# GPU 프로파일러를 이용한 GCN 추론 모델의 특성 분석[1]

김인제○ 구건재
고려대학교 컴퓨터학과
sis013@korea.ac.kr, gunjaekoo@korea.ac.kr

# Revealing Characteristics of GCN Inference Models Using a GPU Profiler

InJe Kim○ GunJae Koo
Department of Computer Science and Engineering, Korea University

**요 약**

Graph Convolutional Neural Network (GCN)은 그래프 구조를 이용한 인공 신경망 (Graph Neural Network, GNN) 중의 하나로서 소셜 네트워크 분석 및 소비자 성향 분석, 추천 시스템 등의 여러 응용 분야에 이용될 수 있다. GCN은 비유클리드형 자료 구조의 형태를 가지는 그래프 구조에 대한 데이터 처리를 요구하기 때문에 기존의 Deep Neural Network (DNN)에 사용되는 데이터와는 차이점을 가지고 있다. 그렇기 때문에, GCN을 기존의 DNN을 처리하는 데에 주로 쓰이는 하드웨어 구조에서 실행했을 때의 특성을 분석하는 것은 향후 효과적인 GCN용 알고리즘 및 하드웨어를 설계하는 데에 있어서 필수적이다. 본 연구에서는 인공 신경망 구조를 처리하는 데에 주로 쓰이는 GPU에서 여러가지 GCN 추론 알고리즘을 실행하고 이를 GPU 프로파일러로 분석하여 해당 하드웨어 구조에서 GCN 추론 커널이 가지는 특성을 밝히고 있다. 본 연구에서는 GCN 추론 과정에 쓰이는 커널들이 크게 두 종류의 큰 차이를 보이는 특성을 보이는 커널들로 분류할 수 있음을 밝히고 있으며, 이러한 특성에 기반하여 GCN의 커널들이 GPU에서 실행 엔진과 캐시 메모리와 같은 하드웨어 자원을 비효율적으로 사용하고 있음을 밝혀내었다. 본 연구를 통하여 GCN의 최적화 방법 및 GCN을 효율적으로 실행하기 위한 구조적인 접근 방법에 대해서 도움을 줄 수 있다.

## 1. 서 론

그래프 구조는 데이터 분석 및 처리를 위한 대표적인 구조체 중의 하나로서 구성 요소 사이의 관계성을 vertex와 edge로 표현한다. 이러한 그래프 구조는 소셜 네트워크 및 상거래 사이트에서의 고객 니즈 분석, 추천 시스템, 분자 구조 분석 및 유전자 분석 등의 수많은 응용 분야에 사용되고 있다. 이러한 그래프 데이터 분석을 효과적으로 처리하기 위한 여러 그래프 처리 알고리즘 (graph processing algorithm)이 개발되어 왔으며 최근에는 이러한 그래프 처리를 인공 신경망 (neural networks)를 [1][3] 이용하여 처리하고자 하는 연구가 활발하게 이루어지고 있다. 그렇지만 그래프 구조는 기존의 심층 신경망 (deep neural networks, DNN)에서 사용되는 것과는 [2] 다른 형태를 가지고 있기 때문에 그래프 구조를 기존의 신경망 구조에 적용하기에는 어려운 점이 있다. 예를 들어, DNN의 경우 데이터의 특성을 추출하기 위해 합성곱 연산을 주로 사용하지만, 그래프 구조의 경우 구성 요소 사이의 관계성 정보를 합성곱 연산에 직접적으로 적용할 수가 없으며 기존의 신경망 연산에 주로 사용되는 행렬곱 연산을 이용하기 위한 데이터 전처리가 필수적이다. 또한 그래프 구조는 기본적으로 매우 큰 희소 행렬 형태로 표현되기 때문에 기존의 하드웨어 구조로는 효율적인 연산에 어려움이 따른다.

본 연구에서는 이러한 그래프의 신경망 처리, 즉 Graph Convolutional Neural Networks (GCN)의 추론 처리를 기존의 그래픽스 프로세서(graphics processing unit, GPU)에서 처리할 때의 특성을 분석하고 있다. 이를 위해서 본 연구에서는 GCN 알고리즘을 여러 형태의 데이터 세트를 사용하여 GPU에서 수행하고 GPU의 프로파일러를 사용하여 성능 측정값을 추출하였다. 이렇게 추출된 성능 측정값들을 분석하여 GCN의 추론에 사용되는 커널들의 서로 상반되는 특성을 밝혔으며, GCN의 추론 커널들이 GPU에서 수행될 때의 성능 저하 요소들을 밝혀내었다. 이를 통하여서 GCN을 기존의 하드웨어 구조에서 실행할 때의 최적화 방법 및 향후 GCN의 효율적인 처리를 위한 효과적인 하드웨어 구조 설계 방향에 대해서 제시를 하고 있다.

