에 따라 다양한 파라미터 설정을 가지며, 동일한 모델이라도 파라미터 따라 필요한 메모리와 계산 자원의 양이 크게 달라진다. 따라서 본 실험은 실제 활용되는 딥러닝 모델들을 대상으로 대표적인 파라미터인 배치 크기(batch size)와 입력 데이터 크기(image size)의 변화에 따른 메모리와 계산 자원 사용량 변화를 확인하여, 응용의 실행 규모에 영향력이 큰 파라미터를 조사, 분석한다. 배치 크기와 같은 파라미터는 응용의 모델 정확도, 학습 시간, 학습 속도 등 학습 성능을 극대화하기 위한 값을 선택하는 것이 일반적이지만, 본 논문에서는 응용의 실행 규모에 미치는 영향 및 시스템의 자원 효율성을 높이기 위한 측면에 중점을 두어 해당 파라미터를 분석하였다.

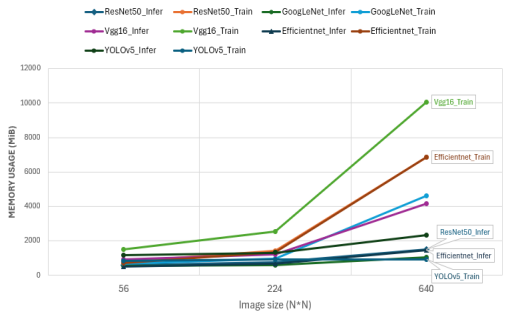
그림 2는 PyTorch 버전 2.0.0로 구현된 표 2의 딥러닝 학습 및 추론 모델을 대상으로 배치와 이미지 크기를 조절하였을 때의 메모리 사용량과 SM 활용률



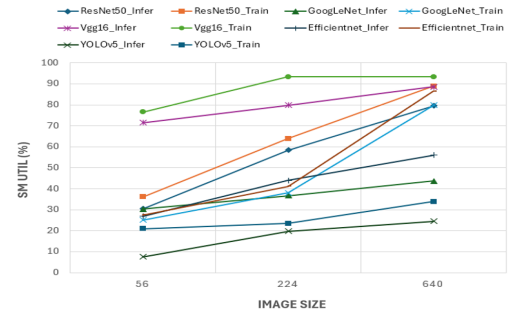
(a) 배치 크기 변화에 따른 메모리 사용량 변화
(a) Memory usage per batch size variation



(b) 배치 크기 변화에 따른 SM 활용률 변화
(b) SM utilization per batch size variation



(c) 이미지 크기 변화에 따른 메모리 사용량 변화
(c) Memory usage per image size variation



(d) 이미지 크기 변화에 따른 SM 활용률 변화
(d) SM utilization per image size variation

그림 2 응용 별 배치 크기 및 이미지 크기에 따른 메모리 사용량과 SM 활용률
Fig. 2 Application's memory usage and SM utilization for varying batch sizes and image sizes

(utilization)을 나타낸다. 배치 크기를 조절하는 경우 입력 데이터 크기는 224*224로 고정하며, 반대로 입력 데이터 크기를 조절하는 경우에는 배치 크기는 8로 고정한다. 그림 2-(a), (b)은 배치 크기가 8에서 64까지 증가할 때 모델별 메모리 사용량과 SM 활용률의 변화 비교를 보여준다. 일반적으로 메모리 사용량은 배치 크기에 비례하여 증가하는 경향을 보이지만 모델마다 그 증가폭이 다르다. 전반적인 평균 메모리 사용량이 큰 모델은 VGG16 학습이지만, 배치 크기 변화에 가장 큰 메모리 증가폭을 보인 응용은 ResNet50 학습으로 약 4.89배 증가하였다. 이는 ResNet50 학습 모델이 실행 시 요구하는 메모리 자원이 배치 크기에 따라 민감하게 변화함을 의미한다. 반면 YOLOv5 추론 모델은 메모리 사용량이 1.66배 증가하여 가장 낮은 증가폭을 보이는데, 이는 해당 모델이 배치 크기에 따라 필요로 하는 메모리 자원이 크게 변하지 않음을 의미한다. SM 활용률의 경우, 메모리 사용량과 달리 배치 크기에 따른 증가폭이 거의 없거나 특정 배치 크기에서 급격히 활용률이 증가하는 등 모델별로 다른 양상을 보인다. 특정 배치 크기 이상이 되는 시점에 큰 증가폭을 가지는 모델은 ResNet50 학습, EfficientNet 학습, GoogLeNet 학습, ResNet50 추론 모델로, 그 중 GoogLeNet 학습 모델이 배치 크기에 따라 SM 활용률이 지속적으로 증가하여 최대 2.15배의 증가폭을 보인다. 반면 나머지 응용들은 전반적으로 미세한 변화율을 보인다. 즉 일부 응용(ResNet50 학습, EfficientNet 학습 등)들은 배치 크기에 의해 메모리 사용량과 SM 활용률 변화에서 유의미한 차이를 보이며 실행 규모가 크게 달라진다.

