GPU 기반의 그래프 엔진에서 메모리 할당 전략에 따른 성능 분석

정동규○ 장명환 김상욱†
한양대학교 컴퓨터·소프트웨어학과
{dkjeong91, sugichiin, wook}@hanyang.ac.kr

Performance Evaluation of A GPU-based Graph Engine with Different Memory Allocation Strategies

Dongkyu Jeong○ Myung-Hwan Jang Sang-Wook Kim
Department of Computer Science, Hanyang University

요 약

그래프 크기가 증가하면서 빅 그래프를 효율적으로 처리하기 위한 다양한 그래프 엔진들이 제안되었다. 그중 수천 개의 GPU 코어를 활용하여 빅 그래프를 처리하는 연구들이 진행되었다. 하지만 GPU 메모리만으로는 이러한 빅 그래프를 온전히 적재하기 어렵다는 한계점이 있다. 본 논문은 GPU를 활용하여 그래프를 분석할 수 있도록 구현된 RealGraph^{GPU}에서 사용 가능한 메모리 크기에 따른 세 가지 메모리 할당 전략들을 분석한다. 각 할당 전략에 따라 GPU$^{in-memory}$ 모드, CPU$^{in-memory}$ 모드, Disk 모드로 구현하고, 이를 실 세계 그래프를 사용하여 두 가지 알고리즘의 성능을 측정함으로써 분석한 바를 확인한다.

1. 서론

그래프는 웹, 소셜 네트워크, 인공 신경망 등 다양한 분야에서 객체 간의 관계를 표현하기 위해 널리 사용되는 자료구조이다. 그래프는 노드와 엣지로 구성되어 있으며 그래프 내에서 노드는 객체를 표현하고 엣지는 객체 간의 관계를 표현한다. 이러한 그래프를 분석하여 가치 있는 정보를 얻고자 많은 연구가 진행되어 왔다[1][2].

그래프 내 객체의 수가 증가하면서 그래프의 크기가 많이 증가하고 있다. 이러한 빅 그래프를 효율적으로 처리하기 위해 다양한 그래프 엔진들이 연구되었다[3][4]. RealGraph는 이러한 그래프 엔진 중 하나로서 빅 그래프를 처리하기 위해 SSD와 같은 보조 기억 장치를 활용하여 실 세계 그래프를 효율적으로 처리하는 그래프 엔진이다[5]. CuSha의 경우 GPU를 활용하여 빅 그래프를 더욱더 빠르게 처리하기 위한 그래프 엔진 연구 중 하나이다[6].

하지만 GPU 메모리만으로는 이러한 빅 그래프를 온전히 적재하기 어렵다는 한계점이 있다. 이러한 배경으로 실 세계 그래프 크기가 GPU 메모리 크기를 넘어서는 경우에도 효율적으로 처리하고자 하는 연구가 진행되었다[7][8].

본 논문은 GPU 기반으로 그래프를 처리할 수 있도록 구현된 RealGraph^{GPU}에서 GPU 및 CPU 메모리의 사용 가능한 크기에 따른 세 가지 메모리 할당 전략을 분석 및 구현하고 실험을 통해 분석한 바를 확인한다.

2. RealGraph

RealGraph는 대용량 데이터를 효과적으로 저장 및 검색을 할 수 있는 데이터베이스 스토리지 시스템의 설계 철학을 기반으로 설계되었다. 그림 1은 RealGraph의 구조를 보여준다. 총 4개의 레이어로 구성되어 있으며 각 레이어는 디스크 관리, 메모리 버퍼 관리, 멀티 스레드 관리 등 서로 다른 역할을 수행한다. 그래프 데이터가 저장되는 공간은 표준 I/O 유닛과 일치하는 블록 단위로 분할된다. 한 블록은 하나 이상의 오브젝트들이 저장되며, 각 오브젝트는 노드와 해당 노드의 엣지들로 구성되어 있다. RealGraph는 CPU 기반과 GPU 기반의 두 가지 버전이 구현되어 있다.

