GPU 를 활용한 분산 컴퓨팅 프레임워크 성능 개선 연구

송주영, 공용준, 심탁길, 신의섭, 성기진
SK C&C, 인프라 사업개발 본부, Cloud Computing 기술담당
e-mail : jy.song@sk.com, andrew.kong@sk.com, terryshim@sk.com
shine62@
sk.com, keejin.seong@sk.com

A Study on Performance Improvement of Distributed Computing Framework using GPU

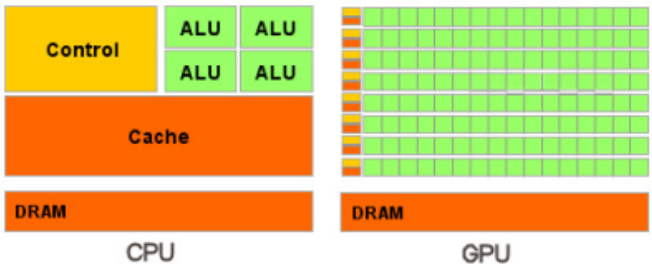
Ju-young Song, Yong-joon Kong, Tak-kil Shim, Eui-seob Shin, Kee-kin Seong
Cloud Computing Technology Team, Dept. of Infra Business Dev, SK C&C

요 약

빅 데이터 분석의 시대가 도래하면서 대용량 데이터의 특성과 계산 집약적 연산의 특성을 동시에 가지는 문제 해결에 대한 요구가 늘어나고 있다. 대용량 데이터 처리의 경우 각종 분산 파일 시스템과 분산/병렬 컴퓨팅 기술들이 이미 많이 사용되고 있으며, 계산 집약적 연산 처리의 경우에도 GPGPU 활용 기술의 발달로 보편화되는 추세에 있다. 하지만 대용량 데이터와 계산 집약적 연산 이 두 가지 특성을 모두 가지는 문제를 처리하기 위해서는 많은 제약 사항들을 해결해야 하는데, 본 논문에서는 이에 대한 대안으로 분산 컴퓨팅 프레임워크인 Hadoop MapReduce 와 Nvidia 의 GPU 병렬 컴퓨팅 아키텍처인 CUDA 를 연동하는 방안을 제시하고, 이를 밀집행렬(dense matrix) 연산에 적용했을 때 얻을 수 있는 성능 개선 효과에 대해 소개하고자 한다.

1. 서론

최근 대규모 연산의 수행을 필요로 하는 다양한 과학 기술 연구 분야에서 GPU(Graphics Processing Units)를 활용하는 방법들이 활발하게 논의되고 있다. (그림 1)에서 보는 바와 같이 GPU 는 CPU 에 비해 칩 면적의 많은 부분을 연산 유닛에 할당하고 있기 때문에 대량의 부동소수점 연산을 비교적 빠르게 처리할 수 있다. 이러한 GPU 와 CPU 의 근본적인 설계 철학의 차이는 GPU 를 이용하여 계산 집약적인 문제나 병렬성이 풍부한 문제를 수행할 경우 보다 많은 성능 향상 효과를 얻을 수 있게 해준다. 실제로 다양한 과학, 기술, 산업 분야에서 GPU 기술을 활용한 솔루션들을 사용하고 있으며 이를 통해 많은 효과를 얻고 있다.[1]



(그림 1) CPU 와 GPU 의 설계 철학의 차이

하지만 입력 데이터의 크기가 커지거나 문제의 복잡도가 높아지는 경우 하나의 GPU 만으로는 문제를 해결할 수 없게 되어 추가적인 GPU 자원을 활용해야 하는데 이것이 어느 시점에 이르면 단순히 장비에 GPU 를 추가하는 방법만으로는 문제를 해결할 수 없게 된다. 왜냐하면 시스템 버스 대역폭 제약, 전력이나 발열 문제, PCIe 슬롯 수 제한 등 하드웨어적인 문제 뿐만 아니라, 다수의 GPU 를 효율적으로 관리하기 위한 소프트웨어적인 방법론에 대한 추가적인 고민들이 필요하기 때문이다. 물론 이런 제약을 피하기 위해 단일 또는 다수의 GPU 가 장착된 다수의 노드들을 클러스터로 구성하고 MPI(Message Passing Interface) 같은 분산 메모리 구조의 프로그래밍 모델을 활용할 수도 있다. 하지만 이 경우에도 역시 환경 구축을 위한 병렬화 프로그래밍 기법 등의 선행 지식이 필요하며 모든 데이터 전송을 직접 관리해야 하기 때문에 프로그래밍하기가 어렵다는 단점이 존재하고, 내고장성(fault-tolerance) 지원의 미흡함 등의 이유로 여전히 한계가 있다.

