https://doi.org/10.7840/kics.2024.49.10.1377

거대 언어 모델의 일괄 추론 수준에 따른 추론 시간, 정확도, 통신 및 GPU 메모리 점유량 비교분석

신 창 용◆, 고 영 훈*, 유 연 호*, 양 경 식°, 유 혁°°

An Analysis on Inference Time, Accuracy, Communication, and GPU Memory Usage for Inference Batch of Large Language Models

Changyong Shin◆, Younghun Go*, Yeonho Yoo*, Gyeongsik Yang°, Chuck Yoo°°

요 약

최근 거대 언어 모델(GPT, LLaMA, PaLM 등)은 의학, 교육, 금융, 법학, 마케팅 등 다양한 분야에서 활발히 활용되고 있다. 이러한 모델들은 매우 많은 매개변수를 지니고 있어 여러 GPU를 활용해야만 모델을 로드하고 추론을 수행할 수 있다. 추론 서비스를 운영하는 클러스터나 클라우드의 시스템 관리자에게는 주어진 GPU와 네트워크 자원을 최대한 효율적으로 사용하면서 많은 사용자 요청에 빠르게 응답하는 것이 매우 중요하다. 이를 위해 현재의 거대 언어 모델 추론 시스템은 다양한 병렬화 및 최적화 전략을 사용하고 있다. 본 논문은 LLM의 추론 과정에서 병렬화, 최적화 전략, 및 배치 사이즈의 변화에 따른 추론 시간, 예측 정확도, GPU 통신량과 GPU 메모리 점유량을 상세히 프로파일링하고 분석한다. 특히, 본 연구는 GPU에 대한 엄밀한 자원 측정을 위해 프로파일러를 새롭게 개발하여 사용한다. 프로파일링 및 분석 결과, 본 연구는 배치 사이즈가 증가하면 병렬화 전략에 의해 GPU 통신량이 증가하여 비효율성을 초래할 수 있음을 관측한다. 반면, GPU 메모리 측면에서는 배치 사이즈가 커질수록 메모리를 더 적극적으로 활용하나, 물리 메모리 크기를 초과하여 메모리 부족(out-of-memory)이 발생하는 특정 임계점이 존재함을 확인한다. 이러한 관측은 향후 LLM의 효율적인 추론 시스템을 설계하는 데 필요한 중요한 기반이 될 것으로 기대된다.

키워드 : 거대 언어 모델, GPU 활용률, 통신 병목, 모델 병렬화, 텐서 병렬화, 커널 병합, 배치 사이즈
Key Words : Large language model, GPU utilization, Communication overhead, Model parallelism, Tensor parallelism, Kernel fusion, Batch size

ABSTRACT

Recently, large language models, such as GPT, LLaMA, and PaLM, have been actively applied in various fields such as medicine, education, finance, law, and marketing. These models have a vast number of parameters

※ 이 논문은 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(NRF-2021R1A6A1A13044830), 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2023R1A2C3004145, RS-2024-00336564), 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 ICT명품인재양성사업(IITP-2024-2020-0-01819), 차세대네트워크(6G)산업기술개발(RS-2024-00405128)의 지원을 받아 수행된 연구임. 또한 본 연구는 Google Cloud Research Credits program의 지원을 받은 연구임.
◆ First Author : Korea University Department of Computer Science and Engineering, cyshin@os.korea.ac.kr, 학생회원
° Corresponding Author : Korea University Department of Computer Science and Engineering, g_yang@korea.ac.kr, 정회원
°° Corresponding Author : Korea University Department of Computer Science and Engineering, hxy@korea.ac.kr, 종신회원
* Korea University Department of Computer Science and Engineering, yhgo@os.korea.ac.kr; yhyoo@os.korea.ac.kr, 학생회원
논문번호 : 202406-117-C-RU, Received June 14, 2024; Revised July 3, 2024; Accepted July 15, 2024

that require multiple GPUs to perform inference. For system administrators of inference services in clusters or clouds, it is critical to utilize the given GPU and network resources as efficiently as possible to quickly respond to numerous user requests. To achieve this, existing inference systems employ various parallelization and optimization strategies. This paper profiles and analyzes inference time, prediction accuracy, GPU communication amount, and GPU memory usage for different parallelization strategies, optimization techniques, and batch size changes. Notably, we develop a new resource profiler for precise resource measurement of GPU resources. Our profiling results reveal that increasing batch size can lead to inefficiencies due to increased GPU communication. In terms of GPU memory, larger batch sizes result in more aggressive memory utilization, but a specific threshold exists where out-of-memory issues arise for the limited GPU memory. Such observations are expected to serve as a baseline for designing efficient inference systems for large language models.

Ⅰ. 서 론

최근 컴퓨터 기술의 급속한 발전으로 인해 기존의 텍스트 위주의 사용자 환경에서 벗어나 이미지, 그래픽, 오디오 및 비디오 데이터 등을 제공하는 멀티미디어 사용자 환경으로 변화하고 있다.