그림 2(c), (d)는 이미지 크기와 응용 규모와의 관계를 보인다. 이미지 분류 및 인식 모델에서 이미지 크기를 56*56부터 640*640까지 증가시킬 때, 모델에 따라 서로 다른 메모리 사용량 증가폭을 보인다. 그림 2(c)의 모든 이미지 크기에서 VGG16 학습 모델이 가장 높은 메모리 사용량을 보이며, EfficientNet 학습의 메모리 사용량이 약 9.74배로 증가폭이 가장 크다. 대부분의 모델이 약 2배 이상 사용량이 증가하였으나, YOLOv5 학습 모델은 1.06배로 이미지 크기에 따른 메모리 사용량에 거의 변화가 없다. 또한 대부분의 모델(ResNet50,

VGG16, GoogLeNet, EfficientNet)은 추론보다 학습 모델에서 더 큰 메모리 사용량 증가폭을 보인다. SM 활용률의 경우, 모든 모델에서 배치 크기를 늘릴 때 보다 이미지 크기를 늘리는 경우에 비교적 큰 증가폭을 보인다. 그 중 YOLOv5 추론, GoogLeNet 학습, EfficientNet 학습 모델이 3배 이상의 큰 증가폭을 보였으며, VGG16 학습 모델이 1.21배로 가장 작은 증가폭을 보인다. 즉, 배치 크기와 이미지 크기에 따른 메모리 사용량과 SM 활용률의 변화는 딥러닝 모델에 따라 다른 양상을 보이므로, 응용의 실행 규모를 판단할 때에는 모델 및 파라미터를 함께 고려하여 공유 기술을 선택해야 한다.

3.2 자원 민감도

응용의 자원 민감도(sensitivity)는 할당된 자원 성능 변화에 따라 실행 성능이 영향을 받는 정도를 의미한다. 자원 민감도가 높을수록 성능이 낮은 자원에 배치되거나 자원이 부족한 경우 응용 성능이 크게 저하되거나 오류가 발생할 수 있기 때문에 이는 자원 할당 시 매우 중요한 요소이다. 따라서 본 절에서는 공유 기술 선택 시 응용의 자원 민감도를 고려하기 위해 (1) 응용의 모델들의 민감도 분석 및 (2) 동일 모델에 대한 파라미터 값 조합에 따른 민감도를 분석한다. 실험은 총 3가지 파라미터 조합에 대해 MIG 인스턴스들을(1g.6gb, 2g.12gb, 4g.24gb) 이용하여 할당하는 자원의 성능을 증가시켜가며 응용의 성능을 측정한다.