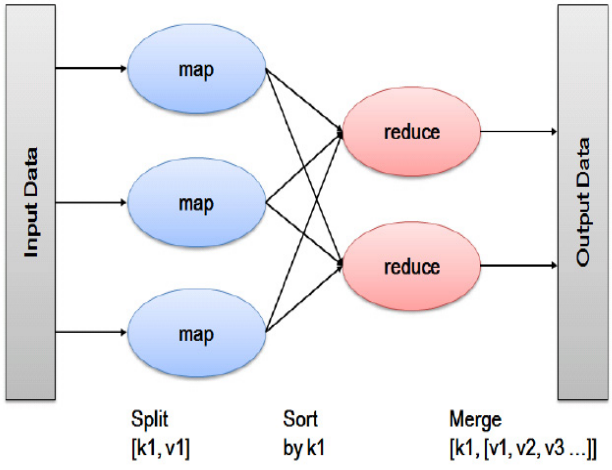
이에 본 논문에서는 오픈 소스 분산 컴퓨팅 프레임워크인 Hadoop MapReduce 와 Nvidia 의 GPU 컴퓨팅 플랫폼인 CUDA 를 연동하는 방안을 제시하며, 이를 통해 대용량 데이터 특성과 계산 집약적 특성을 동시에 가지는 문제를 손쉽게 해결하기 위한 방안을 모색한다. 또한, 하나의 장비에서 처리 불가능한 크기의 거대 행렬-벡터 곱셈 연산에 대한 성능 평가 분석 결과를 토대로 연동 시 고려할 점이나 향후 보완점에

대해서도 논해보도록 한다.

2. CUDA 와 MapReduce 소개

CUDA(compute unified device architecture)는 Nvidia 의 병렬 컴퓨팅 아키텍처로서 GPU 를 이용한 범용적인 프로그램을 개발할 수 있도록 Nvidia 제공하는 프로그램 모델, 프로그램 언어, 컴파일러, 라이브러리, 디버거, 프로파일러 등의 통합 환경을 지칭한다. GPU 는 본래 CPU 의 그래픽스 작업으로 인해 생기는 병목현상을 해결하기 위해 고안된 특수 목적 처리장치(special-purposed processor)이지만 CPU 보다 높은 트랜지스터 집적도와, SIMT(single instruction multiple thread) 아키텍처에 의한 탁월한 병렬처리 능력, 높은 메모리 대역폭 등으로 인해 GPU 를 계산 생물학/화학, 이미지/비디오 프로세싱, 각종 시뮬레이션, 레이트레이싱, 암호학 등 많은 분야에서 범용 처리장치로 사용하기 위한 연구가 활발하게 진행되고 있다.[2]

MapReduce 는 클러스터 환경에 분산 저장된 대용량 데이터를 독립된 작업으로 나누어 효과적으로 처리할 수 있도록 설계된 소프트웨어 프레임워크이다. MapReduce 는 (그림 2)과 같이 map 단계와 reduce 단계로 구성되며, 특정 블록 단위로 나뉘어 각 분산 노드에 저장된 데이터들을 입력으로 사용한다. Map 단계에서는 입력된 데이터를 <key, value>의 값으로 초기화 및 변형 단계로 각각의 입력 레코드들이 병렬로 처리된다. Reduce 단계에서는 Map 단계에서 생성된 데이터들을 사용자의 정의에 따라 요약 혹은 구성을 통해 결과 값을 출력한다. 위의 과정에서 개발자는 Map 함수와 Reduce 함수만을 구현하면 된다. 나머지 분산, 병렬 처리와 같은 과정은 MapReduce 프레임워크에서 자동으로 처리해준다.[3]



(그림 2) MapReduce 모델

3. CUDA-MapReduce 연동 방안

이 절에서는 MapReduce 에서 CUDA 를 사용하기 위해 사용한 라이브러리와 실제 map/reduce 함수의 구현 방법, 사용된 key/value 설계 방식에 대해 소개한다.