## 2. 배 경

그래프 구조는 구성 요소를 vertex (V)로 표현하고 구성 요소 사이의 관계를 edge (E)로 표현한다. 그러므로 그래프 구조 G = (V, E)라고 표현되며, 이때 V는 vertex들의 집합, E는 edge들의 집합이다. GNN은 그래프 구조에 신경망의 계층(layer)를 구현한 것으로서 각 N번째 layer는 아래와 같이 aggregation과 combination 단계로 나눌 수 있다.

[1]이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(Nos. 2018R1C1B5086594, 2021R1C1C1012172)

(1) $a_v^{(N)} = \text{Aggregate}^{(N)}(\{h_u^{(N-1)} \forall\ u \in N(v)\})$

(2) $h_v^{(N)} = \text{Combine}^{(N)}(h_v^{(N-1)}, a_v^{(NI)})$

위의 식에서 (1)의 aggregation 단계에서 $a_v^{(N)}$은 주변 이웃 노드들의 특성이 축약된 것이며, $h_u^{(N-1)}$은 노드 v의 이웃이 되는 노드의 특성이다. 또한 (2)의 combination 단계에서 $h_v^{(N)}$은 N번 반복한 노드의 표현이다. 즉, GCN은 이웃 노드로부터 특성 벡터를 받아 축약한뒤, multi-layer perceptron (MLP)를 통해서 각 노드를 업데이트한다.

본 연구에서는 위에서 밝힌 것과 같이 GCN의 핵심 연산 단계인 aggregation과 combination의 연산 특성에 집중하여 GCN을 기존의 병렬처리 하드웨어 구조에서 실행했을 때의 특성을 밝히는 데에 초점을 두었다.

표 1. 모델의 연산 방식과 출력채널 설정

| Algorithm | | Aggregation & Combination Operator |
|---|---|---|
| GCN | GraphSage(SAG) [1] | Mean & MLP-128 |
| | Graph Attention(GAT) | MLP-8H * 16 & Mean |
| | GraphSAINT(SAINT) | Sum & MLP-128 |
| | GCN | A(XW)-128 & Sum |
| CNN | Resnet-50 | Conv: 7x7x64 |

표 2. 실험에 사용된 그래프 데이터 셋 정보

| | Cora | Citeseer | Pubmed | Reddit |
|---|---|---|---|---|
| Density (%) | 0.18 | 0.11 | 0.028 | 0.21 |
| Vertex | 2708 | 3327 | 19717 | 232965 |
| Feature | 1433 | 3703 | 500 | 602 |

## 3. 분석 방법

이번 연구에서는 다양한 특성 길이를 가진 그래프 데이터셋들을 사용한다. 또한, 모델 간 성능을 신뢰성있게 비교하기 위해 입력데이터의 크기를 비슷하게 설정하였다. 실험은 Ubuntu 18.04에서 NVIDIA RTX 2080 GPU를 사용하여 Cuda Toolkit 10.2와 Pytorch 1.7.1 프레임워크에서 모델을 구현하여 실험하였으며 메모리를 효과적으로 사용하기 위해 그래프 데이터 셋을 희소행렬 표현 방식 중 하나인 COO(Co-Ordinate Format) 포맷을 사용해 기존방식인 노드$^2$가 아닌 2x엣지 의 크기로 인접행렬을 표현할 수 있다. 실험에서 사용한 GCN 모델인 GraphSAGE [1] 알고리즘은 기존의 GCN 알고리즘과 달리 전체 그래프 구조를 사용할 필요가 없는 귀납적 방식을 사용해 입력데이터 크기를 맞춰야 하는 이번 실험에 적절하다. CNN과의 특성 비교에 사용된 그래프 데이터셋은 Reddit 데이터셋을 사용했다. Reddit 데이터는 232,965개의 노드를 가지며 노드의 벡터 길이는 602, 엣지의 개수는 11,606,919개이며, 추론에 사용된 노드의 개수는 79,534개이다. 비교를 위한 CNN 알고리즘으로 Resnet-50 [2]을 사용하였으며 ImageNet 데이터를 입력 데이터셋으로 사용하였다. 위에서 언급하였듯이 서로 다른 알고리즘과 데이터셋을 사용하여 비교 실험을 진행해야 하므로 보다 정확한 비교를 위하여 입력 벡터의 크기를 비슷하게 설정하였다. 예를 들어 ImageNet의 이미지 데이터는 3x224x224인 반면 Reddit의 노드 하나 당 벡터의 길이는 602이므로, 입력 크기를 비슷하게 설정하기 위해 배치 크기를 각각 4와 1024로 설정하였다.