최근 GPT, LLaMA, PaLM 등의 거대 언어 모델(large language model, LLM)은 의학, 교육, 금융, 법학, 마케팅 등 다양한 분야에서 활발히 활용되고 있다[1]. 이러한 모델들은 매우 많은 매개변수를 지니고 있어, 1기의 GPU 메모리에 모두 로드될 수 없으며 여러 GPU를 활용해야 모델을 로드하고 추론을 수행할 수 있다. 사용자는 거대 언어 모델을 사용하기 위해 자신의 질의를 프롬프트로 입력하고, 모델은 이 프롬프트를 기반으로 추론을 수행하여 결과를 반환한다.

추론 서비스를 운영하는 클러스터나 클라우드의 시스템 관리자에게는 주어진 GPU와 네트워크 자원을 최대한 효율적으로 사용하면서 많은 사용자 요청에 빠르게 응답하는 것이 매우 중요하다[2,17]. 이를 위해 현재의 거대 언어 모델 추론 시스템은 다양한 병렬화 및 최적화 전략을 사용하고 있다.

첫째, 병렬화는 거대한 모델의 추론을 위해 여러 대의 GPU를 사용하는 방식이다. 이는 모델을 분할하여 여러 GPU를 통해 추론을 수행하는 방식으로, 한정된 메모리를 지닌 GPU를 여러 대 활용한다. 이 과정에서 GPU 간 통신이 발생하게 되는데, 통신을 수행하는 시간 동안에는 모델 레이어의 연산이 멈추게 되므로 통신량을 사전에 파악하고 줄이는 것이 중요하다[3-4]. 또한, 각 GPU는 32GB 등의 한정된 메모리를 가지고 있어, 분할되어 배치된 모델과 그 파라미터들이 메모리를 얼마나 효율적으로 사용하는지가 중요하다.

둘째, 병렬화된 모델에서 각 GPU의 연산 병목을 줄이기 위해 단일 연산 병합과 같은 GPU 내 최적화 전략이 자주 사용된다. 이는 모델 내 레이어를 구현하는 많은 개별 연산을 GPU에서 실행할 때 데이터 로드와 문맥 준비 등의 병목을 개선하기 위함이다. 이러한 최적화 전략은 효율적인 추론 시스템 구성을 위해 적극적으로 고려되는데, 전략의 활용 여부에 따라 GPU 간 통신량 또는 메모리 활용률이 상이해진다.

더 나아가 GPU 자원의 활용은 1회 추론 시 사용되는 입력 데이터의 규모를 결정하는 배치 사이즈에 따라 영향을 받으며, 이는 전체 추론 성능과 직접 관련이 있다. 따라서 병렬화/최적화 전략 및 배치 사이즈의 대소 관계에 대한 복합적인 이해와 분석이 필요하다.

즉, 위에서 설명한 병렬화 방식, 최적화 전략(커널 병합) 및 배치 사이즈의 선택 등에 따라, GPU 자원의 효율적 소모가 매우 달라진다. 하지만, 지금까지 제안된 연구들은 이러한 기법과 전략에 따른 자원 소모 양상을 정량적으로 분석하지 않았다[5-7]. 따라서 특정 기법이 모델의 추론 정확도, 시간, 자원 사용량에 어떤 복합적인 영향을 미치는지에 대한 정보가 부족하기 때문에, 실제 추론 파이프라인 구축 과정에서는 각 요소를 경험적으로 추가 또는 삭제하면서 결정을 내리고 있다. 이는 많은 수의 GPU와 시간을 소모하는 LLM 추론 시스템을 고려할 때, 높은 사회적 및 경제적 비용을 야기한다.

본 논문은 이러한 지점에서 LLM의 추론 과정에서 1) 병렬화, 최적화 전략 및 2) 배치 사이즈의 변화에 따른 LLM 모델의 예측 정확도와 추론 시간을 측정한다. 또한, 핵심 시스템 자원인 GPU 통신량과 GPU 메모리 점유량을 상세히 프로파일링하고 분석한다. 구체적으로, 우리는 Meta에서 제안한 LLaMA 모델을 사용하여 대표적인 병렬화 전략들(모델 병렬화, 텐서 병렬화)과 커널 병합을 선택적으로 적용했을 때 자원 사용량의 변화를 측정한다. 또한 배치 사이즈가 변화할 때 상기

전략들을 복합적으로 분석한다. 특히, 본 연구는 GPU 통신량과 메모리 사용량의 엄밀한 자원 측정을 위한 프로파일러를 새롭게 개발하여 사용한다.

실험 결과, 배치 사이즈가 증가하면 병렬화 전략에 의해 GPU 통신량이 증가하여 비효율성을 초래할 수 있음을 밝혔다. 반면, GPU 메모리 측면에서는 배치 사이즈가 커질수록 메모리를 더 적극적으로 활용하나, 물리 메모리 크기를 초과하여 메모리 부족(out-of-memory, OoM)이 발생하는 특정 임계점이 존재함을 확인했다. 이러한 관측은 향후 LLM의 효율적인 추론 시스템을 설계하고 개발하는 데 필요한 중요한 지식을 제공한다.