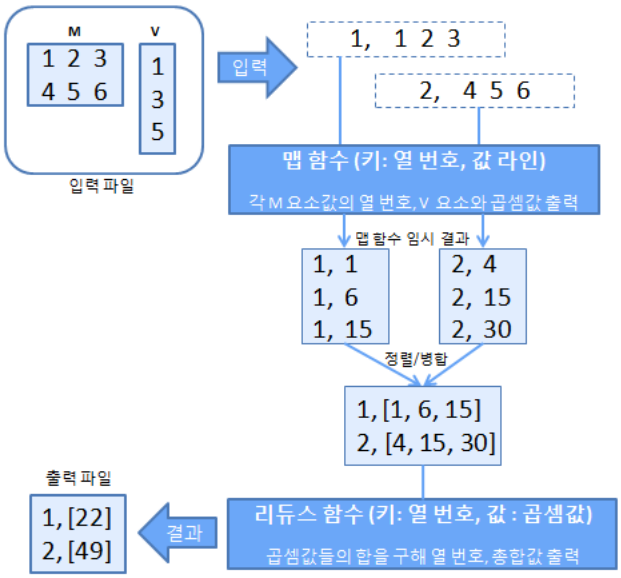
MapReduce 프레임워크는 Java 로 구현되어 있기 때문에 C 기반의 CUDA 를 연동하기 위해서 JCUDA[4]를 사용하였다. JCUDA 는 CUDA 를 위한 Java 바인딩 라이브러리로서 Driver API 와 Runtime API 모두를 지원하며 CUBLAS(CUDA BLAS library), CUFFT(CUDA FFT library) 등의 CUDA 기반 라이브러리들에 대한 지원도 함께 제공하고 있다.

Map/Reduce 함수의 상세 구현은 다음과 같다. 행렬-벡터 곱셈 연산의 결과 벡터 한 요소에 대한 내적 연산을 map 함수에서 수행하며, reduce 함수에서는 결과 벡터의 인덱스를 key 로 하여 연관된 모든 value 들을 단순히 합하는 역할을 수행하도록 구성하였다. 특히, 각 map 함수 내부 곱셈 연산 구현을 CUDA 커널로 구현하여 성능 향상을 얻을 수 있도록 하였는데, 만약 행렬의 한 행이나 입력 벡터가 메인 메모리에 적재가 불가능할 정도로 큰 경우에는 입력 데이터를 여러 개의 stripes 로 분할하여 처리하는 것도 가능하다.[5]

<표 1> MapReduce 프로그램 모델의 기본 개념

map	***: (k1, v1) -> list(k2, v2)***
reduce	***: (k2, list(v2)) -> list(v3)***

MapReduce 프로그래밍 모델의 기본 개념은 <표 1>과 같이 각각의 map 과 reduce 단계에서 key 와 value 의 쌍으로 이루어진 데이터를 처리하는 프로세스로 이루어진다.[6] 여기서는 행렬-벡터 곱셈 연산을 위해 k1, k2 를 입력 행렬 요소의 열번호로, v1 을 입력 행렬의 한 라인으로, v2 를 내적 연산을 구성하는 곱셈 결과로, v3 는 결과 벡터의 한 요소값인 최종 내적값을 갖도록 각각 설계하였다. (그림 3)에서 이러한 key/value 설계 방식의 실제 동작 구성을 2x3 크기의 행렬을 이용하여 설명하였다.



(그림 3) Matrix-Vector Multiplication 을 위한 MapReduce 구현의 동작 예시[7]

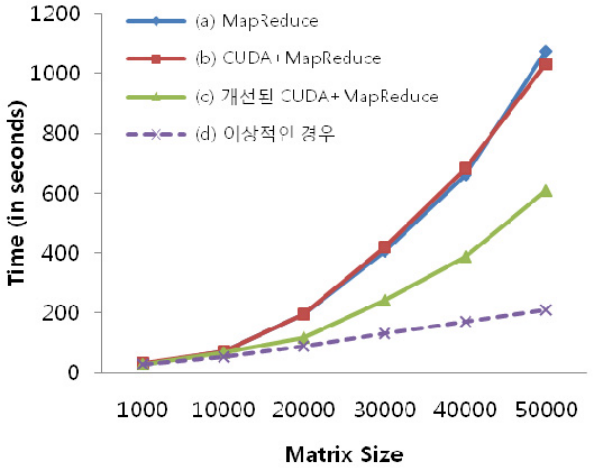
4. 실험 및 결과

4.1 실험 환경

실험에 사용된 시스템 환경은 GPGPU 그래픽카드를 장착한 PC 급 장비 3 대(GeForce GTX 460 을 장착한 Intel Core2 Quad CPU 2.4GHz 장비 2 대, GeForce 9600 GT 를 장착한 Intel Core2 Quad CPU 2.50GHz 장비 1 대)로 구성되어 있으며, 운영체제는 CentOS 6.0 을 사용하였고 CUDA 는 버전 4.0 을 사용하였다. 이 3 대의 장비를 이용하여 1 Namenode, 3 Datanode 로 Hadoop 클러스터를 구성하였고 Hadoop, MapReduce 디폴트 설정을 그대로 사용하였다.