어플리케이션을 분석하기 위한 프로파일러로 NVIDIA Nsight compute 2019.5와 CUDA 10.2 버전의 NVIDIA Visual Profiler를 사용했다. Visual Profiler를 통해 전체 연산 동안 커널의 사용 비중을 파악과 API가 사용되는 타임라인을 시각적으로 확인했다. GCN 도메인에서의 하드웨어 특성을 추출하기 위해 표 2에 서술된 데이터셋에 대해 4개의 GCN 모델을 표 1의 설정 하에 Nsight compute profiler를 통해 각 커널 별로 GPU 리소스 사용 형태와 프로세서 및 메모리 사용량, 캐시 적중률, 지연 분석 등을 측정하였다. 또한 NVIDIA NVProf로 모델의 추론 타임라인을 추적해 연산 순서와 커널 별 비율을 밝혔다.

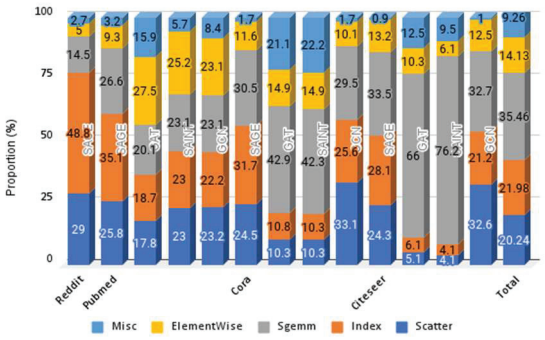


그림 1. 모델/데이터 별 커널의 시간분포

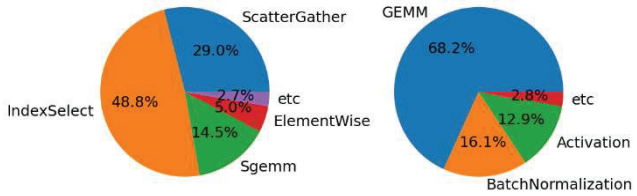


그림 2. Reddit 데이터셋에서의 GraphSAGE와 Resnet-50에 대한 실행 시간 분석

## 4. 분석 결과

### 4.1 분석 개요

그림 1는 GCN 모델 별 각 데이터 셋의 추론 작업에서 커널의 비중을 나타낸다. Sgemm커널은 가중치 행렬과 특성 벡터와의 연산이며 combination 단계에 속한다. IndexSelect와 ScatterGather 커널은 모든 노드에 대해 aggregation 단계를 적용한다. IndexSelect 커널은 이웃 노드 ID를 이용해 이웃 노드 특성 벡터를 가져와 Scatter 커널의 입력으로 사용될 수 있도록 구성한다. ScatterGather 커널은 이웃 특성 벡터들은 모델의 Aggregation Operator에 따라 각 이웃 노드에 대해 aggregation을 진행 후, update를 진행한다. Element 커널은 element-wise 연산에 해당되는 커널들이며 loop-unrolling이나 연산 전 mapping을 하는 역할을 한다. 그림 2. 에 나와 있듯이, GCN에 사용된 주된 3개의 커널은 전체 추론 시간에서 65% 에서 최대 92%까지 차지했다. 4개의 모델에서 데이터의 크기가 커질수록 Sgemm 커널의 비중이 낮아지며, Aggregation phase 관련 커널의은 증가하는 경향성을 확인했다. 인접행렬의 밀집도가 비슷할 때, 노드의 수가 증가할 수록 인접 노드의 수와 증가하기 때문에 인접배열의 크기또한 증가해 aggregation 단계의 시간이 증가한다. 하지만 각 데이터에 대해 시험 노드의 수가 고정되어 있어 데이터셋의 크기가 작아질 수록 모델 간 연산량의 차이를 확인 할 수 있다. 특성 벡터의 길이에 따라 Sgemm 커널의 비중변화하지 않으며, 커널 비중의 변화의 가장 큰 요인은 그래프의 크기임을 알 수 있다.