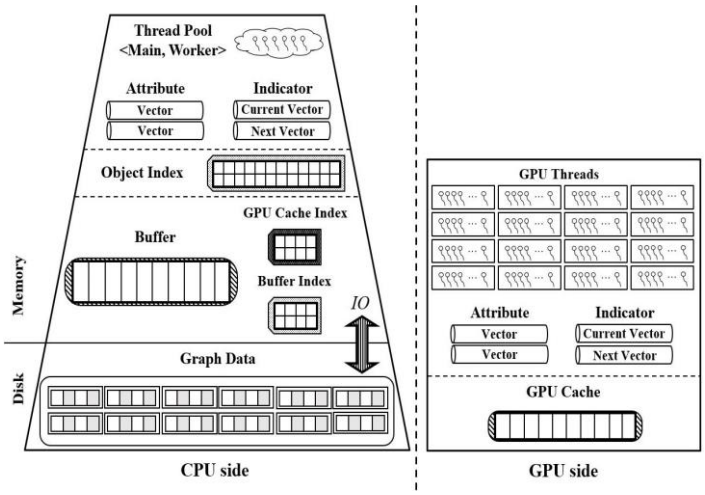

그림 1 RealGraph의 구조

† 교신저자.

그래프 알고리즘 수행 시 RealGraph는 블록 단위로 저장된 그래프 데이터와 입력 애트리뷰트 값을 이용하여 계산된 결과를 출력 애트리뷰트에 업데이트한다. 이때 애트리뷰트는 그래프 알고리즘 수행 시 노드가 가지는 특성값을 의미하며 노드 수만큼 존재한다.

3. 메모리 크기에 따른 할당 전략

RealGraph^{GPU}를 활용하여 Disk에 존재하는 데이터를 처리할 때 GPU로 데이터 전송이 이루어진다. 데이터 전송 시 데이터 처리 못지않게 많은 클럭을 요구하기 때문에 GPU와 CPU의 사용 가능한 메모리 크기에 따라 적절한 메모리 할당 전략이 필요하다. 이를 위해 우리는 GPU 및 CPU 메모리상에 애트리뷰트의 적재 가능 여부에 따라 세 가지 메모리 할당 전략을 구상하였다.

첫째, 애트리뷰트 크기가 GPU 메모리보다 작은 경우 애트리뷰트를 CPU 메모리 혹은 Disk에 두고 매 이터레이션마다 전송해서 처리하는 것보다 애트리뷰트를 GPU 메모리에 적재한 상태로 처리하는 것 (i.e., GPU$^{in-memory}$ 모드)이 최초, 최종 이터레이션 총 2회만 애트리뷰트 전송을 수행하기 때문에 데이터 전송 오버헤드가 가장 적다. 그러나 GPU 메모리에 적재되지 못할 정도로 큰 그래프는 처리하지 못하는 단점이 있다.

둘째, 애트리뷰트 크기가 GPU 메모리보다 크지만, CPU 메모리보다 작은 경우 Disk로부터 애트리뷰트를 읽어 처리하는 것보다 애트리뷰트를 CPU 메모리에 적재하고 필요한 일부만 GPU 메모리에 전송하여 처리하는 것 (i.e., CPU$^{in-memory}$ 모드)이 Disk로부터 전송 오버헤드를 발생시키지 않아 데이터 전송 오버헤드가 가장 적다. 그러나 CPU 메모리에 적재되지 못할 정도로 큰 그래프는 처리하지 못하는 단점이 있다.

셋째, 애트리뷰트 크기가 GPU 및 CPU 메모리보다 큰 경우 Disk에서 필요한 애트리뷰트의 일부만 CPU 메모리를 거쳐 GPU 메모리로 전송하여 처리하는 것 (i.e., Disk 모드)이 데이터 전송 오버헤드가 가장 적다. 다른 두 모드에 비해 매 이터레이션마다 Disk에서 CPU 메모리를 거쳐 GPU 메모리로 애트리뷰트 전송이 일어나지만, CPU 메모리 크기를 넘어서는 빅 그래프도 처리할 수 있다는 장점이 있다.

4. 평가 및 분석

본 장에서는 RealGraph^{GPU} 환경에 앞에서 언급한 메모리 크기에 따른 할당 전략 세 가지를 구현하고 수행 시간을 측정하여 결과를 분석하였다. 실험은 Intel i7-8770k, Samsung SSD 860 PRO 1TB, 64GB RAM, NVIDIA TITAN Xp 12GB, Ubuntu 18.04.2 LTS 64bit 환경에서 수행하였다. 실험 파라미터로 CPU 메모리는 32GB, GPU 메모리는 12GB로 설정하였다.

