빅 데이터를 위한 행렬 곱셈의 성능 분석

권일택*, 조용연**, 김상욱§**
*한양대학교 컴퓨터공학부
**한양대학교 컴퓨터 소프트웨어학과
e-mail : {itkwon, jyy0430, wook}@hanyang.ac.kr

Performance Analysis of Matrix Multiplications for Big Data

Il-Taek Kwon*, Yong-Yong Jo**, Sang-Wook Kim**
*Dept. of Computer Science, Hanyang University
**Dept. of Computer &Software, Hanyang University

요 약

행렬 곱셈은 다양한 사회연결망을 포함한 빅 데이터 분석에 핵심이 되는 연산 중 하나이다. 본 연구에서는 행렬 곱셈 방법 중 내적과 행-행 곱셈에 대한 성능 분석과 실제 사회연결망 데이터 셋을 이용한 행렬 곱셈 시간을 분석한다. 본 연구의 실험환경에서 행렬 곱셈 방법 중 행-행 곱셈이 내적보다 약 125 배 빠르다는 것을 확인했고, 실제 사회연결망 데이터 셋을 행렬 곱셈했을 때의 시간은 읽기, 쓰기 등 저장장치 접근 시간이 행렬 곱셈 전체 수행 시간의 약 90% 이상 차지한다는 것을 확인했다. 따라서 사회연결망 데이터 분석을 위한 행렬 곱셈에서 저장 장치 접근 시간을 줄이는 것이 전체 계산 수행 시간을 줄이는 것의 핵심임을 이야기한다.

1.서론

사람들 간의 연결을 도와주는 트위터나 페이스북 같은 SNS 서비스가 대중화되면서 이에 대한 연구도 많이 진행되고 있다. 이러한 SNS 서비스의 대중화는 사람들을 연결하며 SNS 를 이용하는 사람들 수만큼 대규모의 사회연결망을 형성하게 되었다. 그리고 요즘은 이를 마케팅으로 이용하기 시작하면서 많은 가치가 창출되었고 이 때문에 사회연결망 분석은 시장에서 그 어느 때보다도 중요해졌다.

사회연결망은 사용자를 노드로, 사용자 간의 관계를 간선으로 표현하여 그래프로 표현할 수 있다. 이 때, 사회연결망에는 다음과 같은 특징이 있다. 첫째, 사회연결망을 행렬로 표현하면 희소행렬로 나타내어진다. 둘째, 빅 데이터로 구성되어 있다. 앞서 말한 메모리의 낭비를 줄이기 위해 사용하는 특별한 형태의 행렬 표현 방식으로 메모리에 차지하는 공간을 줄이더라도 사회연결망 데이터를 한 번에 메모리에 적재하지 못하는 경우가 많다. 따라서 계산 과정에 저장장치 접근이 필요하다.

사회연결망에서 유의미한 정보들을 추출하기 위한 여러 가지 기술들이 존재하는데, 이때 핵심이 되는 연산 중 하나는 행렬 곱셈이다. 행렬을 곱하는 방법은 다양하지만, 주로 행렬곱셈에 많이 사용되는 방법은 내적과 행-행 곱셈이 있다 [1, 2].

연구에서는 이러한 행렬곱셈 방법을 이용한 사회연결망 데이터의 행렬 곱셈을 수행하여 내적으로 수행했을 때의 시간이 행-행 곱셈으로 수행했을 때의 시간의 약 125 배 걸림을 확인했다. 그래서 본 연구에서는 행-행 방법을 사용하기로 했다. 그리고 행-행 곱셈을 통한 사회연결망 데이터의 행렬 곱셈 수행 시간 결과를 통해 전체 계산 수행 시간에서 저장장치 접근 시간의 비율이 90% 이상 차지한다는 것을 확인했다.

본 연구에서는 이처럼 사회연결망 분석을 위한 행렬 곱셈을 할 때 저장장치 접근 시간을 줄이는 것이 전체 계산 수행 시간을 줄이는 것의 핵심임을 논의한다.

2. 관련 연구

2.1. 행렬의 표현 방법

희소 행렬의 효율적 표현 방법으로 ELLPACK(ELL), Compressed Sparse Row(CSR), Compressed Sparse Column(CSC), Coordinate(COO) Format 등이 있다. 이들 모두는 행렬에서 0 이 아닌 값들만 저장한다는 공통점이 있다 [1]. 본 논문에서는 실제 행렬 곱셈에서 사용되는 다음 두 가지를 소개한다.

