그래프 특성 벡터의 희소성에 따른 GCN 추론 커널의 성능 분석*

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

Analyzing the Performance of GCN Inferences with respect to Sparsity of Graph Features

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 중의 하나로 소셜 네트워크 분석 및 소비자 성향 분석 등 여러 응용 분야에 이용될 수 있다. GCN 커널은 크게 aggregation 과 combination 의 두 종류로 나눌 수 있다. Combination 은 일반 행렬 연산과 유사하여 GPU 에서 높은 성능을 낼 수 있다. 하지만 aggregation 커널은 불규칙적인 메모리 접근을 유발하는 간접 메모리접근을 많이 포함하고 있어서 GPU 구조에 적합하지 않다. 이러한 문제점을 다루기 위해 여러 가속기 연구에서는 aggregation 커널을 최적화하고 있지만 그래프 구조의 희소성에만 집중하고 있다. 이번 연구에서는 그래프 특성 벡터의 밀집도에 따른 GCN 커널의 성능 분석을 진행하여 특성 벡터 또한 압축된 희소행렬방식으로 처리하였을 때, 2 중 간접 주소 접근이 발생함에도 불구하고 sparse feature 에서 더 잘 동작하는 것을 확인했다. 또한 이번 연구에서 GCN 커널의 특성을 분석하여 GCN 커널을 GPU 에서 실행할 때의 성능 저하 요소를 밝히고 커널의 수행 시간을 줄일 수 있는 방법을 제시하고자 한다.

1. 서 론

그래프 구조는 데이터 분석 및 처리를 위한 대표적인 구조체 중의 하나로서 구성 요소 사이의 관계성을 노드와 연결선으로 표현한다. 또한 이런 그래프 구조를 활용한 Graph Convolutional Neural Network (GCN)은 소셜 네트워크 및 상거래 사이트에서의 고객 니즈 분석, 추천 시스템, 분자 구조 분석 및 유전자 분석 등의 수많은 응용 분야에 사용되고 있다. GCN 은 크게 일반적인 neural networks 에서 사용하는 Multi-Layer Perceptron (MLP)연산을 위한 combination 과 인접 노드로부터 feature vector 를 중심 노드로 모아 연산하는 aggregation phase 의 두 개의 phase 로 이루어져 있다. Combination phase 는 일반 matrix multiplication 으로 연산량이 많으며, aggregation phase 는 희소한 인접 행렬로 인해 Compressed Sparse Row (CSR)과 같은 희소행렬 표현법을 사용해 간접 메모리 접근이 많이 발생하며 feature vector 를 가져오는 특성으로 메모리 접근이 많은 phase 이다. 이러한 특징을 가진 GCN 을 효과적으로 가속하기 위한 연구가 활발하게 이루어지고 있다 [1][2]. 그렇지만 실제 그래프 구조의 데이터 셋들은 매우 다양하며 feature density 또한 매우 상이하다. 이러한 상이한 feature density 를 갖는 그래프 데이터 셋을 고려하지 않는다면, 에너지 효율적이고 높은 처리량을 가진 설계가 불가하다. 따라서 본 연구에서는 GCN 의 추론 처리를 담당하는 두 phase 의 커널을 일반 행렬 표현법과 희소행렬 표현법을 사용해 구현하여 그래픽스 프로세서 (Graphics Processing Unit, GPU)에서 처리했을 때, 데이터 셋에 따른 적절한 행렬 표현법을 찾고 있다. 이를 위해 본 연구에서는 GPU 의 프로파일러를 사용하여 커널의 이렇게 측정한 실행시간을 통하여 데이터 셋의 크기가 커질수록 aggregation 커널의 실행시간의 비중이 커지는 것을 밝혔으며, 커널들은 데이터 셋의 특성에 따라 알맞은 행렬 표현법이 존재함을 밝혔다. 특히 sparse 형태의 aggregation 커널은 2 중 간접 주소 접근이 발생함에도 불구하고 희소한 특성행렬을 가진 데이터 셋에서 더 잘 동작하는 것을 확인하였다. 이를 통해 향후 GCN 의 효율적인 처리를 위한 성능 저하 요소를 밝히고, 커널의 수행 시간을 줄일 수 있는 방법을 제시하고자 한다.