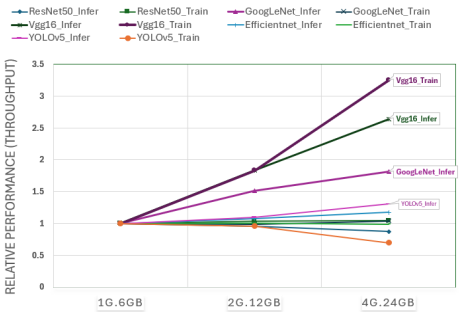


그림 3 모델별 자원 증가에 따른 처리량 비교

Fig. 3 Throughput comparison for models for resource increase

그림 3은 표 2의 딥러닝 모델들의 추론, 학습 응용 간의 민감도 차이이다. 파라미터 값은 배치 크기 8, 이미지 크기 56으로 고정하고, 가장 낮은 성능의 자원인 1g.6gb 인스턴스에서의 처리량을 기준으로 정규화하여 상대적인 민감도 차이를 계산하였다. 동일 파라미터 값에 대해 VGG16 모델이 추론, 학습 모델 모두에서 높은 자원 민감도를 보이며, 다음으로는 GoogLeNet의 추론 모델이 높은 민감도를 보인다. 이외의 모델들은 비교적 낮은 민감도를 보인다. 즉, 민감도가 높은 VGG16과 GoogLeNet 추론 모델의 경우 자원 선택에 있어서 자원의 성능이 중요한 요소이며, 민감도가 낮은 ResNet50, EfficientNet, YOLOv5, GoogLeNet의 학습 모델들은 비교적 자원을 더 유연하게 선택할 수 있다. 따라서 이기종 GPU 클러스터 상에서 공유 기술이 제한된 GPU 자원에서 time-slicing 공유 기술이 적용된다고 가정했을 때, 민감도가 낮은 응용들을 배치한다면 전반적인 성능 저하를 줄일 수 있다. 이어 그림 4는 각 ResNet50 추론 모델과 YOLOv5의 학습 모델의 민감도 변화를 보인다. 두 응용은 그림 3에서 배치 크기 8, 이미지 크기 56의 파라미터 조합일 때 공통적으로 낮은 자원 민감도를 보인다. 그러나 ResNet50 추론 모델은 파라미터가 달라지는 경우, 최대 2.19배 성능이 변화(향상)하며 자원 민감도가 증가한다. 반면 YOLOv5 학습 모델은 파라미터 값이 달라져도 성능이 최대 1.15배 변화하며 여전히 낮은 자원 민감도를 보인다. 즉, YOLOv5 학습 모델과 같이 파라미터 조합이 변하더라도 민감도가 일정하게 낮은 모델들은 자원을 더 유연하게 선택할 수 있으며, ResNet50 추론 모델과 같이 파라미터 조합에 따라 자원 민감도가 변하는 모델들은 자원 선택 시 파라미터를 고려해야 한다.

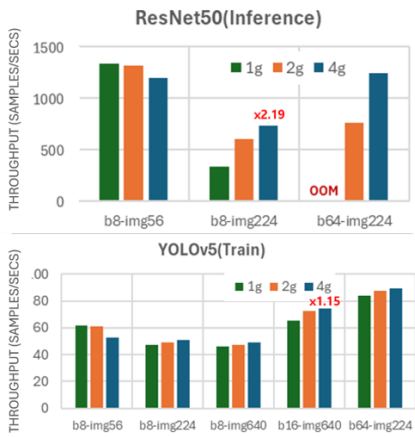


그림 4 ResNet50 추론, YOLOv5 학습의 파라미터 조합에 따른 자원 민감도 비교

Fig. 4 Comparison of resource sensitivity for ResNet50 inference and YOLOv5 training for different parameter combinations

3.3 응용의 자원 활용 특성에 따른 분류

본 절에서는 3.1절의 그림 2로부터 계산 자원 활용률과 메모리 자원 활용률의 비율(C/M)을 계산하여 각 모델을 계산 집약 응용 또는 메모리 집약 응용으로 분류한다. C/M이 10 이상일 경우 계산 집약, 10 미만은 메

표 3 응용 모델별 특성 분류
Table 3 Application classification

	Model	C/M	Classification
1	ResNet50 Inf	18.49	C (Compute intensive)
2	VGG16 Inf	16.41	
3	EfficientNet Inf	15.89	
4	GoogLeNet Inf	14.97	
5	ResNet50 Train	11.13	
6	GoogLeNet Train	9.60	M (Memory intensive)
7	VGG16 Train	9.04	
8	EfficientNet Train	7.57	
9	YOLOv5 Train	6.34	
10	YOLOv5 Inf	3.69	

모리 집약으로 분류한다(표 3).