그림 2에서 볼 수 있듯이 aggregation 단계는 전체 실행시간의 약 77%를 차지한다. 그림 3의 SM과 memory작업 활용도를 통해 aggregation 단계에서 코어보다 메모리 작업의 비중이 높아 GPU를 활용도가 저하됨을 확인했다. 그에 반해 CNN의 경우 GEMM 커널의 비중이 68.2%로 행렬 연산이 주된 작업이다. 해당 커널의 활용도가 SM이 55.2%, memory가 48.4%로 균형있는

모습을 보여주어 부족한 자원활용률을 배치크기를 늘림으로써 쉽게 해결해 CNN 작업에서 GPU를 통한 주 커널의 효과적인 가속이 가능하다.

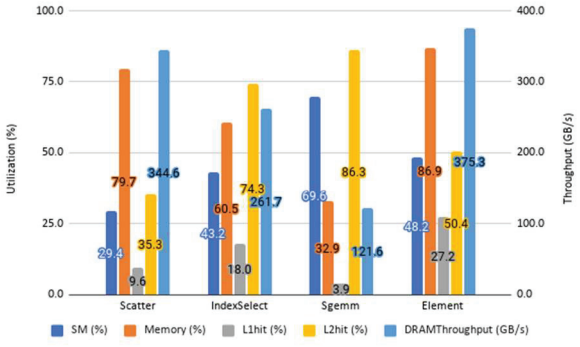


그림 3. GPU 상에서 커널 별 동작 특성

## 4.2 GPU 구조 상에서 GCN 작업 분석

그림 3은 각 커널에 대한 하드웨어 자원의 활용도를 모델/데이터의 평균값으로 나타낸 차트이다. Aggregation 단계에서 L1캐시 적중률이 각각 9.6%, 18%이며 L2 캐시 적중률도 80% 미만으로 데이터 지역성을 사용하지 못하고 있으며 이는 외부 메모리 접근을 크게 증가시켜 메모리 성능에 의한 병목현상을 발생시킨다. Aggregation의 낮은 캐시 적중률은 일반적인 행렬연산과 달리 ID를 통한 이웃 노드 특성 벡터를 가져오는 비정규적 데이터 접근과 각 노드마다 다른 이웃 노드를 갖고 있는 불규칙적인 작업에서 기인한다. 따라서 기존의 캐시 정책으로는 GCN 작업에 적합하지 않으며 많은 DRAM 처리량을 유발한다. GCN 작업에 대한 캐시 성능저하에 대한 증거로 CNN 추론 작업에서 가장 많은 비율을 차지하는 GEMM 커널의 캐시 적중률은 각각 89.2%, 93.9%이다.

Combination 단계의 경우, sgemm 커널에 대한 L1 캐시 적중률이 2%, L2는 86.65%이다. GCN 모델에서는 레이어 별로 가중치 값을 공유하므로 combination 단계는 aggregation 단계에 비해 정규적인 연산이 주를 이루어 L2 캐시 적중률이 상대적으로 높아 DRAM 접근도 aggregation 단계보다 낮다. 레이어마다 가중치 행렬이 공유되어 그 크기가 크지 않지만, 입력에 사용되는 행렬이 노드의 크기에 비례해 증가하기 때문에 입력 데이터가 L1 캐시에 맞지 않아 저장이 힘들어 L1 캐시 적중률은 aggregation 단계보다 낮다.