2. 배 경

GCN 을 일반화하면 그래프 구조에서 각 노드 간 관계를 표현한 인접행렬과 노드의 특성벡터로 이루어진 특성 행렬, 그리고 가중치 행렬의 세개의 행렬 연산이다. 인접행렬을 A, 특성 행렬을 X, 가중치 행렬을 W라고 했을 때, 레이어의 연산 결과로 X'를 계산하기 위한 식은 아래와 같다.

$$X' = AXW$$

그래프의 노드의 개수가 V, 특성 벡터의 길이를 F, 그리고 출력 채널의 길이를 H라고 했을 때, 인접 행렬 $A = V \times V$, 행렬 $X = V \times F$, 행렬 $W = F \times H$가 된다. 따라서, 이번 연구에서 다루는 두 커널 aggregation kernel 과 combination kernel 은 (AX) 와

* 이 성과는 2021 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2021R1C1C1012172)

(XW)로 나눌 수 있다.

본 연구에서는 GCN 의 중심이 되는 두 커널을 다양한 데이터셋에 대해 저장 형태를 달리한 프로그래밍 모델을 적용하여 전체 성능 비교를 진행하여 차후 GCN 가속기 연구를 위해 필요한 점에 대해 논한다.

표 1 실험에 사용된 그래프 데이터 셋 정보

	Number of Nodes	Feature Length	Feature Density (%)	Graph Density (%)
corafull	19793	8710	0.65	0.0374
coauthor-phy	34493	8415	0.39	0.0446
amazon-photo	7650	745	34.72	0.4250
amazon-com	13752	767	34.85	0.2728
reddit	232965	602	49.55	0.2100
yelp	716848	300	50.90	0.0028

3. 분석 방법

1) 분석 환경

이번 연구에서는 pytorch-geometric 에서 가져온 표 1 과 같이 희소한 인접행렬을 가지지만 다양한 특성(특성 길이, 특성 벡터 밀집도)들을 가진 데이터셋을 준비하여 데이터 특성에 따른 성능 비교를 진행한다. 실험은 Ubuntu 18.04 에서 NVIDIA RTX 2080 GPU 를 사용하여 Cuda Toolkit 10.2 환경에서 CUDA C/C++을 사용하여 Fast-GCN[3]을 기반으로 한 aggregation 커널과 combination 커널 전부 sparse 와 dense, 두 가지의 저장 형태를 가진 커널을 구현했다. 4 장에서 사용하는 실행시간 데이터는 NVIDIA Visual Profiler 를 사용했다. Visual Profiler 를 사용해 CPU 에서 측정하는 것보다 정확하게 전체 연산 중 실제 커널 실행시간 및 메모리 복사 시간을 타임라인으로 확인할 수 있다.

2) Combination 커널

Combination 커널의 출력채널의 크기 H를 128 로 설정하였다. Combination-Dense 커널은 일반적인 행렬 곱셈과 마찬가지로 GPU 에서 효율적으로 실행할 수 있다. 한 스트림 멀티프로세서(SM)에 할당할 수 있는 스레드 수인 1024 개로, 메모리 지연을 최소화하면서 연산을 하기 위해 특성 행렬의 일부와 가중치 행렬의 일부를 공유 메모리에 저장한다. Combination-Sparse 커널은 특성 벡터의 non-zero 값 만을 연산하므로 특성 벡터를 CSR 형태로 저장하며, 따라서 특성 행렬에 접근할 때에는 간접 메모리 접근이 발생한다.