계산 집약 응용은 실행 시간 동안 SM 자원을, 메모리 집약 응용은 메모리 자원을 집중적으로 사용한다. 계산 집약 응용과 계산 집약 응용을 동시 실행하는 경우, SM 자원에 대한 경쟁이 발생하여 성능이 저하될 수 있다. 메모리 집약 응용과 메모리 집약 응용을 동시 실행하는 경우 또한 두 응용 간에 메모리 자원에 대한 경쟁이 발생하여 성능이 저하되거나, 혹은 할당받은 GPU 자원의 가용 메모리를 초과하여 Out Of Memory(OOM) 오류를 일으킬 수 있다. 반면, 계산 집약 응용과 메모리 집약 응용을 동시 실행하는 경우 각 응용이 집중적으로 사용하는 자원의 종류가 다르므로 자원을 보다 효율적으로 공유하여 성능이 향상될 수 있다.

4. 실험 및 평가

본 절에서는 서로 다른 GPU 아키텍처에서 딥러닝 응용 간에 GPU 자원이 공유될 때 응용의 성능을 확인한다. 4.1절에서는 앞서 3.1절에서 분석한 응용의 실행 규모 분석을 기반으로 응용의 규모에 따라 어떤 공유 기법이 더 적절한지 분석한다. 이어서 4.2절은 3.3절의 응용 특성 분류에 기반하여 MPS에서 각 응용 조합별 동시 실행 시의 성능 저하를 측정한다. 4.3절에서는 앞선 분석들을 바탕으로, 이기종 GPU 클러스터 환경에서 모델 및 파라미터를 고려하여 적절한 공유 기술을 적용하였을 때 성능이 향상됨을 검증한다.

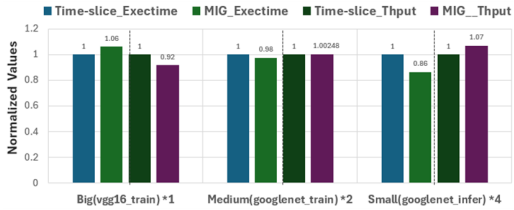


그림 5 응용 규모별 (동일 파라미터 조합의 서로 다른 응용) Time-slicing, MIG 실행 성능 비교

Fig. 5 Performance comparison of different applications (same parameter combinations) shared using time-slicing and MIG

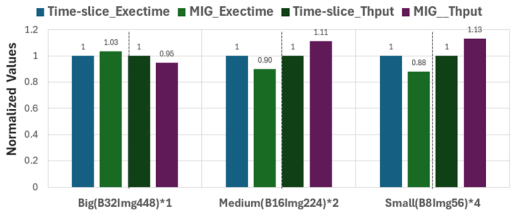


그림 6 응용 규모별(동일 모델의 다른 파라미터 조합) Time-slicing, MIG 실행 성능 비교

Fig. 6 Performance comparison of different applications (same model with different parameter combinations) shared using time-slicing and MIG

4.1 공유 기법 선택을 위한 적절한 응용 규모 선택 실험

본 절에서는 응용 실행 규모에 따라 최적의 성능을 보장하는 공유 기술을 선택하기 위해, 3.1절에서 분석한 응용의 실행 규모와 GPU 공유 기술 간의 성능 관계를 실험을 통해 분석한다. 표 4는 실험 1, 2의 대상 응용 및 실행 환경이다. 실험 1(그림 5)은 동일 파라미터 조건에서 서로 다른 응용을 대상으로, 상대적인 자원 소비량을 통해 워크로드 규모를 정의하여 각 공유 기술에 대해 실행 시간과 처리량을 비교한다. 워크로드 응용은