실험 데이터로는 표 1에 기입된 실 세계 그래프들을 사용하였다. 실험에 사용하는 세 가지 데이터셋은 애트리뷰트 크기가 GPU 메모리 크기보다 작은 크기를 가진다. 하지만 Yahoo 데이터셋의 크기보다 큰 데이터셋의 경우 애트리뷰트 크기가 GPU 메모리 크기를 넘어갈 것이 농후하다.

표 1 실 세계 그래프 데이터셋

Datasets	Twitter	Friendster	Yahoo
# of nodes (M)	61	68	1,413
# of edges (B)	1.4	2.5	6.6
Size (GB)	24	44	114

알고리즘으로는 그래프 처리에 폭넓게 사용되고 있는 SpMV[1]와 PageRank[2] 두 가지를 사용하였다. SpMV는 희소 행렬과 벡터 간 곱셈을 뜻하며 PageRank는 그래프 네트워크에서 노드가 가지는 중요도를 측정하기 위해 제안된 알고리즘이다. PageRank의 이터레이션 회수는 10회로 설정하였다. 각 모드별 전송 오버헤드를 측정하기 위해 GPU$^{in-memory}$ 모드는 GPU에 전체 애트리뷰트를 적재하여 처리하고, CPU$^{in-memory}$ 모드는 CPU에 전체 애트리뷰트를 적재하여 처리에 필요한 일부만 GPU로 전송하여 처리하고, Disk 모드는 Disk에 존재하는 애트리뷰트의 일부만 CPU 메모리를 거쳐 GPU 메모리로 전송하여 처리하도록 하였다.

그림 2와 그림 3은 각각 SpMV와 PageRank 알고리즘의 수행 결과를 보여준다. 두 그림에서 x축은 각 데이터셋을 나타내고 y축은 수행 시간을 나타낸다.

그림 2에서 전송 오버헤드로 인한 차이가 크지 않았다. 이는 SpMV 알고리즘은 이터레이션 1회만 수행되기 때문에 각 모드에서 GPU 메모리로 같은 수치의 데이터 전송이 일어났기 때문이다. 먼저 모든 데이터셋에서 GPU$^{in-memory}$ 모드와 CPU$^{in-memory}$ 모드의 수행 시간이 비슷한 이유는 앞에서 설명했던 것과 같이 이터레이션 1회로 두 모드의 전송량이 같기 때문이다. 반면에 Friendster와 Yahoo 데이터셋에서 CPU$^{in-memory}$ 모드와 Disk 모드의 수행 시간에 차이가 존재하는데 이는 데이터셋의 크기가 증가하면서 Disk-CPU 간의 전송 오버헤드가 크게 증가하였기 때문이다.

그림 3에서 PageRank 알고리즘은 이터레이션을 10회 수행하기 때문에 데이터 전송 오버헤드 차이가 크게 나타났다. 실험 데이터 중 가장 큰 Yahoo 데이터셋의 경우 모드별 최대 전송 오버헤드가 약 3.4배까지 차이가 났다. GPU$^{in-memory}$ 모드의 경우 최초, 최종 이터레이션에서의 전송을 제외하면 전송 오버헤드가 발생하지 않기 때문에 모든 데이터셋에서 가장 빠른 성능을 나타냈다. CPU$^{in-memory}$ 모드의 경우

매 이터레이션마다 CPU-GPU 간의 전송 오버헤드가 발생하기 때문에 $GPU^{in-memory}$ 모드에 비해 수행 시간이 증가하였다. Disk 모드의 경우 매 이터레이션마다 Disk-CPU-GPU 간의 전송 오버헤드가 발생하여 수행 시간이 가장 오래 소요되었다. 특히 Yahoo 데이터셋의 경우 데이터 크기가 가장 크기 때문에 전송 오버헤드 또한 그만큼 크게 증가하였다.