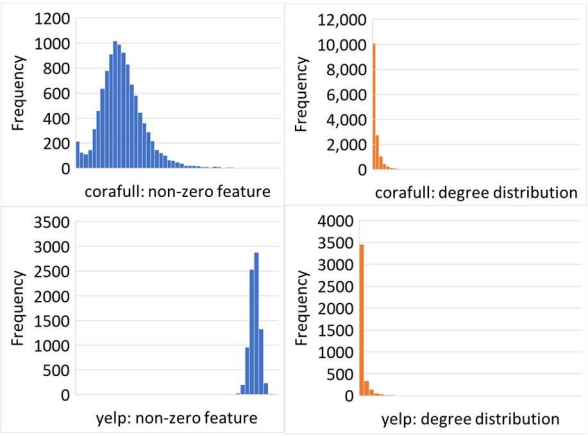

그림 1 Feature density (왼쪽)와 graph density (오른쪽)

3) Aggregation 커널

Aggregation 커널은 중심 노드에서 인접 노드로 접근한 후, 인접 노드의 특성 벡터를 가져와 연산한다. 인접 행렬과 특성 행렬 둘 다 접근하는 커널이기 때문에 2 중 루프가 발생한다. 그림 1 은 yelp 와 corafull 데이터 셋의 feature density 와 graph density 를 나타내는 히스토그램이다. Feature density 는 특성행렬에서 0 이 아닌 값인 요소의 비율이고, graph density 는 인접행렬에서 0 이 아닌 값인 요소의 비율이다. 그림 1 의 특성 행렬은 밀집하는 형태가 가우시안 분포(Gaussian Distribution)와 비슷한 형태를 보이되, 실제 밀집도는 데이터마다 다양하다. 하지만 인접 행렬의 경우 모든 데이터가 멱급수(power series) 형태를 띤다. 즉, 그래프의 중심 노드 대부분이 적은 인접 노드의 수를 갖는다. 따라서, 모든 인접 행렬이 매우 희소한 구조를 갖고 있기 때문에 희소 행렬 저장방식인 CSR 형태로 저장했다.

Aggregation-Dense 커널은 인접 노드에서 특성벡터를 특성길이만큼 가져와 연산하지만, Aggregation-Sparse 커널은 특성 행렬도 CSR 형태로 저장하여 인접 노드의 특성 벡터에서 0 이 아닌 값 만을 가져와 연산한다. 따라서 Aggregation-Sparse 커널에서는 2 중 간접 메모리 접근이 발생한다.

4. 분석 결과

1) Memory Transfer

그림 2 는 저장형태에 따른 커널의 호스트에서 GPU 메모리로 복사한 크기를 보여주고 있다. 'D'는 dense 커널을, 'S'는 sparse 커널을 의미한다. Dense 커널들의 경우, 특성 행렬을 그대로 저장하기 때문에 $V \times F$ 만큼 특성 행렬을 저장할 것이다. 그리고 CSR 형태로 특성 행렬을 저장한 sparse 커널의 경우 3 개의 배열을 사용하여 저장한다. 3 개의 배열의 크기는 각각 (V+1), (Number of non-zero elements), (Number of non-zero elements)으로, 행렬의 밀집도가 $(F-1)/2$ 이하일 때 dense 커널보다 효율적인 데이터 저장이 가능하다. 두 커널에서 저장형태에 따른 데이터 복사크기의 차는 같지만, aggregation 커널은 인접행렬 또한 복사를 하기 때문에 변화하는 비율이 다르다. Combination 커널에서 sparse 형태는 corafull 과 coauthor-phy 데이터 셋에서 dense 에 비해 각각 30 배, 38 배 이상 메모리 전송량이 감소하였으나, aggregation 에서는 총 데이터에서 특성 행렬의 비중이 줄어들었기 때문에 각각 메모리 전송량이 각각 1.97 배, 1.98 배 정도만 줄어들었다. amazon-photo 와 amazon-com, reddit 데이터 셋에 대해서도 전송 크기가 dense 커널에 비해 감소한 것을 확인할 수 있다. 즉, 희소행렬의 경우, sparse 커널을 사용하여 메모리 전송 시간 및 저장 공간을 감소시킬 수 있다. 하지만 데이터 셋의 밀집도와 커널에 따라 효과가 다르다.