표 4 그림 5, 6의 실험 대상 응용 및 실험 환경
Table 4 Experiment setup for Figures 5 and 6

	Experiment 1			Experiment 2		
Model	VGG16 train	GoogLeNet train	GoogLeNet inf	EfficientNet train		
Batch*image size	64*224			32*448	16*224	8*56
# of co-running apps	1	2	4	1	2	4
MIG instance	4g.24gb	2g.12gb	1g.6gb	4g.24gb	2g.12gb	1g.6gb

규모에 따라 대(VGG16 학습), 중(GoogLeNet 학습), 소(GoogLeNet 추론)을 사용하며 배치와 이미지 크기는 64*224로 고정한다. 실험 2(그림 6)는 동일 응용에 대해 다른 파라미터 조합을 적용하여 워크로드 규모를 조절하고, 공유 기술에서의 실행 시간 및 처리량을 비교한다. EfficientNet 학습 모델은 배치 크기와 이미지 크기에 따라 메모리 사용량 및 SM 활용률이 크게 달라지는 응용이므로(3.1절 그림 2), EfficientNet 학습 모델의 파라미터를 조절하여 실행 규모를 대, 중, 소로 조절한다. 두 실험에서 응용 규모에 따라 동시 실행하는 응용의 개수를 조절하여, 전반적인 실행 규모를 균일하게 유지한다.

그림 5는 실험 1에서 time-slicing을 기준으로 정규화한 실행 시간과 처리량 비교 결과이다. 대규모의 응용은 time-slicing에서 처리량이 MIG보다 8% 높은 성능을 보였고, 중규모에서는 성능의 차이가 미미하다. 반면 소규모에서는 실행 시간이 MIG를 사용시 14% 감소, 처리량이 7% 증가하여 유의미한 성능 차이를 보인다. 그림 6는 실험 2에서 소규모를 대상으로 MIG를 활용 시 실행 시간, 처리량이 각각 12% 감소, 13% 증가하여 실험 1과 유사한 결과를 보인다. 또한 대규모 응용을 time-slicing에서 수행했을 때 실행 시간과 처리량이 각각 3% 감소, 5% 증가하여 전반적 성능 향상을 보인다. 반면 실험 1과 달리 중규모 작업에 대해서 성능 향상 비율이 증가하였다.

본 실험들을 통해, 모델 종류와 파라미터 조합에 의해 달라지는 응용의 실행 규모에 따라 공유 방식 선택을 달리하였을 때 성능 차이가 발생함을 보였다. 특히, 자원의 최대 가용 메모리가 작업이 요구하는 메모리 사용량을 충족하는 조건 하에, 응용의 실행 규모가 클수록 time-slicing 또는 MPS를 활용하고 중간 또는 작은 규모의 응용은 MIG에 배치하는 것이 실행 시간 및 처리량을 향상시켰다.

4.2 응용 간 간섭 실험

Time-slicing, MPS은 MIG와 달리 동시 실행되는 응용 간의 완전한 성능 격리를 제공하지 않으므로 응용 간의 간섭(interference)이 생겨 이에 의한 성능 저하가 발생할 수 있다. 따라서 본 실험에서는 3.3절에서의 응용 분류를 기반으로 대상 딥러닝 응용의 조합에 대해 동시 실행 시 응용별 실행 시간을 1 GPU(24GB)에서 MPS를 이용해 측정하고, 2g.12gb(12GB)에서 측정한 단일 실행 시간 결과를 바탕으로 상대적 성능을 계산하여 간섭도를 도출하였다.