본 논문의 주요한 기여점은 아래와 같다:

- **새로운 프로파일러 개발:** LLM의 추론 과정에서 GPU 통신량과 메모리 사용량을 정밀하게 측정할 수 있는 프로파일러를 새롭게 개발한다. 이를 통해 기존 도구로는 얻기 어려운 세밀한 시간 단위의 자원 사용량을 측정하고 분석한다.
- **배치 사이즈의 영향 분석:** LLM 모델에 대해 최초로 배치 사이즈의 변화가 모델의 예측 정확도와 자원 사용량에 미치는 영향을 다각도로 분석한다. 특히, 배치 사이즈 증가에 따른 GPU 통신량 증가와 메모리 부족 현상의 임계점을 확인한다.
- **병렬화/최적화 전략 분석:** 모델 병렬화와 텐서 병렬화와 같은 다양한 병렬화 전략이 GPU 자원 사용에 미치는 영향을 상세히 분석한다. 또한 커널 병합과 같은 최적화 전략이 자원 효율성에 미치는 영향을 평가한다. 이를 통해 적절한 병렬화, 최적화 전략의 선택이 GPU 내 연산 병목을 줄이고, 전반적인 추론 성능을 향상시킬 수 있음을 확인하고 효율적인 추론 시스템 구축에 필수적인 지식을 제공한다.

Ⅱ. 거대 언어 모델의 추론 시스템 구성

본 장은 LLM 추론 시 모델을 GPU에 로드하기 위한 병렬화와 배치 사이즈 선택에 대해 설명한다.

2.1 병렬화 및 최적화 전략

2.1.1 모델 병렬화

LLM의 추론을 위해서는 모델의 구조 및 파라미터를 GPU 메모리에 로드해야 한다. 일반적인 LLM은 수십에서 수백 억 개의 파라미터를 지니며, 한 기의 GPU에 로드될 수 없다. 따라서 복수의 GPU에 모델을 레이어별로 배치하는 "모델 병렬화"(model parallelism)[15]를 활용 가능하다. 그림 1은 간단한 모델 구조에서의 모델

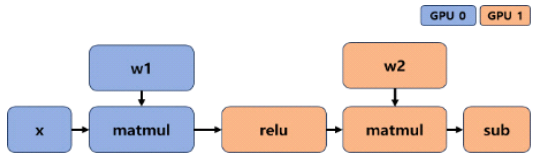


그림 1. 모델 병렬화 기법 예시
Fig. 1. Model parallelism example.

병렬화 예시를 보여준다. 모델의 레이어가 GPU 0와 GPU 1에 분할되어 배치되며, 첫 레이어부터 순차적으로 수행된다. 이 때, GPU 0에서 도출되는 중간과정 가중치들을 GPU 1에 네트워크를 통해 송신하여 연산을 재개한다. LLM의 경우 각 트랜스포머(transformer) 레이어의 크기 및 연산량이 동일하며, 모델 병렬화 적용 시 각 GPU 마다 동일한 수의 디코더 레이어가 로드된다.

2.1.2 텐서 병렬화

그림 1과 같이 모델 병렬화를 사용하면 1회의 추론 과정에서 특정 GPU가 연산을 수행하는 동안, 선행 또는 후행 레이어가 배치된 GPU는 연산을 수행하지 않아 유휴하며 활용률이 낮아지는 문제가 있다. 이를 극복하기 위해 최근 LLM을 트랜스포머 레이어 내의 텐서별로 분할하여 배치하는 "텐서 병렬화" 기법(tensor parallelism)[8]이 활용되고 있다. 그림 2는 텐서 병렬화의 예시를 보여주는데 모델의 레이어를 구성하는 텐서를 GPU 0과 GPU 1에 나누어 배치한다.

모델 병렬화에 비교해보면, 두 GPU가 레이어 전체의 구조를 유지하고 있으므로 항상 연산을 수행하는 상태에 놓이게 된다. 그러나, 개별 텐서의 연산마다 두 GPU 사이의 연산 결과 교환이 필요하므로 통신량이 증가한다는 특징이 있다. 일반적으로 고성능 GPU가 확보되는 거대 자연어 모델의 추론에 있어서는 텐서 병렬화 기법이 적극 활용된다.

또한 LLM의 트랜스포머 레이어는 self-attention 레이어와 MLP 레이어로 구성되어 있으며, 텐서 병렬화 구현 시 이러한 레이어 구조를 고려하여 all-reduce 통신을 최소화할 수 있도록 구현되었다. 따라서, 텐서 병렬화를 LLM이 아닌 CNN 등의 다른 레이어 구조를

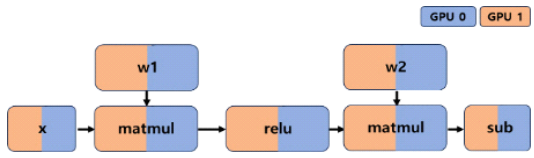


그림 2. 텐서 병렬화 기법 예시
Fig. 2. Tensor parallelism example.

갖는 딥러닝 모델에 적용하기 위해서는 해당 딥러닝 모델에 최적화된 별도의 텐서 병렬화 구현이 사용된다.

