GPU기반의 계산속도와 CPU기반의 계산속도 비교 및 특정 프로그램에 따른 적합한 모델 찾기에 대한 연구

신현수
경북대학교 컴퓨터학부
e-mail : hyunsoo7708@gmail.com

A Study on comparison of calculation between CPU-intensive and GPU-intensive and finding proper model for specific program

Hyun-Soo Shin
Dept. of Computer Science, Kyungpook National University

요 약

최근 기술이 발달함으로 인해 더 짧은시간에 더 많은 계산량이 필요해진 시대가 왔다. 본 연구에서는 CPU와 GPU의 구조를 파악하고 계산속도를 비교한다. 직렬 방식의 알고리즘에서의 병렬 방식의 알고리즘 및 현재 GPU 병렬처리 적용 사례 및 추후 적합한 모델 찾기에 대해 연구한다.

1. 서론

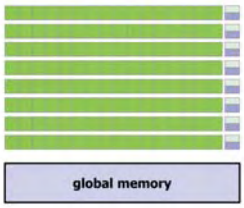
4차산업혁명을 시작한 이후로 요구되는 컴퓨터의 계산량은 꾸준히 증가해오고 있다. 딥러닝, 블록체인(PoW)[1],데이터가공 및 처리 분야에서 컴퓨터의 계산량이 매우 중요해졌다. 기존의 CPU에서 처리하던 작업은 요구되는 계산량을 더 이상 따라가지 못하는 상황이 발생하여 많은 공학자들은 GPU를 기반으로한 계산에 초점을 맞춰 연구를 진행해오고 있다. 본 연구는 CPU기반의 계산속도와 GPU기반의 계산속도 비교 및 목적 프로그램에 무엇의 모델이 더 적합한지 장단점을 비교한다. CPU기반의 계산속도와 GPU기반의 계산속도를 비교하기위해 입력값으로 주어진 랜덤문자열을 찾아내는 시뮬레이터를 작성함으로써 비교해본다. 본 시뮬레이터의 GPU기반의 시뮬레이터는 C++로 작성했으며 CUDA(GPU 가속화 라이브러리)[2]를 활용한다. CPU기반의 시뮬레이터는 Go언어로 작성되었으며 Go-routine을 활용한다. 작성된 소스코드는 깃허브에 올려두었다.

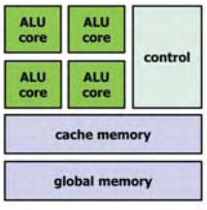
https://github.com/hyunsooda/Parallel-Brute-Force-Algorithm

2. 동시성과 병렬성

CPU의 각 코어는 인스럭션을 파이프라인으로 처리한다. UNIX 시스템은 쓰레드를 여러개 생성하여 동시성 프로그래밍을 할 수 있다. 이것은 온전히 운영체제 정책에 의존하며 운영체제는 모든 프로그램의 실행시간 할당을 공정하게 배분하여 골고루 CPU의 점유를 확보해야한다. 서비스 혹은 백그라운드로 실행되고있는 시스템들의 스케쥴링 우선순위가 높고 유저 프로그램은 상대적으로 우선순위가 낮다. 최근 출시되고 있는 CPU가 가지고있는 코어갯수는 8~16개 사이이다 코어의 클럭성능에 차이로 인해 각 코어가 쓰레드를 생성하고 스케쥴링 할 수 있는 쓰레드의 갯수는 실험적이며 통상적으로 많지않다. 코어가 가지고있는 성능을 오버하여 쓰레드를 생성할 경우 Thrasing[3]의 이슈로 인해 성능은 더 낮아지게 된다. 결국 CPU기반의 멀티쓰레드 프로그래밍은 운영체제에 의해 CPU 실행시간을 번갈아가면서 점유하므로 병렬처리가 아니며 이를 동시성이라고 한다. 따라서 많은 쓰레드가 필요한 최신 기술들의 요구되는 계산량을 고려했을때 코어의 갯수가 중요하다는 사실을 알 수 있다. 트랜지스터는 CPU의 클럭 수 를 높이는데 중요한 전자소자이다. 트랜지스터의 갯수가 많을수록 클럭 수 는 높아진다. 하지만 트랜지스터는 heating-problem[4]으로 인해 무어의법칙에 따라 2년마다 증가한다.