그림 7은 응용 조합에 따른 동시 실행 시 상대적 실행 시간이다. 그림에서 x, y축의 1-10은 3.3절에서 계산 집약과 메모리 집약으로 분류한 응용들을 나타내며, 1에

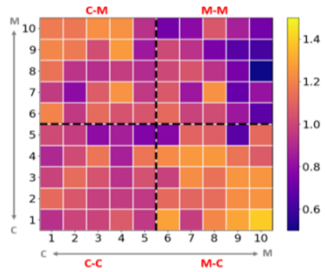


그림 7 동시 실행 시 응용별 상대적 실행 시간
Fig. 7 Relative execution time per application during concurrent execution

가까울수록 계산 집약적(C), 10에 가까울수록 메모리 집약적(M)이다. 그림 7의 (x, y) 좌표는 응용 x의 단일 실행 시간을 응용 x, y가 동시에 실행될 때 x의 실행 시간에 대해 상대적으로 정규화한 것이므로 값이 작을수록 두 응용 간의 간섭이 큼을 의미한다. 1사분면(x>5, y>5)은 메모리-메모리(M-M) 집약 응용의 조합으로 성능 저하가 가장 큰 경향을 보인다. (10, 10)으로 갈수록 메모리 집약적인 성격이 강한 응용들이 한정된 메모리 자원을 두고 경쟁하게 되므로 성능 저하가 극심하다. 4사분면(x>5, y=<5)은 메모리-계산(M-C) 집약 응용의 조합으로 대체로 1.0 이상의 값을 가지며 성능 저하의 정도가 낮다. 이는 두 응용이 단일 실행되는 경우에 비해, MPS를 통해 동시 실행되는 경우, 상대 응용이 사용하지 않는 공유 자원을 활용할 수 있기 때문이다. 3사분면(x=<5, y=<5)은 계산-계산(C-C) 집약 응용의 조합으로, 메모리-메모리(M-M) 집약과 메모리-계산(M-C) 집약의 중간 수준의 성능 저하를 보인다. 즉 계산 집약 응용 간의 동시 실행(C-C)보다 메모리 집약 응용 간의 동시 실행(M-M)이 더 극심한 성능 저하를 유발하며, 메모리 집약 응용과 계산 집약 응용(M-C)을 동시 실행하는 것이 가장 높은 효율을 가진다. 즉, 성능 저하가 적은 응용의 조합은 time-slicing 또는 MPS에 우선 배치하고, 반대로 성능 저하가 큰 응용 조합은 MIG 기반의 자원에 배치하는 것이 상호 간섭을 줄여 시스템 전체의 성능을 향상시킨다.

4.3 전체 시스템 성능 검증

본 절에서는 앞 세 가지 실험 분석을 토대로 구성한 작업 배치 기준을 실제 쿠버네티스로 구성한 GPU 클러스터 상에서 적용하여 성능을 측정 및 분석한다.

4.3.1 실험 환경

표 5는 실험에 사용한 클러스터의 GPU 자원 및 적용 공유 기술 정보이며, 쿠버네티스 v1.29.0 환경에서 진행되었다. 대상 응용 및 파라미터는 표 6과 같으며, 각 응용은 동일한 모델에서 파라미터를 조절하여 대, 중, 소

표 5 실험 GPU 자원 및 공유 기술
Table 5 GPU resources and sharing technologies used in the experimental cluster

	GPU count	Architecture	Sharing technology
Titan XP	4	Pascal	Time-slicing
RTX 2080 Ti	1	Turing	MPS
A30 PCIe	2	Ampere	MIG

표 6 워크로드 크기에 따른 모델 및 파라미터
Table 6 Workload sizes based on model parameters

Workload Size	Batch* Image Size	Model
Small	5*56	ResNet50 train/inf, VGG16 train/inf, GooLeNet train/inf, EfficientNet train/inf, YOLOv5 train/inf
Medium	16*224	(The same as above)
Big	48*448	ResNet50 train, VGG16 train, GoogLeNet train, EfficientNet train
	64*448	VGG16 inf, YOLOv5 train/inf
	64*640	ResNet50 inf, GoogLeNet inf, EfficientNet inf

의 실행 규모를 갖도록 구성하였다. 대규모의 응용은 실험에 사용되는 GPU의 가용 메모리를 초과하지 않는 선에서 최대한 큰 파라미터 값을 갖도록 설정한다. 각 응용을 3개씩 중복시켜 총 180개의 응용으로 이루어진 워크로드를 구성하며, 해당 워크로드를 20회 반복 실행해 얻은 실행 시간과 처리량의 평균을 측정한다.