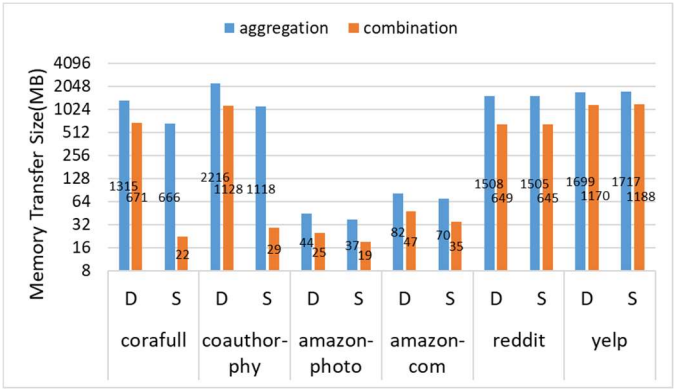

그림 2 커널의 CPU→GPU 로의 메모리 복사 크기 분포

2) Combination 커널

그림 3 은 NVIDIA Visual Profiler 를 사용하여 CPU->GPU memory transfer time 과 kernel execution time 을 측정한 것이다. 모든 경우에서 dense 커널이 sparse 커널보다 커널 실행시간

(kernel)이 짧은 것을 알 수 있다. 0% 밀집도의 데이터에서는 전송시간이 총 실행시간을 대부분 차지하여 희소행렬이 더 빠르지만 amazon-com 과 amazon-photo 의 경우에는 특성행렬의 밀집도가 35% 정도로 희소한 편이지만 커널의 실행시간에 의해 총 실행시간이 dense 커널보다 각각 약 35%와 22% 더 걸리는 것을 확인하였다.

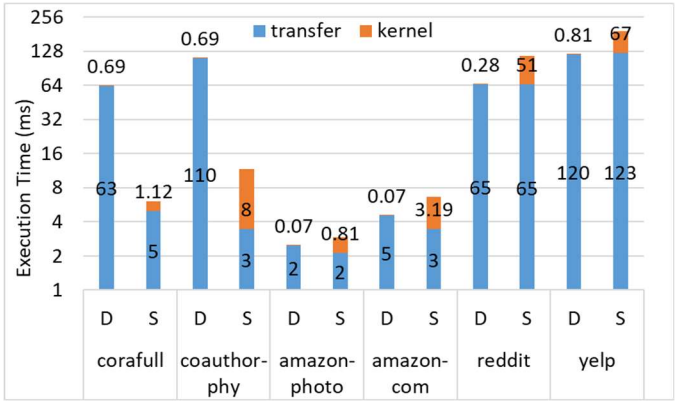

그림 3 Combination 커널의 데이터 및 저장 형태에 따른 총 실행시간

3) Aggregation 커널

Aggregation 커널의 경우, 그림 3 에 따라 combination 커널과 달리 인접 행렬도 추가적으로 사용되므로 dense 와 sparse 간 transfer size 의 차이가 combination 과 같더라도 비율이 감소한 것을 확인할 수 있다. 그림 4 를 통해 aggregation 커널의 총 실행시간에 대한 분포를 확인할 수 있다. corafull ~ amazon-com 의 4 개의 데이터 셋에 대해서는 sparse format 이 dense format 보다 빠른 총 실행시간을 보여주며, 50% 대의 feature density 를 갖는 reddit 과 yelp 데이터 셋에 대해서는 combination 커널에 비해 비슷한 총 실행시간을 보여준다 (reddit 의 경우, sparse 커널 사용에 의한 total runtime overhead 가 combination 에서 75%, aggregation 에서 8% 발생한다).