GPU는 한개의 칩 안에 GPU모델에 따라 1024개에서 4096개의 코어를 탑재하고있다. CPU에 비해 훨씬 더 많은 코어를 확보하고있으며 많은 GPU 라이브러리는 최적화된 쓰레드 스케쥴링을 하며 범용 계산에 초점을 맞추어 개발되고있다.





(그림 1) GPU 구조 (그림 2) CPU 구조

3. 계산속도 비교를 위한 시뮬레이터 구현

CPU에서 멀티쓰레드 프로그래밍 방식에서의 race condition, load balancer(work-balance)같은 이슈는 GPU에서의 프로그래밍 방식에서도 존재한다.

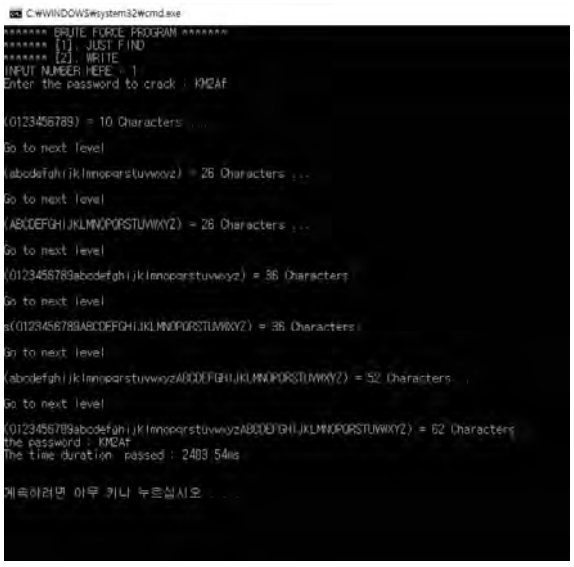
본 시뮬레이터는 무작위 문자열을 입력받아 brute-force[5] 알고리즘으로 문자열을 순차적으로 생성하여 비교한다. 3개의 버전으로 작성했으며 1개는 Go언어로 Go-routine를 이용해 작성했고 1개는 C++언어로 작성했고 1개는 CUDA를 활용해 작성했다. 이 시뮬레이터는 주어진 입력값에 대해 무작위로 문자를 하나씩생성하여 비교하기때문에 태스크병렬(Task parallelism)[6]이 아닌 데이터 병렬(Data parallelism)[7]에 초점을 맞추었다. POSIX 표준 함수인 pthread를 활용 할 수도 있었지만 동시성을 최대한 활용하기 위해 Go-routine을 활용하였다. 시뮬레이터가 찾아야할 무작위 문자열의 경우의 수 는 총 7가지이다.

<표 1> 무작위로 생성할 문자열 경우의 수 정의

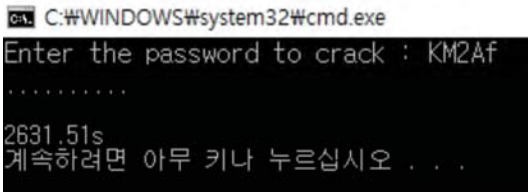
case 1	숫자 **10개**(0~9)로만 구성
case 2	영어소문자 **26개**로만 구성
case 3	영어대문자 **26개**로만 구성
case 4	숫자 10개 + 소문자26개 = **36개**
case 5	숫자 10개 + 대문자26개 = **36개**
case 6	소문자 26개 + 대문자 26개 = **52개**
case 7	숫자 10개 + 소문자 26개 + 대문자 26개 = **62개**

예를들어 “12345”를 찾아야 한다면 경우의 수 는 10^5 = 100000이고, “123ab”를 찾는다면 경우의 수 는 36^5 = 60466176이고, “a1b2G”를 찾는다면 경우의 수 는 62^5 = 916132832이다.

(그림 3) GPU기반 시뮬레이터 실행화면



(그림 4) CPU기반 시뮬레이터 실행화면



(그림 5) 멀티쓰레드기반 시뮬레이터 실행화면

<표 2> 1000번 실행 후 통계 비교

	GPU(CUDA)	CPU(C++)	Go(go-routine)
찾을값 : KM2Af	3초	44분 21초	5분 35초

<표 3> 실행 환경

CPU	Intel i7-7700 (3.60Ghz)
메모리	8GB
시스템	64bit Window
그래픽카드	NVIDIA GeForce GTX 1050

4. 적용 가능한 분야 및 관련연구

최근 이목을 받고있는 비트코인의 작업증명방식[8]은 많은 컴퓨팅 파워를 요구한다. 작업증명방식은 마이너가 무작위 난수 해쉬값을 많은 컴퓨팅 파워를 이용해 찾아냈는지 체크한다[9]. 많은 경쟁자 마이너들이 존재하며 컴퓨팅파워가 높을 수 록 채굴에 유리하다. ASIC 채굴기가 많이 이용되고 있지만 향후 Anti asic 알고리즘으로 GPU파워의 수요가 늘어 날 것이다. 현재 이더리움은 Anti asic 알고리즘을 개발중에 있다[10].