4.3.2 배치 기준

앞 절의 분석을 바탕으로 구성한 자원 배치의 구체적인 기준은 다음과 같다. 우선, 대규모이며 동시에 자원 민감도가 낮은 응용을 time-slicing에 배치한다. 중/소규모이면서 동시에 자원 민감도가 높은 응용들은 MPS와 MIG에 배치한다. 이때 GPU에 실행 중인 작업이 없다면 자원 민감도를 고려하여 자원 민감도가 높은 응용을 MIG에, 낮은 응용을 MPS에 배치한다. GPU에 메모리 집약 작업이 실행 중인 경우 새로 들어오는 작업이 메모리 집약이라면 MIG에 배치하고, 메모리 집약이 아닌 경우 MPS에 배치한다. 이미 실행 중인 작업과 새로 들어오는 작업의 조합이 계산-계산 집약(C-C) 또는 계산-메모리 집약(C-M)인 경우, 자원 민감도가 높은 응용을 MIG, 자원 민감도가 낮은 응용을 MPS에 배치한다. 비교 대상으로는 각 자원에 무작위로 워크로드를 배치하는 Random과 응용의 실행 규모만을 고려하여 자원을 배치하는 ByScale을 사용한다. ByScale은 대규모에 해당하는 응용들은 time-slicing에, 중규모에 해당하는 워크로드들은 MPS, 소규모의 워크로드들은 MIG에 배치한다.

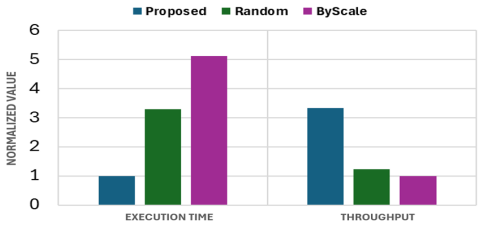


그림 8 응용 배치 기준에 따른 실행 시간 및 처리량 비교
Fig. 8 Comparison of execution time and throughput for different application placement criteria

4.3.3 실험 결과

그림 8은 제안된 작업 배치 기준과 비교 대상들의 실행 시간 및 처리량을 가장 최소값을 기준으로 정규화한 값이다. 작업 배치 기준을 적용하였을 때 전체 실행 시간이 Random과 ByScale보다 약 3.1배, 5배 감소하였으며, 처리량은 약 2.7배, 3.3배 증가하였다. 이는 제안된 작업 배치 기준이 실행 규모, 응용 조합별 간섭도, 자원 민감도를 동시에 고려하여 공유 기술 선택 및 자원 배치를 효과적으로 수행했음을 의미한다. 특히 ByScale의 경우 비교적 긴 실행 시간을 보이는데, 이는 ByScale이 응용의 실행 규모만을 고려하므로 메모리 집약 응용들 간의 동시 실행을 허용하고 높은 자원 민감도를 갖는 응용들을 time-slice에 배치하였기 때문이다. 즉 응용의 실행 규모뿐만 아니라 응용 조합별 간섭도 및 자원 민감도를 함께 고려하는 것이 성능에 유의미한 영향을 미친다.

5. 결론 및 향후 연구

본 논문은 딥러닝 응용의 모델 및 파라미터 조합을 고려하여 메모리 사용량, SM 활용률 및 자원 민감도를 분석하였다. 이어서 자원 활용 특징을 통해 응용을 분류하고 공유 기술에 따른 최적의 조합을 파악하였다. 분석 결과를 통해 구성한 작업 배치 기준으로 각 자원과 GPU 공유 방식에 최적화된 작업 배치를 수행하여, 성능 평가 실험을 통해 작업 배치 기준의 효율성을 보였다. 향후 연구에서는 Large Language Model(LLM)을 포함한 확장된 워크로드를 대상으로 응용의 GPU 활용도 및 공유 기술 사용 시의 성능 변화를 분석할 계획이다.