본 연구에서는 0 이 아닌 값들이 행, 열, 데이터 순으로 저장되어 있는 사회연결망 데이터 셋을 사용했고 이를 읽어서 활용하기 용이하도록 COO Format 으로 변환하여 저장한다. COO Format 은 0 이 아닌 값들

본 연구는 (1) 한양대-삼성전자 반도체 산학협력 연구과제와 (2) 미래창조과학부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업 (NIPA-2013-H0301-13-4009)의 연구 결과로 수행되었음.
§교신저자

의 행, 열, 데이터 값을 각각 배열에 저장한다 [1, 2].

$$A = \begin{bmatrix} 3 & 0 & 0 & 9 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 \\ 4 & 0 & 6 & 5 \end{bmatrix}$$

row = [0 0 1 1 2 2 3 3 3]
col = [0 3 0 2 2 3 0 2 3]
data = [3 9 1 2 1 1 4 6 5]

그림 1. 행렬 A 의 COO Format

그리고 본 연구에서 사용하는 행렬 곱셈 방법인 행-행 방법의 효율적인 사용을 위해 COO Format 으로 입력 받은 데이터를 희소행렬을 효율적으로 저장하는 방법 중 하나인 CSR Format 으로 변환한 후 곱셈을 수행한다. CSR Format 은 그림 2 와 같이 행렬 A 는 그 행렬의 0 이 아닌 값들만 data 배열에 저장하고, 그 저장된 값들의 행렬의 열 상의 위치를 표시하기 위해 0 이 아닌 값들의 열 index 를 저장하는 indices 배열, 각 행마다 그 전 행까지 0 이 아닌 값들이 얼마나 저장되어있는가를 표시하기 위한 ptr 배열을 이용한다 [1, 2].

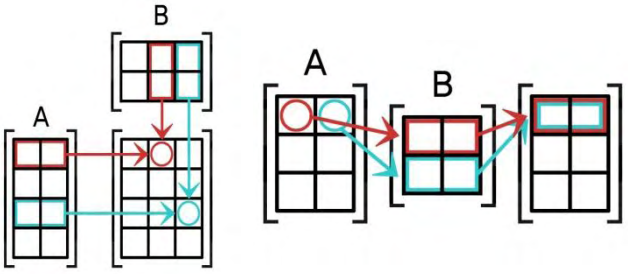
$$A = \begin{bmatrix} 3 & 0 & 0 & 9 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 \\ 4 & 0 & 6 & 5 \end{bmatrix}$$

ptr = [0 2 4 6 9]
indices = [0 3 0 2 2 3 0 2 3]
data = [3 9 1 2 1 1 4 6 5]

그림 2. 행렬 A의 CSR Format

2.2. 행렬의 곱셈 방법

그림 3 은 행렬 곱셈 방법들을 나타낸다. 그림 3-(a)의 내적을 나타내며, 가장 일반적인 행렬의 곱셈 방식이다 [3]. 그림 3-(b)는 행-행 곱셈을 나타낸다 [4]. 행-행 곱셈 방법은 행렬 A 의 하나의 행 내의 하나의 원소와 그 원소의 열 번호와 대응되는 행렬 B 의 하나의 행 내의 모든 원소와 곱셈을 수행한다. 이렇게 행렬 A 의 하나의 행 내의 모든 원소가 행렬 B 의 원소와 계산하여 하나의 행을 생성한다 [2].



(a) 내적 (b) 행-행 곱셈

그림 3. 행렬 곱셈 방법

3. 실험

본 연구에서는 사회연결망 분석을 위해 행렬 곱을 수행하여 수행 시간을 분석한다. 실험 환경으로 CPU 는 Intel i7-4770 3.40GHz 이고 메모리 용량은 32GB, 운영체제는 Windows 7 을 사용하였다.

실험에 사용된 데이터는 실제 사회연결망 데이터 2 가지를 사용하였다 [2]. Amazon 데이터는 간선 925,872 개, 노드 334,863 개로 구성되어 있으며, Amazon 웹 사이트에서 수집한 데이터 셋이다. 각 노드는 Amazon 에서 판매되고 있는 제품을 나타내고, 각 간선은 동시에 구매된 제품간의 관계를 나타낸다. DBLP 데이터는 간선 1,049,866 개, 노드 317,080 개로 구성되어 있으며, DBLP 에서 수집된 데이터 셋이다. 각 노드는 DBLP 의 저자를 나타내며, 각 간선은 두 저자간의 관계를 나타낸다.

3.1. 내적과 행-행 곱셈의 시간 비교

본 실험은 두 곱셈 방법을 이용하여 행렬 곱셈의 수행 시간을 비교해보기 위한 실험이다. 실험 방법은 데이터 셋 DBLP 로 동일한 행렬 2 개를 만든 후, 이를 곱하며 계산 시간과 저장장치 입출력 시간을 포함한 행렬 곱의 전체 수행 시간을 측정한다. 본 실험에서는 두 행렬 곱셈 방법의 성능을 알아보기 위한 실험이므로 행렬 곱의 계산시간만을 측정하였다.