2.1.3 커널 병합

커널 병합(kernel fusion)[11]은 GPU에서 실행되는 여러 개의 커널들을 하나의 커널로 합치는 기술을 의미한다. 커널은 GPU에서 수행되는 개별 연산 단위를 의미하는데, 각 커널을 실행시킬 때마다 1) GPU에 해당 커널을 로드하는 시간, 2) 메모리를 통한 데이터 전송 시간, 3) 커널 간의 동기화 및 스케줄링 병목 등 맥락 병목(context overhead)이 발생한다[9]. 커널 병합은 상호 간 의존성이 없는 커널들을 최대한 하나의 커널로 병합하여, 커널을 로드하는 데 소요되는 맥락 병목을 개선한다. 딥러닝 모델 별 실행하는 커널의 종류와 각 커널의 의존성이 다르기 때문에, 이를 고려한 커널 병합이 요구된다. 예를 들어 LLM의 경우 트랜스포머 레이어 내에서 bias 파라미터와 residual 파라미터의 덧셈을 위한 커널들을 병합하면, 병합하지 않은 상황에 비해 최대 1.2배 가량 맥락 병목이 개선됨이 알려져 있다[10].

LLM에 적용 가능한 커널 병합의 대표적인 연구는 Microsoft의 SBI-GeMM으로, LLM의 학습과 추론에 빈번히 사용되는 행렬 곱셈에 소요되는 커널들을 병합한다[11]. 구체적으로 SBI-GeMM은 self-attention과 같은 핵심적인 연산을 병합할 뿐만 아니라, 서브 레이어(변형 레이어) 또한 병합하여 연산 및 메모리의 효율성을 높인다.

2.2 배치 사이즈 선택

배치 사이즈는 학습 또는 추론 과정에서 얼마나 많은 데이터를 한 번에 함께 처리할지를 나타내는 지표이다. 가령 추론에서 배치 사이즈가 1이라는 의미는, 1회의 추론과정 시 거대 자연어 모델에 대한 입력 프롬프트를 1개를 처리함을 뜻한다. 배치 사이즈를 1과 같이 작게 설정하면, 사용자의 입력 프롬프트가 즉각 추론을 거치기 때문에 추론 요청에 대한 처리 지연을 줄일 수 있다. 그러나 이러한 방식은 GPU가 지닌 수백, 수천의 연산 코어를 충분히 활용하지 못하는 심각한 문제가 존재한다.

반면 배치 사이즈를 512 등과 같이 크게 설정하면, 여러 추론 요청이 1회 추론으로 처리되기 때문에 GPU의 활용률은 개선할 수 있으나 개별 사용자의 요청은 배치 사이즈 만큼의 요청들이 축적될 때까지 지연이 증가한다. 더 나아가 GPU는 32GB(NVIDIA V100) 등 제한된 크기의 메모리를 지니므로, 메모리 크기를 넘어서는 배치 사이즈는 메모리 부족을 발생시켜 추론을 중단시킨다. 따라서 처리 지연과 GPU의 활용량을 균형적으로 달성하는 적절한 배치 사이즈 결정이 중요하다.

Ⅲ. 기존 연구의 한계

병렬화, 최적화 전략 및 배치사이즈의 크기에 따라 GPU의 네트워크 통신량 및 메모리 점유량이 변화함에도 불구하고, 현재까지 제안된 연구에서는 해당 양상이 명확하게 파악되지 않았다. 표 1은 관련 연구의 통신량 및 GPU 사용량 양상 관찰 유무를 나타낸다. 가령 Megatron-LM[5]과 TeraPipe[6]는 배치 사이즈 변화에 따른 처리량 또는 레이턴시를 관찰했지만, 추론 시 발생하는 통신량과 GPU 사용량에 대한 양상을 관찰하지 않았다. 또한 Zhuang et al.[7]은 병렬화 전략에 따른 통신량을 관찰했지만, 배치 사이즈의 변화에 따른 통신량 및 GPU 사용량의 양상은 관찰하지 않았다.

표 1. 통신량 및 GPU 사용량 관찰 연구
Table 1. Related work on analysis of communication and GPU usage.

	Communication analysis	GPU usage analysis
Megatron-LM[5]	X	X
TeraPipe[6]	X	X
Zhuang et al.[7]	X	X
This study	O	O

Ⅳ. 자원 사용량 프로파일링 방법 및 분석 결과

본 장은 텐서 병렬화를 적용한 LLM이 일괄 추론 수준에 따라 통신과 GPU 사용량이 어떻게 변화하는지 살펴보고 그 결과를 논의한다. 먼저 자원 사용량을 프로파일링하는 방식에 대해 설명하고, 다음으로 측정 결과를 논의한다.

4.1 자원 사용량 프로파일링

추론 과정의 통신량과 GPU 자원 활용률을 세밀하게 측정하기 위해 우리는 자체 프로파일러를 개발한다. 일반적으로 실시간 GPU 모니터링을 위해 NVIDIA-SMI 등의 도구를 사용하며, GPU의 상태 및 목표로 하는 지표의 확인이 가능하다. 그러나 기본적으로 1초 주기로 지표를 갱신하기 때문에, 빠른 속도로 수행되는 GPU에서의 연산을 측정하기에 부정확하다[12].