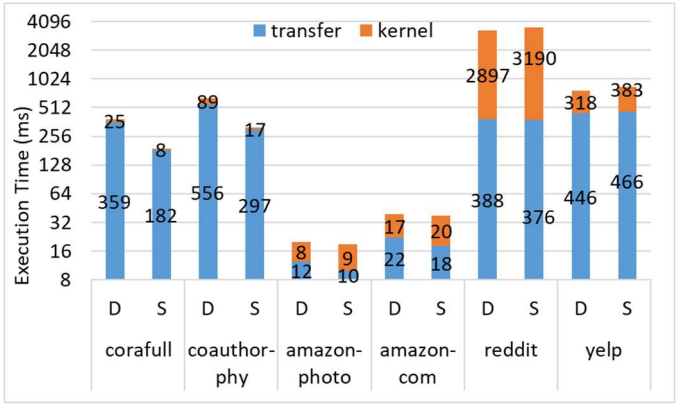

그림 4 데이터 및 저장 형태에 따른 총 실행시간

4) 주요한 실행 파트 선별 및 데이터에 따른 경향성 분석

그림 5 를 통해 데이터 셋에 따라 어떤 구간의 실행시간이 총 실행시간에 가장 큰 영향을 끼치는 지 알 수 있다. Reddit 데이터 셋이 가장 커널 실행비율이 높고, amazon-photo, amazon-com 와 yelp 데이터 셋 3 개가 비슷하며, coauthor-phy 와 coraful 이 차례로 작은 커널 실행비율을 보여준다. 오른쪽 그래프를 통해 aggregation 커널 실행비율은 그래프의 평균 인접 노드 수와 매우 밀접하다. 평균 인접 노드 수는 그래프의 중심 노드 수와 graph density 에 비례하므로, 인접행렬이 조밀하고, 노드 수가 많아질수록 aggregation 커널의 실행시간 비율이 증가한다.

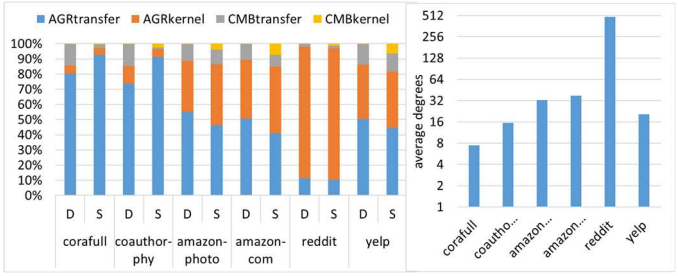

그림 5 Aggregation 커널과 combination 커널의 실행시간 분포도(왼쪽)와 데이터 셋 별 평균 인접 노드 수(오른쪽)

5. 결론 및 향후 연구

이 논문에서는 GCN 에서 주로 사용되는 커널을 두 가지 저장 형태로 구현하여 feature density 가 다른 데이터 셋에 대해 전체 실행시간을 측정하였다. 4-2)과 4-3)을 통해 각 커널에서 데이터 셋의 feature density 가 희소하면 간접 메모리 접근이 dense 커널보다 더 많음에도 sparse 커널이 더 실행시간이 짧은 것을 확인했다. 즉, feature vector 의 density 에 따라 다른 접근 방법이 존재한다. 또한, 4-4)에 의해 인접 행렬이 희소 행렬일지라도 그래프의 노드 수가 증가하여 매우 큰 그래프 구조를 갖게 되면 대부분의 실행시간은 combination 보다 aggregation 커널의 실행시간이 차지하게 됨을 알 수 있다. 따라서 GCN 어플리케이션을 효과적으로 가속하려면 aggregation 커널을 가속화하는 것이 가장 효과적이며, GPU 커널을 개선하려면 공유 메모리를 사용하거나 동시에 여러 인접 노드를 처리할 수 있도록 병렬성을 확장하는 방법 등이 있다.

GCN 에 사용되는 그래프 데이터는 사용자가 늘어남에 따라 점점 커지고 있으며 그래프 마다 특성 벡터의 밀집도는 상이할 것이다. 따라서 그래프 구조의 빅데이터를 효율적으로 가속하기 위해서는 데이터 셋의 특성에 따라서 접근 방법을 다르게 해야 한다.

참 고 문 헌

[1] J. Li, A. Louri, A. Karanth and R. Bunescu, "GCNAX: A Flexible and Energy-efficient Accelerator for Graph Convolutional Neural Networks," 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021.

[2] T. Geng et al., "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020.

[3] Chen, Jie, Ma, Tengfei et al., FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. ICLR, 2018.