최근 몇년간 ImageNet[11] 대회를 비롯한 이미지인식에 대한 딥러닝 연구가 활발히 진행되어왔다. 이미지인식에서 가장 많이 활용되는 CNN[12] 알고리즘은 학습을 위해 많은 연산을 필요로한다. 그에따라 강력한 컴퓨팅 파워를 제공하는 GPGPU의 수요가 동시에 급증했다.

cuDNN은 NVIDIA에서 제공하는 딥러닝 가속화 프레임워크이다[13]. 영상처리 분야에서 활발하게 활용되고 있으며 다음은 이미지처리에 대한 성능 비교 테이블이다.

<그림 6> 딥러닝을 활용한 이미지처리 성능비교

Batch 사이즈	훈련시간 CPU	훈련시간 GPU	GPU 가속 능력
64 images	64 초	7.5 초	8.5배
128 images	124 초	14.5 초	8.5배
256 images	257 초	28.5 초	9.0배

5. 직렬 알고리즘에서 병렬알고리즘으로 전환

본 시뮬레이터를 CPU기반의 직렬코드로 작성한다면 for문 중첩으로 구현할 수 있다. 하지만 이는 한개의 작업이 끝나야 다른 작업을 진행할 수 있다. GPU기반의 병렬코드로 작성했을때 CPU기반의 직렬코드처럼 for문 중첩으로 작성한다면 각 쓰레드들은 중복해서 똑같은 일을 하게 된다. 따라서 직렬코드에서 병렬코드로 전환하려면 알고리즘을 수정해야한다. n을 문자열의 길이라고 했을 때 각 쓰레드는 n^2 만큼의 비교연산을 한다. 각 쓰레드의 목표는 입력값과 일치하는 문자열을 찾는것이 목표이다. 만약 각 쓰레드가 n^3 혹은 n^4 혹은 n^5 혹은 n^k 만큼의 연산을 한다면 쓰레드를 덜 생성해도 되므로 쓰레드 스케쥴링에는 유리하지만 문자열을 빨리찾는 경우에 다른 쓰레드들은 더이상 하지 않아도 될 연산으로 낭비하게 된다. 물론 공유에 의한 통신(communication by sharing)으로 문자열을 빨리 찾았을때 공유변수에 특정 플래그를 대입해서 다른 쓰레드들을 종료시키게 할 수있다. 하지만 공유변수에 대한 비교횟수가 추가되기에 문자열을 찾는 시간이 조금 더 오래 걸리게된다. 이러한 최적화 문제는 만들고자하는 프로그램에 따라 그때그때 다르며 충분히 많은 실험을 해봐야한다. 마지막으로 데이터가 병렬성을 띄는지 분석해봐야 한다. 데이터가 병렬성을 내포하지 않는다면 데이터를 다시 모델링 할 필요가 있다.

6. 목적 프로그램 분석 및 적합한 모델 찾기

항상 GPU에서의 계산속도가 CPU에서의 계산속도를 압도하는 것은 아니다. Merge sort, Binary search 등 여러 정렬 및 탐색 알고리즘은 CPU기반의 직렬 알고리즘이 GPU기반의 병렬알고리즘보다 우세하다.

<표 4> 임의의 정수10000개 생성 후 정렬 및 탐색비교

	C++ STL	CUDA
Sequential Search	0.005132 msec	16.537249 msec
Binary Search	0.002138 msec	0.090528 msec
Bubble Sort	0.28011 msec	0.545723 msec

데이터를 더 많이 생성할수록 CUDA에서의 성능이 더 떨어지게 되는데, global memory의 참조 횟수가 많아지기 때문이다. 공유메모리(shared memory)나 상수메모리(constant memory)를 사용할 경우 접근속도가 100배 빨라진다. CUDA를 설치하면 demo_suite 디렉터리에 deviceQuery.exe라는 프로그램이 제공된다. 이 프로그램은 로컬컴퓨터가 탑재하고있는 그래픽카드에 따라 세부스펙을 보여준다.