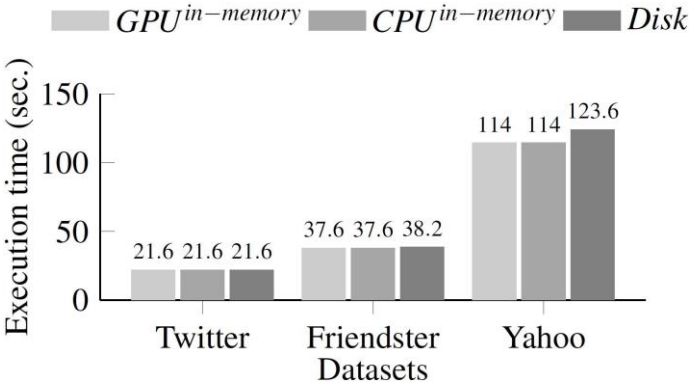

그림 2 SpMV 알고리즘 성능 평가

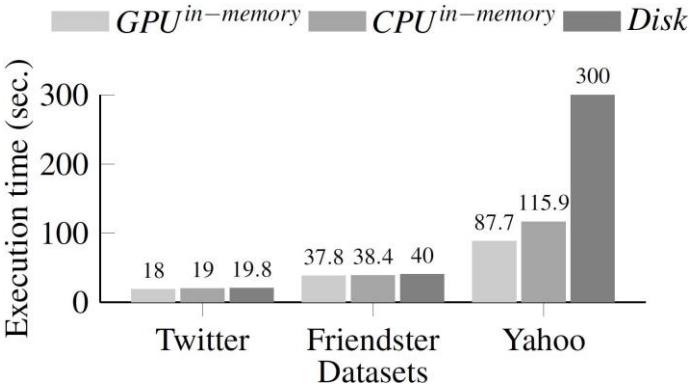

그림 3 PageRank 알고리즘 성능 평가

5. 결론

본 논문에서는 $RealGraph^{GPU}$에서 GPU를 효율적으로 활용하기 위해 GPU 및 CPU 메모리의 사용 가능한 크기에 따른 메모리 할당 전략 세 가지를 분석하고 이들을 구현하였다. 실험 결과 $GPU^{in-memory}$ 모드가 Disk 모드에 비해 최대 3.4배까지 좋은 성능을 내는 것을 확인했다. Disk 모드는 Disk I/O 전송 오버헤드로 인해 나머지 두 모드보다 성능이 느리지만, CPU 메모리에 적재되지 않는 빅 그래프를 처리 할 수 있는 장점이 있다.

6. 사사

본 연구는 삼성전자의 산학협력과제인 'Flash 기반의 그래프 빅 데이터 처리 플랫폼 기술 개발'의 일환으로 수행하였고, 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-
차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행되었음(No. NRF-2017M3C4A7069440).

참 고 문 헌

[1] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on GPUs: implications for graph mining," in *Proc. of the Very Large Data Bases Endowment*, vol. 4, No. 4, pp. 231-242, 2011.

[2] L. Page et al., "The PageRank Citation Ranking: Bringing Order to the Web," in Technical Report Stanford InfoLab, 1999.

[3] Y. Low et al., "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," in *Proc. of the Very Large Data Bases Endowment*, vol. 5, No. 8, pp. 716-727, 2012.

[4] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-Scale Graph Computation on Just a PC," in *Proc. USENIX Symp. on Operating Systems Design and Implementation*, pp. 31- 46, 2012.

[5] Y. Jo et al., "RealGraph: A Graph Engine Leveraging the Power-Law Distribution of Real-World Graphs," in *Proc. ACM Int'l Conf. on World Wide Web*, pp. 807-817, 2019.

[6] F. Khorasani et al., "CuSha: Vertex-Centric Graph Processing on GPUs," in *Proc. ACM Int' Symp. on High-performance parallel and distributed computing,* pp. 239-252, 2014.

[7] A. Gharaibeh et al., "Efficient large-scale graph processing on hybrid CPU and GPU systems," in Technical Report arXiv, 2014.

[8] M. Kim et al., "GTS: A Fast and Scalable Graph Processing Method based on Streaming Topology to GPUs," in *Proc. ACM Int'l Conf. on Management of Data,* pp. 447-461, 2016.