이에 우리는 NVIDIA의 NVML API[13]를 이용하여 1/6초마다 자원 사용량을 정확히 측정하는 프로파일러를 개발한다. 그림 3은 C++ 언어로 작성한 프로파일러

코드의 핵심 루틴이다. 프로파일러 초기화 이후 노드에 장착된 GPU의 개수를 카운트하여 각 GPU의 핸들러를 생성한다. 이후 GPU 활용률 등의 지표를 측정하기 위해 NVML API를 호출한다. 구체적으로, GPU 활용률, GPU 메모리 점유량, GPU 통신량 측정을 위해 각각 nvmlDeviceGet UtilizationRates, nvmlDeviceGet MemroyInfo, nvml DeviceGetPcieThroughput API를 사용한다. 또한, 매 API 호출마다 소모량 수신까지 걸리는 지연을 고려하여, API 호출에 소요된 시간을 측정하고 이전 측정과 다음 측정의 간격이 1/6초가 될 수 있도록 대기시간을 보정한다.

```
int main(int argc, char* argv[])
{
    ...
    result = nvmlInit();
    if (result != NVML_SUCCESS)
        return 1;

    result = nvmlDeviceGetCount(&device_count);
    if (result != NVML_SUCCESS)
        return 2;

    for (int i = 0; i < device_count; ++i) {
        nvmlDevice_t device;
        result = nvmlDeviceGetHandleByIndex_v2(i, &device);
        if (result != NVML_SUCCESS)
            return 3;
```

그림 3. C++ 언어 기반 프로파일러 코드
Fig. 3. Profiler code implemented in C++.

4.2 프로파일링 방법

4.2.1 환경

자원 사용량을 측정하기 위해 본 연구는 Intel Xeon Gold 5218 CPU(총 64 코어) 및 512GB 메모리, PCIe로 연결된 NVIDIA V100 GPU(32GB 메모리) 4기를 장착한 서버 1대를 사용한다. 서버의 GPU들은 PCIe 3.0 버스로 연결되어 있다. 딥러닝 프레임워크로 PyTorch 2.0 버전, CUDA 11.6 버전, 파이썬 3.9.1 버전을 사용한다. LLM 모델은 오픈소스로 공개되어 있고 Hugging Face 커뮤니티에서 활발히 활용되는 LLaMA-30B 모델(llama-30b-hf) 및 HellaSwag 데이터셋을 사용한다.

4.2.2 측정 지표

본 연구에서는 먼저 모델의 예측 정확도 및 추론 시간을 비교, 분석한다. 또한, GPU 활용에 있어서 두 가지 핵심 지표인 1) GPU 통신량과 2) GPU 메모리 점유량을 측정한다. 자원 사용량 측정을 위해 배치 사이즈를 1부터 2배씩 증가시키며 실험을 진행하였고, 모델 병렬화와 텐서 병렬화는 각각 64까지, 커널 병합은 32까지 증가시켰다. 배치 사이즈의 최대값은 모델 병렬화, 텐서 병렬화, 커널 병합 적용 시 메모리 부족(OoM)이 발생하지 않는 최대 크기로 산정한다. 동일한 배치 사이즈라도 각 모델 병렬화 및 최적화 전략에 따라, 개별 GPU에 모델의 레이어와 파라미터가 배치되는 정도가 다르기 때문에 해당 상한선은 상이하다.

각 실험은 앞서 II장에서 소개한 세 가지 병렬화 및 최적화 전략인 1) 모델 병렬화, 2) 텐서 병렬화, 그리고 3) 커널 병합 기법에 대해 각각 시행하고 비교한다. 커널 병합 실험은 기존 연구 및 기술이 대부분 텐서 병렬화를 기반으로 구현 및 활용되고 있어[5, 14], 텐서 병렬화가 활성화된 상태에서 커널 병합을 추가로 수행하여 진행하였다. 커널 병합은 Microsoft의 DeepSpeed[11] 프레임워크를 사용하며, 프레임워크가 제공하는 병합된 커널을 추론에 활용한다.

4.3 모델 정확도, 추론 시간 결과 및 분석

표 2는 배치 사이즈 별 모델 병렬화(표의 MP), 텐서 병렬화(TP), 커널 병합(IoK) 기법을 적용했을

때 모델의 예측 정확도를 보여준다. 예측 정확도의 기초값(baseline)은 Meta에서 공식 발표한 LLaMA-30B 모델[16]의 정확도인 82%이다. 표의 결과를 살펴보면 IoK를 제외한 두 기법에서 배치 사이즈가 작을 때보다 클 때 전반적으로 추론 시간이 빨라지는 결과를 보인다. 이는 다수의 추론 요청을 배칭하는 효과로 kernel invocation 오버헤드를 감소시키고, GPU의 코어를 충분히 활용하기 때문이다. IoK 기법은 배치 사이즈 64에서 메모리 부족을 겪어, 모델의 추론이 이루어지지 않는다.

IoK에서는 배치 사이즈의 증가에 따라 추론 시간이 선형적으로 개선되지는 않는데, 이는 커널 병합을 통해 요청 별 GPU 커널 병목을 개선하는 IoK 기법이 작은 배치 사이즈에서 더욱 유효하게 나타나기 때문이다.

표 2. 모델 추론 시간 및 정확도 결과.
Table 2. Model inference time and accuracy results.