<그림 6> deviceQuery 실행결과

```
Total amount of constant memory:               65536 bytes
Total amount of shared memory per block:       49152 bytes
```

GTX 1050 ti 기준으로 상수메모리는 65kb, 공유메모리는 49kb인것으로 확인된다. 이는 매우 작은 사이즈이며 정렬이나 탐색알고리즘에 사용될 데이터가 클 경우 매우 한정된 자원이다.

<표 5> Bitonic 정렬 비교

Bitonic Sort	CPU	CUDA
Data size = **10000**	1.750508 msec	0.121120 msec
Data size = **32M**	5791.780601 msec	12265.739258 msec

Bitonic sort는 병렬기반의 정렬 알고리즘에서 효과가 좋은것으로 알려져 있다.[14] 위의 표에서 데이터 갯수가 10000개일 때는 공유메모리를 활용한 결과이며 데이터 갯수가 32M일때는 글로벌메모리를 사용한 결과이다. 데이터 갯수가 작을때 공유메모리를 활용할 경우 병렬알고리즘이 더 나은 경우가 있지만 이는 아주 소수이며 대부분은 CPU기반의 코드가 더 빠르다. 물론 데이터갯수가 많아질경우에는 반드시 CPU코드가 더 빠르다.

따라서 목적 프로그램이 필요로 하는 operation들을 정확히 파악한 후 적합한 모델을 선택해야 한다.

7. 결론

머신러닝과 IoT등 여러 기술산업이 발달하면서 GPGPU의 수요가 증가해오고 있다. 본 시뮬레이터에서의 결과에서는 GPU가 멀티코어를 활용하지 않는 CPU보다 887배 더 빠른 실행결과를 보였다. 멀티코어를 적극적으로 활용하는 Go-routine에서의 결과는 GPU가 CPU보다 약 111배 더 빠른 실행결과를 보였다. 인스트럭션은 결국 프로세서에 의해 실행 되는 것이며 많은 인스트럭션을 많은 프로세서에서 병렬로 처리하여 빠른 실행속도를 보이는 병렬 프로그래밍의 수요가 많아 질 것으로 보인다. 하지만 이것은 그래픽카드만 있다고 가능한 것은 아니다. 기존의 직렬 알고리즘을 병렬 알고리즘으로 전환하는 작업이 필요하며 이는 많은 시간과 노력이 요구된다.

참고문헌

[1] Proof of Work, https://en.bitcoin.it/wiki/Proof_of_work
[2] CUDA, https://en.wikipedia.org/wiki/CUDA
[3] Thrainsg, https://en.wikipedia.org/wiki/Thrashing_(computer_science)
[4] Heat Generation and Transport in Nanometer-Scale Transistors by Eric Pop, Sanjiv Sinha, and Kenneth E. Goodson, https://nanoheat.stanford.edu/sites/default/files/publications/A90.pdf
[5] Brute force, https://en.wikipedia.org/wiki/Brute_force
[6] Task parallelism, https://en.wikipedia.org/wiki/Task_parallelism

[7] Data parallelism, https://en.wikipedia.org/wiki/Data_parallelism
[8] Bitcoin: A Peer-to-Peer Electronic Cash System by Satoshi Nakamoto : https://bitcoin.org/bitcoin.pdf
[9] Source code of BitCoin, https://github.com/bitcoin/bitcoin/blob/master/src/pow.cpp
[10] Report of Ethereum Anti asic, https://ethereumworldnews.com/ethereum-anti-asic-protocol-delayed-crypto-developers/
[11] ImageNet competition, http://www.image-net.org/challenges/LSVRC/
[12] Convolutional Neural Network, https://en.wikipedia.org/wiki/Convolutional_neural_network
[13] cuDNN, https://developer.nvidia.com/cudnn
[14] Dmitri I. Arkhipov, Di Wu, Keqin Li, and Amelia C. Regan, “Sorting with GPUs, A Survey”

그림 1. https://developer.nvidia.com/educators
그림 2. https://developer.nvidia.com/educators
그림 6. http://webcache.googleusercontent.com/search?q=cache:7Hf8iDTbuqUJ:itsys.hansung.ac.kr/cgi-bin/kplus/board/include/DOWNLOAD.php3%3Ftable%3Dlec_ai%26l%3D3-024903%26inc%3Dlocal%26f%3Dfilelink+&cd=7&hl=ko&ct=clnk&gl=kr