References

[1] Google. (2022, October 20). Google cloud platform. Available: https://cloud.google.com/gpu
[2] Meta. Meta data centers. Available: https://datacenters.atmeta.com/

[3] Amazon Web Services. Amazon data center. Available: https://aws.amazon.com/ko/compliance/data-center/controls/
[4] NVIDIA. NVIDIA Cloud & Data Center. Available: https://www.nvidia.com/en-us/data-center/
[5] Z. Mo, H. Xu, and W. C. Lau, "Optimal Resource Efficiency with Fairness in Heterogeneous GPU Clusters," *arXiv preprint arXiv:2403.18545,* 2024.
[6] H. Albahar, S. Dongare, Y. Du, N. Zhao, A. K. Paul, and A. R. Butt, "SchedTune: A Heterogeneity-Aware GPU Scheduler for Deep Learning," *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid),* pp. 695-705, 2022.
[7] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22),* pp. 945-960, Apr. 2022.
[8] NVIDIA. NVIDIA Time-Slicing GPUs in Kubernetes. Available: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html
[9] NVIDIA. NVIDIA Multi-Process Service (MPS). Available: https://docs.nvidia.com/deploy/mps/index.html
[10] NVIDIA. NVIDIA Multi-Instance GPU (MIG). Available: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/
[11] HUAWEI CLOUD. Volcano device plugin for Kubernetes. Available: https://github.com/volcano-sh/devices
[12] Kubernetes. Kubernetes. Available: https://kubernetes.io/ko/docs/concepts/overview/
[13] B. Li, S. Samsi, V. Gadepally, and D. Tiwari, "Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service," *International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23),* Article 20, 1-15, 2023.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition,* pp. 770-778, 2016.
[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556,* 2014.
[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2015.
[17] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *36th International Conference on Machine Learning,* Vol. 97, pp. 6105-6114, 2019.
[18] G. Jocher, "Ultralytics YOLOv5 (Version 7.0) [Software]," AGPL-3.0, https://doi.org/10.5281/zenodo.3908559, 2020.

하 지 원

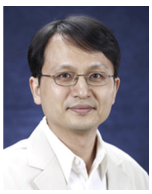
2023년 고려대학교 컴퓨터학부 졸업(학사) 2023년~현재 서울대학교 대학원 컴퓨터공학부 석박통합과정. 관심분야는 GPU 스케줄링, 딥러닝, 분산 시스템

김 서 영

2010년 숙명여자대학교 컴퓨터과학과 졸업(학사). 2012년 숙명여자대학교 대학원 컴퓨터과학부 졸업(석사). 관심분야는 그리드/클라우드 관리, 메타 스케줄링, 워크플로우 제어

테오도라 아두푸

2013년 가나대학교 컴퓨터과학 및 경제학과 졸업(학사). 2016년 숙명여자대학교 컴퓨터과학과 졸업(석사). 2022년~현재 숙명여자대학교 컴퓨터과학과 박사과정. 관심분야는 컨테이너, 클라우드 컴퓨팅, GPU 스케줄링, 딥러닝



엄 현 상

1992년 서울대학교 계산통계학과(학사) 1996년 메릴랜드대학교 전산(석사). 2003년 메릴랜드대학교 전산(박사). 2003년~2005년 삼성전자 정보통신총괄 책임연구원. 2005년~2011년 서울대학교 컴퓨터공학부 전임강사 후 조교수. 2012년~2017년 2월 서울대학교 컴퓨터공학부 부교수. 2017년 3월~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 분산시스템, 블록체인

김 윤 희

1991년 숙명여자대학교 전산학과 졸업(학사). 1996년 Syracuse University 전산학과 졸업(석사). 2000년 Syracuse University 전산학과 졸업(박사). 1991년~1994년 한국전자통신연구원 연구원. 2000년~2001년 Rochester Institute of Technology 컴퓨터공학과 조교수. 2001년~2016년 숙명여자대학교 컴퓨터과학부 교수. 2017년~현재 숙명여자대학교 소프트웨어학부 교수. 관심분야는 클라우드 시스템, 워크플로우 제어