Batch size	MP	TP	IoK
1	10258 s/82%	6548 s/82%	4959 s/82%
2	6159 s/82%	5084 s/82%	4171 s/82%
4	4622 s/82%	4538 s/82%	4250 s/82%
8	4043 s/82%	4280 s/82%	4152 s/82%
16	3807 s/82%	4164 s/82%	4027 s/82%
32	3705 s/82% (best)	4087 s/82%	4201 s/26%
64	3800 s/82%	4079 s/82%	OoM

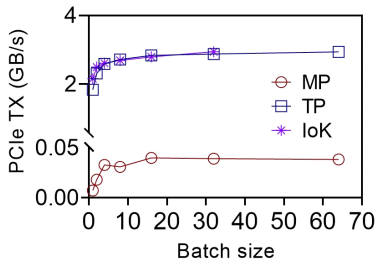
4.4 GPU 통신량 프로파일링 결과 및 분석

그림 4는 배치 사이즈 별 모델 병렬화(그림의 MP), 텐서 병렬화(TP), 커널 병합(IoK) 기법의 GPU 통신량을 나타낸다. 특히, 본 실험에서는 각 GPU의 관점에서 송신량(TX)과 수신량(RX)을 상세히 나누어 분석한다. 실험은 모두 4기의 GPU를 활용하는데, 그래프에 도시한 값은 각 GPU에서 측정한 송/수신량의 평균이다. 즉, 그림 4a의 PCIe TX는 GPU가 초당 송신하는 데이터의 양이며, 그림 4b의 PCIe RX는 GPU가 초당 수신하는 데이터의 양이다.

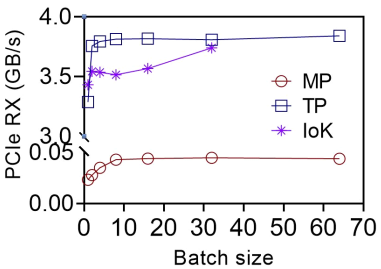
그림 4a, 4b를 살펴보면 송신, 수신량 모두에서 MP 기법에 비해 TP, IoK 기법에서 월등히 높은 통신량을 보이는데(최대 292배), 이 차이는 각 최적화 기법의 동작 방식 때문에 발생한다. MP에서는 앞선 GPU에 위치한 레이어의 연산 결과를 다음 GPU에 위치한 레이어로 전달하기 위해 GPU 간 통신을 사용하며, TP와 IoK에서는 여러 GPU에 분산된 레이어의 연산 결과를 취합하기 위해, 각 레이어마다 GPU 통신이 요구된다. MP는 연산을 마친 결과만 후속 GPU에 전달하지만, TP 및 IoK는 모든 레이어 연산의 중간 결과 및 텐서들을 모두 통신으로 전달하기에 통신량이 훨씬 크다. 이로 인해 최대 292배의 높은 통신량을 보이게 된다.

상기 결과를 추가해석하기 위해, 측정한 네트워크 사용량을 GPU 코어 활용률과 함께 비교한다. GPU 코어 활용률은 상기 그림 4와 동일한 실험을 수행하는 상황에서, GPU의 코어들이 커널을 실행하지 않는 유휴한 시간의 비율을 측정한 것이다. 그림 5는 배치 사이즈가 변화할 때의 통신량(붉은 실선, ∗ 기호)과 GPU 활용률(검은 실선, ○ 기호)을 나타낸다. 먼저 통신량은 모든 병렬화, 최적화 전략에서 배치 사이즈의 증가와 함께 증가했다. MP의 경우 배치 사이즈가 16에 가까워질수록 약 0.085 GB/s에 수렴해가며, TP 및 IoK는 각각 약 6.7 GB/s, 6.5 GB/s에 수렴해간다. 반면 GPU 활용률은 배치 사이즈가 증가할 때 감소하는 양상을 보여준다. 통신량과 함께 고려해보면, 통신량과 GPU 활용률이 서로 반비례하는 결과를 확인할 수 있다.

이는 일반적으로 딥러닝 모델을 여러 GPU에 분산하

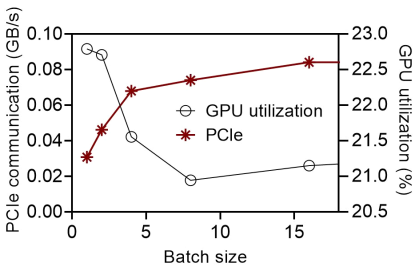


(a) 데이터 송신량(TX)

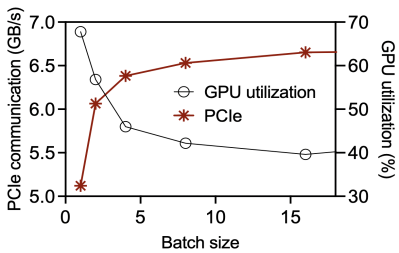


(b) 데이터 수신량(RX)

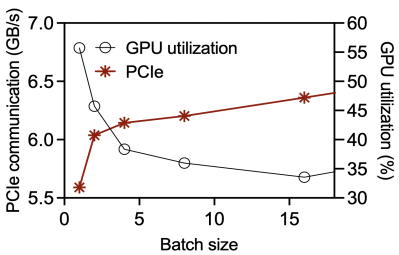
그림 4. 배치 사이즈 및 병렬화/최적화 전략에 따른 GPU 통신량
Fig. 4. GPU communication amount per batch size and parallelism/optimization techniques.