4.2 실험 결과 및 분석

성능 분석을 위한 입력 데이터로는 크기 1000 부터 50000 까지의 정사각행렬을 사용하였는데, 파일의 크기가 5 MB 부터 크게는 14 GB 에 이르기 때문에 다양한 크기의 데이터에 대해 HDFS 와 MapReduce 의 성능을 테스트 할 수 있었다. 실험결과 (그림 4)의 추세선 (a)와 (b)에서 보이는 것처럼 CUDA 를 적용했음에도 불구하고 성능향상 효과가 없는 것으로 나타났다.

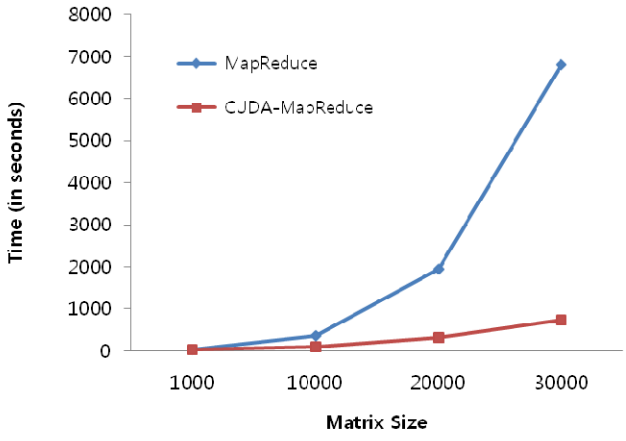


(그림 4) Matrix-Vector Multiplication 에 대한 성능 평가 결과

이는 MapReduce 프레임워크가 실제 행렬-벡터 곱셈 연산을 실행하기 위해 사용하는 전체 수행시간에 비해서 CUDA 를 이용해 병렬화 가능한 구간 즉, 행렬-벡터 곱셈 수행 시 주요 연산이라고 할 수 있는 벡터의 점곱(dot product) 연산이 차지하는 비중이 아주 작기 때문에 나타난 결과이다. 이럴 경우 병렬화 불가능한 부분에 의해서 전체 성능 향상 효과가 제한된다는 암달의 법칙에 의해 성능 향상 효과를 기대할 수 없는 것이다. 여기서는 이를 해결하기 위해 병렬화 가능한 부분이 커지도록 Hadoop 의 split 구성 방식을 수정하여 다수의 라인이 map 함수 안에서 처리하도록 수정하였다. 그 결과 (그림 4)의 추세선 (c)와 같은 결과를 얻을 수 있었다. CUDA 적용의 효과로 행렬의 크기가 커짐에 따라 성능 차이도 점점 늘어나는 것을 확인할 수 있다. 하지만 이 경우에도 역시 이상적인 경우의 추세선 (d)와 같이 선형 증가에 가까운 모습의 결과를 얻지는 못했다.

이것은 CUDA 아키텍처의 기본적인 특성에 기인한다. CUDA 의 데이터 흐름은 메인 메모리에서 그래픽 카드 메모리로, 그래픽 카드 메모리에서 다시 GPU 레지스터로 이동하여 연산을 한 후에 다시 반대의 순서로 메인 메모리로 되돌아가는 식으로 동작한다. 이러한 호스트-디바이스 간 데이터 전송 부분은 GPU 를 사용하면서 추가로 발생되는 부하인데다가 실제 계산 작업에 쓰이는 프로세싱 유닛의 연산 속도에 비하면 매우 느린 작업에 속하기 때문에 극단적인 경우에는 GPU 를 이용했을 때 더 느려지는 경우도 발생할 수 있다. 또한, 그래픽카드 내부에서 발생하는 메모리 접근도 성능을 위해 고려해야 할 중요한 부분이다. CUDA 에서는 한 개의 전역 메모리 접근 당 수행되는 부동소수점 연산의 개수의 비를 CGMA(Compute to Global Memory Access)라는 개념으로 나타내는데[8] 이 수치가 높을수록 GPU 에 의한 성능 향상 효과가 높다. 이렇듯 GPU 도입 효과를 높이기 위해서는 데이터 재사용성과 CGMA 비율을 높여 데이터 전송 비용을 낮추기 위한 노력이 필요한데, 행렬의 성분 하나가 한 연산에만 쓰이는 행렬-벡터 곱셈 연산의 특성상 이러한 수치가 낮게 나올 수 밖에 없기 때문에 이상적인 성능 향상 효과는 기대하기 어렵다.