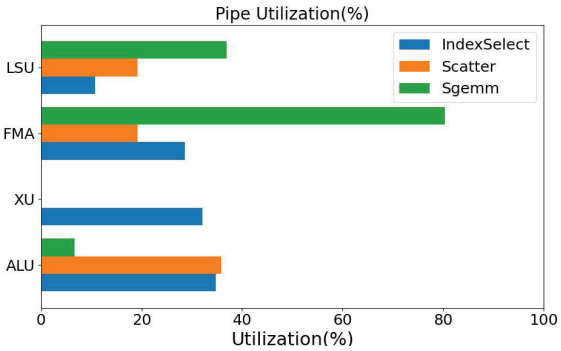


그림 4. 커널 별 파이프라인 활용도

## 4.3 스트림 프로세서(SM) 파이프라인 활용도

그림 4를 통해 GPU안의 SM(Streaming Multiprocessor) 내부에서 커널 별로 어떤 파이프라인 자원이 사용되는지 알 수 있다. Aggregation의 SM 활용도는 ALU가, Sgemm은 FMA가 주로 차지한다. Load Store Unit(LSU)는 메모리와 캐시에 대한 load/store 연산을, Floating point Multiply and Accumulate Unit(FMA)는 32-bit 부동소수점과 정수 연산의 대부분을 처리한다. Transcendental and Data Type Conversion Unit(XU)은 자료형 변환이나 특수 함수를 처리한다. Arithmetic Logic Unit(ALU)는 대부분의 정수 논리연산이나 비트 연산을 담당한다. Combination 단계의 Sgemm커널은 FMA의 활용도가 80%가 넘지만 실제 SM의 활용도는 70%가 되지 않으며, 그 외의 경우 모든 파이프라인에서 40% 이하의 활용도를 보여준다. Sgemm에 비해 높은 사용률을 보이는 파이프가 없음에도 낮은 활용성을 보여줌으로써 파이프가 지연되고 있음을 유추할 수 있다.

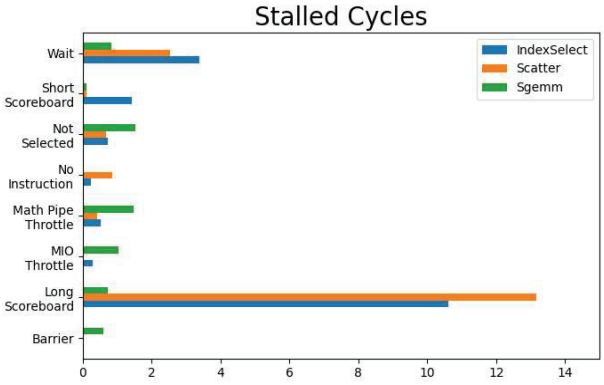


그림 5. 원인 별 지연 사이클

## 4.4 커널 별 지연 분석

그림 5는 SAG model에서 하나의 명령어가 이슈되었을 때, 워프 당 소요되는 cycle을 나타낸 것이다. 실험에서는 평균 18.93 사이클이 소요되었다. 하지만 총 사이클의 69.6%인 13.2사이클이 L1 캐시 의존성에 의해 지연되었기 때문에 큰 병목현상이 발생하였다. 해당 지연은 Aggregation 단계에서 발생한 것을 알 수 있다. Long Scoreboard stall이 L1 캐시 데이터 의존성을 나타내며 앞에서 분석한대로 캐시 성능의 저하로 인한 칩 외 메모리 접근이 많아 지연이 크게 발생했다고 분석할 수 있다. 즉, 전에 언급하였듯이 aggregation 단계에서의 비정규적 데이터 접근에 의해 기존의 GPU의 캐시 정책으로는 데이터 지역성을 이용하기 어려움을 알 수 있다.

## 5. 결 론

GCN 모델은 최신의 인공지능 모델 중 하나로 사회망을 이용하는 시스템에서 추천 알고리즘으로 사용하기 적절하며 여러 데이터셋과 모델에 적용해 하드웨어 특성의 일반화가 가능하며 그래프의 크기가 커질 수록 aggregation 단계의 비중이 커지는 경향성을 확인했다. 하지만 두 단계의 연산방식이 상극인 것에 의하여, 작업 분배가 적절히 되지 않고, 각 단계에서의 GPU활용 또한 제대로 되지 못하고 있는 것을 확인할 수 있다. 기존의 GPU 캐시로는 큰 가중치 행렬을 처리할 수 없어 L1 캐시를 적절히 이용하지 못했으며, aggregation 단계에서 비 정규적 데이터 접근으로 인해 캐시의 정확도가 저조해 발생된 많은 외부 칩 메모리 접근에 의해 병목현상이 발생함을 확인했다. 마지막으로 GPU내부에서 하나의 명령어에 대한 워프가 L1 캐시 의존성 문제로 전체 사이클 중 평균적으로 70% 가량 지연됨을 확인하였다.

이번 연구를 통해 GCN을 가속하기 위한 하드웨어 구조를 설계하고 할 때, 두 단계에 대한 적절한 작업량 분배, aggregation 단계에서의 데이터 접근 패턴 파악과 큰 크기의 데이터를 L1 캐시에서 재사용률을 높이면서 사용하기 위한 방법 등을 고려해야 함을 GPU 프로파일러를 활용해 병목현상을 여러 측면에서 실험적으로 분석하여 밝혀내었다.

## 참 고 문 헌

[1] William L. Hamilton, Rex Ying, and Jure Leskovec, "Inductive representation learning on large graphs," In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17) 2017.

[2] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016,

[3] Thomas N Kipf, et al., Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.