(a) 모델 병렬화 (model parallelism)



(b) 텐서 병렬화 (tensor parallelism)



(c) 커널 병합 (kernel fusion)

그림 5. 통신량 및 GPU 활용률 비교분석
Fig. 5. Comparison of communication and GPU utilization.

여 구동할 때는 네트워크 통신이 주요한 병목이 되기 때문이다[5]. 본 실험에서도, 특히 TP 및 IoK의 경우 통신량이 배치 사이즈가 커지면서 급증하게 되는데, 이는 TP를 사용하게 되면 레이어 간 중간 텐서 연산의 결과가 수시로 전달되며 배치 사이즈가 커질수록 전달되는 연산 결과량이 더 커지기 때문이다. 통신이 수행되는 동안에는 GPU가 후행 연산을 수행하지 않고 대기하게 되는데, 이 때문에 통신량이 늘어날수록 네트워크 전송 지연이 증가하며, 반대로 GPU는 유휴하는 기간이 길어져 활용률이 낮아진다.

배치 사이즈가 일정 크기 이상이 되면, 예를 들어 MP의 경우 8 이상이 되면, GPU 활용률이 증가하는 결과를 보인다. 이는 통신량 증가로 인한 병목 증가보다, GPU 연산 로드 오버헤드가 감소로 인한 이점이 더 크기 작용하기 때문이다. 다만 한정된 GPU 메모리 크기로 인해 과도하게 큰 배치 사이즈를 사용할 경우 메모리 부족(OoM) 에러가 발생하여 추론 자체가 불가능하다. 배치 사이즈에 따른 GPU 메모리 점유량 양상은 다음 항에서 다룬다.

4.5 GPU 메모리 점유량

그림 6은 배치 사이즈 별 MP, TP, IoK 최적화 기법의 GPU 메모리 점유량을 나타낸다. 총 4개의 GPU를 사용하여 추론하며, 그래프의 각 값은 4개의 GPU 메모리 점유량의 평균이다. 세 최적화 기법 모두에서 배치 사이즈가 커짐에 따라 연산 결과를 저장하는 텐서의 크기가 증가하며, 따라서 GPU 메모리 점유량 역시 증가함을 확인할 수 있다. 다만 IoK 기법의 경우 배치 사이즈가 32인 경우 메모리 부족(OoM) 에러를 피하기 위해 실험에 활용한 커널 병합 기법 자체가 LLaMA 모델이 생성하는 내부 토큰의 수를 23개로 제한하여 GPU 메모리 점유량을 줄인다. 하지만, 이로 인하여 배치 사이즈가 변하더라도 82% 수준으로 일정하게 유지되는 추론 정확도가 배치 사이즈 32에 도달하면 26% 수준으로 매우 크게 하락하는 문제점이 존재한다(표 2). MP, TP 기법의 경우 배치 사이즈가 64보다 클 때부터, IoK 기법의 경우 32보다 클 때부터 OoM 에러가 발생하며, 추론 구동이 불가능하다.

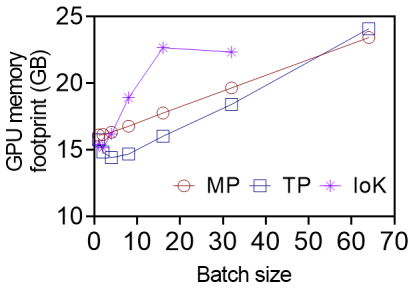


그림 6. GPU 메모리 점유량 실험결과
Fig. 6. GPU memory footprint results.

V. 결 론

본 연구는 LLM의 추론 시스템을 구성할 때 핵심적인 변수인 배치 사이즈가 증가함에 따른 추론 시간 및 정확도, 통신과 GPU 소모량을 분석하였다. 배치 사이즈의 증가로 인해 개별 추론 요청마다 GPU에 연산을 새롭게 로드하지 않고 일괄 처리할 수 있어 GPU의 유휴 구간을 줄이며 개별 추론의 지연을 감소시킬 수 있다는 이점이 있다. 그러나 우리의 실험 결과는 통신량의 증가로 인해 오히려 GPU가 유휴한 시간을 늘릴 수 있고, 추론 지연이 증가할 수 있다는 것을 보여준다. 또한 GPU 메모리 점유량 관점에서는 배치 사이즈가 클수록 한정된 메모리를 더욱 적극적으로 활용하나, 메모리 부족을 일으키거나 이를 피하기 위한 정확도 저하가 발생할 수 있어 배치 사이즈 선택을 신중히 해야 함을 알 수 있다. 이러한 결과를 기반으로 최적의 추론 시스템을 구성하기 위해서는 배치 사이즈, 병렬화 전략, 통신량, GPU 활용률, GPU 메모리 점유량 등 다양한 요소의 상호 작용을 복합적으로 고려해야 함을 알 수 있다.