이를 해결하기 위한 방법으로 CUDA 의 Page-Locked Host Memory 를 사용하는 Zero Copy 나 스트림 기법을 활용할 수 있지만[9] 여기서는 좀 더 명확한 사실 확인을 위해 데이터 재사용성과 계산 복잡도 큰 연산을 임의로 만들어 동일한 환경에서 수행해 보았다. 그 결과 (그림 5)에서 보는 바와 같이 앞에서 기대했던 것과 동일한 결과를 얻을 수 있었다.



(그림 5) 이상적인 데이터 전송, 계산 직접도를 갖는 연산에 대한 CUDA 의 뛰어난 성능 개선 효과

5. 결론 및 향후 계획

본 논문에서는 GPU 사용 가능한 노드들로 이루어진 MapReduce 클러스터에서 계산 집약적 애플리케이션을 수행했을 때의 성능 개선 효과에 대하여 알아보

았다. 실험 결과를 통해 1. 노드 간 데이터 이동이 없는 문제에 한해서 2. 병렬성이 풍부한 embarrassing parallelism, data parallelism 특성을 가지며 3. 데이터 전송 효율을 높일 수 있는 조건을 만족하는 경우에 비약적인 성능 향상 효과를 얻을 수 있다는 것을 알 수 있었다. 이를 통해 유추해보면 행렬-벡터 곱셈 연산 외에도 몬테카를로 시뮬레이션이나 이미지 프로세싱 영역에서 좋은 결과를 얻을 수 있을 것으로 예상할 수 있다.

HPC(High Performance Computing) 분야에서 많이 쓰이는 Linear Algebra 문제의 경우 CUDA-MapReduce 연동의 가장 큰 제약 조건인 노드 간 데이터를 공유하지 않는 shared-nothing 구조가 큰 제약으로 작용하게 된다. 왜냐하면 행렬-행렬 간의 연산을 분산 플랫폼에서 처리하기 위해 부분행렬로 분할할 경우 필연적으로 각 부분행렬 노드 간에 데이터 공유 문제가 발생하기 때문이다. Google 에서 발표한 대용량 그래프 분석 프레임워크인 Pregel[10]은 기본적으로 BSP(Bulk Synchronous Parallel) 기반의 커뮤니케이션 모델을 사용하여 프레임워크 자체에서 노드 간 데이터 전송 기능을 제공하는데, 향후 이러한 Pregel 기반의 분산 프레임워크 구현에 CUDA 를 연동하면 고가의 슈퍼컴퓨터 장비를 구축하지 않아도 $O(N^3)$ 복잡도를 갖는 거대 행렬-행렬 곱셈 연산이나 선형계(Linear System)의 해를 찾는 등의 문제를 비교적 간단히 해결할 수 있을 것으로 기대된다.

참고문헌

[1] http://www.nvidia.com/object/vertical_solutions.html
[2] 성주호, 이윤우, 한아, 최원익, 권동섭. "CUDA 를 활용한 병렬 B+-트리 벌크로드 기법"
[3] 이혁주, 김명진, 이한구 윤효근. "대용량 소셜 데이터의 의미 분석을 위한 MapReduce 기반의 분석 모듈 설계 및 구현"
[4] http://jcuda.org/
[5] Anand Rajaraman, Jeffrey D. Ullman, "Mining of Massive Datasets", Chapter 2. Large-Scale File Systems and Map-Reduce
[6] 김형준, 조준호, 안성화, 김병준. "클라우드 컴퓨팅 구현 기술", p. 259
[7] Tom White, "Hadoop 완벽 가이드"
[8] David B. Kirk, Wen-mei W. Hwu. "대규모 병렬 프로세서 프로그래밍", p. 90
[9] NVIDIA "CUDA C Programming Guide, Version 4.0"
[10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn,N. Leiser, and G. Czajkowski. Pregel: A System for Large-Scale Graph Processing