그림 4 는 행-행 곱셈과 내적의 행렬 곱셈 전체 수행 시간을 보여준다. x 축은 각 곱셈 방법, y 축은 각 방법의 계산시간을 나타내며, 초단위로 표기하였다.

희소행렬에는 원소가 매우 적기 때문에, 내적으로 계산할 경우, 두 원소의 곱셈이 가능한지에 대한 판단 과정이 필요하다. 반면 행-행 곱셈은 행렬 A 의 하나의 원소가 곱셈할 행렬 B 의 행이 주어진다면, 판단 과정이 필요 없이 모든 원소와 곱셈을 수행하면 된다. 실험결과, 내적이 행-행 곱셈에 비해 약 125 배의 시간이 더 걸렸다. 따라서 사회연결망 분석을 위한 행렬 곱셈 수행 시 내적보다 행-행 곱셈의 성능이 뛰어남을 알 수 있다. 이후 실험에서는 행-행 곱셈으로 실험을 수행했다.

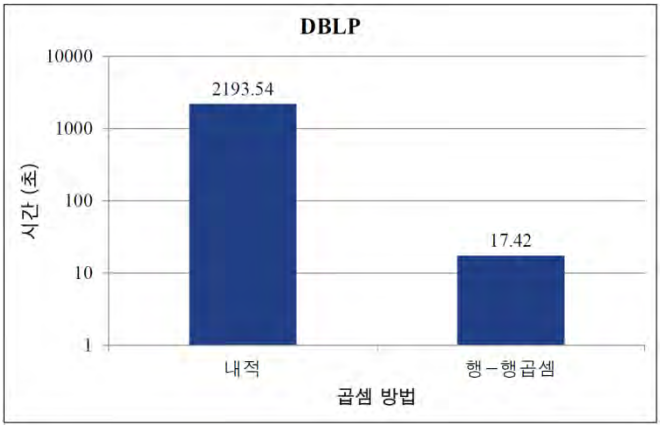


그림 4. 내적과 행-행 곱셈 전체시간 비교

3.2. 행-행 곱셈에서 읽기, 쓰기 시간의 비율

본 실험은 행-행 곱셈에서 전체 수행 시간에서 읽기 시간, 계산 시간과 쓰기 시간의 비율로 나누어 보기 위한 실험이다.

그림 5 는 행-행 곱셈으로 데이터 셋 DBLP, Amazon 에 대해 행렬 곱셈을 수행했을 때 각 요소 별 시간 분포를 나타낸다.

DBLP 의 행-행 곱셈 수행시간은 데이터를 읽는 시간이 약 3.47 초, 계산한 결과를 저장 장치에 쓰는 시간이 약 12.85 초, 계산 시간이 약 0.89 초였고, 쓰기 시간, 읽기 시간, 계산 시간의 백분율은 각각 약 74.66%, 20.16%, 5.19%로 읽기, 쓰기를 포함한 저장장치에 접근하는 시간은 계산 시간의 약 18.3 배로 측정됐다.

Amazon 의 행-행 곱셈 수행시간은 데이터를 읽는 시간이 약 3.09 초, 계산한 결과를 저장 장치에 쓰는 시간이 약 5.69 초, 계산 시간이 0.83 초였고, 쓰기 시간, 읽기 시간, 계산 시간의 백분율은 각각 약 59.15%, 32.18%, 8.66%로 읽기, 쓰기를 포함한 저장장치에 접근하는 시간은 계산 시간의 약 10.5 배로 측정됐다. 종합적으로, 행렬 곱셈 수행 시간의 약 91%가 저장장치 접근에 쓰이고 있음을 알 수 있다.