References

[1] M. U. Hadi, et al., “Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects,” *TechRxiv*, Nov. 2023. (https://doi.org/10.36227/techrxiv.23589741.v4)

[2] Y. Wang, et al., “Tabi: An efficient multi-level inference system for large language models,” in *Proc. Eighteenth Eur. Conf. Comput. Syst.*, pp. 84-99, May 2023. (https://doi.org/10.1145/3552326.3587438)

[3] G. Yang, et al., “Prediction of the resource consumption of distributed deep learning systems,” in *Proc. ACM Measurement and Analysis of Comput. Syst.* vol. 6, no. 2, pp. 1-25, 2022. (https://doi.org/10.1145/3530895)

[4] C. Shin, et al., "Xonar: Profiling-based job orderer for distributed deep learning," *2022 IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, 2022. (https://doi.org/10.1109/CLOUD55607.2022.00030)

[5] D. Narayanan, et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. for High Performance Comput., Netw., Storage and Analysis*, 2021. (https://doi.org/10.1145/3458817.3476209)

[6] Z. Li, et al., "Terapipe: Token-level pipeline parallelism for training large-scale language models," *Int. Conf. Mach. Learn., PMLR*, 2021.

[7] Y. Zhuang, et al., "On optimizing the communication of model parallelism," in *Proc. Mach. Learn. and Syst.,* vol. 5, pp. 526-540, 2024.

[8] B. Wang, et al., "Tesseract: Parallelize the tensor parallelism efficiently," in *Proc. 51st Int. Conf. Parallel Process.*, pp. 1-11, Jan. 2022. (https://doi.org/10.1145/3545008.3545087)

[9] G. Chen and X. Shen, "Free launch: Optimizing GPU dynamic kernel launches through thread reuse," in *Proc. The 48th Int. Symp. Microarchitecture*, 2015. (https://doi.org/10.1145/2830772.2830818)

[10] *DeepSpeed Inference*(2021), Retrieved Mar., 30, 2024, from https://www.deepspeed.ai/2021/03/15/inference-kernel-optimization.html.

[11] R. Y. Aminabadi, et al., "Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale," *SC22: Int. Conf. High Performance Comput., Netw., Storage and Analysis, IEEE*, 2022. (https://doi.org/10.1109/SC41404.2022.00051)

[12] C. Shin, G. Yang, and C. Yoo, "An analysis of fine-grained GPU performance metrics on NVIDIA GPU accelerators," in *Proc. Symp. KICS*, pp. 492-493, 2022.

[13] *NVML API Reference Guide*(2023), Retrieved Jan., 10, 2024, from https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html.

[14] Z. Jiang, et al., "MegaScale: Scaling large language model training to more than 10,000 GPUs," *21st USENIX Symp. Netw. Syst. Design and Implementation (NSDI 24)*, 2024.

[15] S. Lee, et al., "On model parallelization and scheduling strategies for distributed machine learning," *Advances in NIPS*, vol. 27, 2014.

[16] H. Touvron, et al., "Llama: Open and efficient foundation language models," *arXiv preprint, arXiv:2302.13971*, 2023.

[17] Y. Yoo, et al., "Control channel isolation in SDN virtualization: A machine learning approach," *2023 IEEE/ACM 23rd Int. Symp. Cluster, Cloud and Internet Comput. (CCGrid)*, 2023. (https://doi.org/10.1109/CCGrid57682.2023.00034)

신 창 용 (Changyong Shin)

2021년 2월 : 고려대학교 컴퓨터학과 졸업

2021년 3월~현재 : 고려대학교 컴퓨터학과 석박통합과정

<관심분야> AI 시스템, 데이터센터 네트워킹

[ORCID:0000-0002-3323-1054]

고 영 훈 (Younghun Go)

2022년 8월 : 전북대학교 컴퓨터공학부 졸업

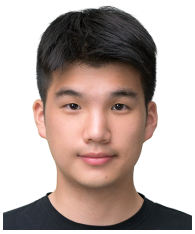
2024년 8월 : 고려대학교 컴퓨터학과 석사

2024년 9월~현재 : 고려대학교 컴퓨터학과 박사과정

<관심분야> AI 시스템, 운영체제

[ORCID:0000-0001-9293-4692]

유 연 호 (Yeonho Yoo)



2017년 2월 : 국민대학교 컴퓨터공학과 졸업
2021년 9월 : 고려대학교 컴퓨터공학과 석사
2024년 2월 : 고려대학교 컴퓨터공학과 박사
2024년 3월~현재 : 고려대학교 융합소프트웨어연구소 박사후연구원
<관심분야> 운영체제, 네트워크 시스템, AI 시스템
[ORCID:0000-0002-2636-633X]

양 경 식 (Gyeongsik Yang)

2015년 2월 : 고려대학교 컴퓨터학과 졸업
2017년 2월 : 고려대학교 컴퓨터학과 석사
2019년 8월 : 고려대학교 컴퓨터학과 박사
2023년 9월~현재 : 고려대학교 컴퓨터학과 조교수
<관심분야> 시스템 소프트웨어, 네트워크 시스템, AI 시스템, 데이터센터 인프라
[ORCID:0000-0003-4560-2972]

유 혁 (Chuck Yoo)

1982년 2월 : 서울대학교 전자공학과 졸업
1986년 8월 : University of Michigan At Ann arbor 석사
1990년 8월 : University of Michigan At Ann arbor 박사
1990년 8월~1995년 2월 : Sun Microsystems Lab 연구원
1995년 3월~현재 : 고려대학교 정보대학 교수
<관심분야> 운영체제, 소프트웨어 정의 네트워크, 디지털 헬스
[ORCID:0000-0002-1115-1862]