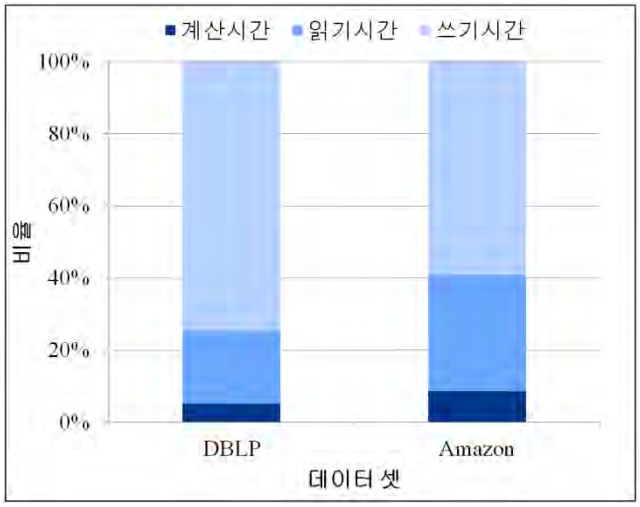


그림 5. 행-행 방법에서 각 시간들의 비율

3.3. SimRank 계산에서 읽기, 쓰기 시간의 비율

본 실험은 사회연결망 분석에 많이 쓰이는 어플리케이션 중 하나인 SimRank 계산에서 읽기 시간, 계산 시간, 쓰기 시간의 비율을 보기 위한 실험이다.

SimRank 는 행렬 곱을 이용하여 두 노드 간의 유사도를 측정하는 방법이다. SimRank 는 3 개의 행렬 곱과 1 번의 덧셈을 반복하여 일정한 값으로 수렴하는 유사도 행렬을 구하는 방법이다.

SimRank 의 계산과정은 수식 1 과 같다. S는 유사도 행렬, W는 인접 행렬, I는 단위행렬, c는 덤핑 팩터일때 k번째 유사도 행렬 S^k는 수식 1 로 표현할 수 있다[5].

$$S^k = cW^T S^{k-1} W + (1-c)I \quad \text{(수식 1)}$$

표 1 은 행-행 방법으로 SimRank 계산을 수행했을 때 각 요소 별 시간 분포를 나타낸다. Amazon 데이터는 행렬 곱의 결과 행렬의 크기가 실험 환경을 초과하여 실험을 마칠 수 없었다. 따라서 본 실험에서는 DBLP 에 대해서만 실험을 진행했다.

실험결과, 데이터를 읽는 시간은 약 211.54 초, 계산한 결과를 저장 장치에 쓰는 시간은 약 1577.56 초, 계산 시간은 약 86.88 초였고, 쓰기 시간, 읽기 시간, 계산 시간의 백분율은 각각 약 84.09%, 11.28%, 4.63%로 읽기, 쓰기를 포함한 저장 장치에 접근하는 시간은 계산 시간의 약 20.6 배로 측정됐다.

수행시간의 약 95%가 저장장치 접근 시간이었다. 즉, 행렬 곱셈을 여러 번 수행하게 되는 SimRank 의 경우에는 더욱 많은 저장장치 접근 시간을 요구한다는 것을 알 수 있다.

<표 1> DBLP 로 SimRank 계산 시 각 시간과 비율

	읽기	쓰기	계산	전체
시간 (초)	211.54	1577.56	86.88	1875.99
비율 (%)	11.27	84.09	4.63	100

4. 결론

본 연구에서는 사회연결망에서 유의미한 정보를 추출하기 위한 여러 가지 어플리케이션들의 핵심이 되는 행렬 곱셈 연산을 빅 데이터에 대해 수행하고 그 결과를 분석했다.

실험을 통해 사회연결망 데이터를 분석하는 행렬 곱셈에는 내적 방법보다는 행-행 방법이 시간 측면에서 훨씬 유리하기 때문에 사용하기 적합함을 확인했고 행렬 곱에 걸리는 시간의 대부분은 읽기, 쓰기를 포함한 저장 장치 접근 시간임을 확인했다. 따라서 행렬을 이용한 사회연결망 데이터의 분석을 빠르게 하기 위해서는 저장 장치 입출력에 걸리는 시간을 줄이는 것이 핵심임을 확인했다.

참고문헌

[1] Nathan Bell and Michael Garland, Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report, NVR-2008-2004, NVIDIA Corp, 2008.

[2] Yong-Yong Jo, Sang-Wook. Kim and Duck-Ho Bae, “GPU-based Matrix Multiplication Methods for Networks Analysis,” In *Proc, Int’l Conf, on Research in Adaptive and Convergent System*, ACM RACS, 2014.

[3] Vasily Volkov and James Demmel, “Benchmarking GPUs To Tune Dense Linear Algebra,” In *Proc, Of Int’l Conf, on Supercomputing*, SC, pp. 1-11, 2008.

[4] Kiran Kumar Matam, Siva Rama Krishna Bharadwaj, and Kishore Kothapalli, “Sparse matrix-matrix multiplication On Hybrid CPU+GPU platforms,” In *Proc, of High Performance Computing*, HiPC, 2012.

[5] Glen Jeh and Jennifer Widom, “SimRank: A Measure of Structural-Context Similarity,” In *Proc, ACM Int’l Conf, on Knowledge discovery and data Mining*, ACM SIGKDD, pp. 538-543, 2002.