연구논문/작품 제안서

2018 년도 제 1학기

논문/작품	○논문(∨) ○작품( ) ※ 해당란에 체크
제목	GPU 멀티태스킹에서의 문제점 파악
GitHub URL	https://github.com/SanghoonARCS/graduation-thesis
팀원명단	홍 재 완 (학번: 2013312784) 조 상 훈 (학번: 2013314616)

2018 년 3 월 21일

지도교수 : 한 환 수 서명

1. 과제의 필요성

Abstract

이미지 렌더링과 연구 목적의 컴퓨팅에 주로 머물던 GPU의 활용 범위가 범용 어플리케이션으로 그 영역을 확장하고 있다. 늘어나는 범용 GPU 어플리케이션의 니즈를 맞추기 위해 GPU에서의 multi-tasking은 선결적으로 해결해야 하는 과제가 되었다. 어플리케이션들은 복잡해지고 세분화되어 가고 있다. 이에 독자적으로 수행되나 또 동시적으로도 수행되어야 할 어플리케이션들에 대한 지원이 GPU에서는 아직 미흡하다. GPU는 태생적으로 자원 공유를 위한 설계의 철학이 부족하다.

GPU란 그래픽 처리장치(Graphics Processing Unit)의 약자로, 이미지와 영상을 처리하는 역할을 담당하는 그래픽카드의 핵심 반도체이다. Intel의 개인용 PC CPU는 코어수가 8개(i7-6900K), 서버용 CPU는 28개(Xeon Platinum 8180) 정도인 반면, NVIDIA의 GPU(Titan X)는 3,584개의 코어를 보유하고 있다. 그래픽 처리 분야에만 사용되던 GPU가 다수 코어에 의한 병렬 연산의 장점을 기반으로 일반적인 데이터 처리에도 활용되는 GPGPU(범용 GPU)로 발전하여 많은 연구들이 진행 되어 오고 있고 GPGPU의 활용의 폭이 더욱 넓어지고 있다. 고정된 형태의 GPU를 사용하던 방식에서 프로그램 가능한 보다 유연한 형태의 GPU로 패러다임이 넘어왔다. 과거 3D그래픽스 분야를 위해서만 사용되었던 GPU의 높은 행렬과 벡터 연산 성능을 일반 컴퓨팅에 적용하고자 하는 최근의 추세이다.

인공지능, 빅데이터 등 '4차 산업혁명'으로 대변되는 새로운 기술의 발전이 대용량의 데이터를 효율적으로 처리하는 GPU의 성장을 견인하고 있다. 방대한 양의 데이터, 스스로 학습하는 알고리즘, 온/오프라인의 구분 없는 연결성 등을 특징으로 하는 '4차 산업혁명'의 등장과 함께 GPU의 사용량이 급격히 증가하고 있다. 그 사례로 2017년 하반기부터 시작된 국내의 가상화폐 시장의 확대와 함께 가상화폐를 빠르게 발굴할 수 있는 GPU의 수요가 급격히 높아져 그 가격이 2018년 상반기 아직도 내려오고 있지 않고 있다. 외장

형 GPU 시장 최대 기업인 NVIDIA의 경우, 대용량 데이터 분석 및 클라우드 컴퓨팅(cloud computing) 관련 GPU 매출이 2년 만에 360% 이상 증가하였으며, 자율주행차 시장 관련 매출도 82% 증가하였다. 가상화폐 붐이 일기 전인 2016년에는 전 세계 GPU 매출이 전년대비 40% 이상 증가하는 등 기존 게임및 엔터테인먼트 영역뿐만 아니라, 빅데이터/인공지능/자율주행차 등 새롭게 성장하는 시장에서 GPU 사용이 보편화되는 것을 확인 할 수 있다[1].

GPGPU는 상당이 매력적인 기술이나 아직 그 범용성에 있어서는 진입장벽이 높다. 특히 프로그래머들의 병렬 프로그래밍 능력이 뒷받침 되어야 할 뿐만 아니라 GPGPU 플랫폼에 대한 이해도가 높아야 고성능의 소프트웨어를 만들 수 있다. 특히나 여러 어플리케이션이 한 GPU에서 돌아가는 경우를 대비한 GPU의 지원은 매우 부족하다. GPU의 자원들이 여러 어플리케이션들이 효율적으로 공유 할 수 있는 하드웨어적 접근은 최근에 들어서야 조금씩 만들어 지고 있으며, 소프트웨어적인 접근은 여러 가지 시도가 있지만 적용된 것들은 극히 일부이다. 최근 컴퓨팅 시스템의 주요 특징 중 하나인 멀티어플리케이션 지원이 GPU에서는 아직 원시적인 수준이다. 고등의 멀티어플리케이션 능력이 앞으로 GPU의 위치를 CPU라는 히어로의 sidekick에서 머물지, 미래의 기술들이 GPU로 옮겨 갈수 있는지 달려있다[3].

GPU들은 한 번에 하나의 어플리케이션을 실행하도록 디자인이 되어있다. 동시에 여러 어플리케이션을 돌릴 경우, GPU의 방대한 multi-threading 패러다임 때문에 어플리케이션들끼리 공용자원을 두고 해로울정도로 경쟁을 벌이며 이는 시스템 전체 throughput에 악영향을 미친다.

공유하기 어려운 여러 자원들 중 특히 GPU의 cache는 멀티어플리케이션을 고려하지 않고 설계되었기 때문에 극히 소량만이 존재한다. GPU 하드웨어를 살펴보면 공간의 대부분을 컴퓨팅 자원에 내어주고 cache는 절대적으로 적은 양만이 남아있다. 가장 최근 아키텍처인 NVIDIA TitanX를 보면 전체 L1 cache의 크기가 1,152KB밖에 되지 않는다. 49,152개의 concurrent

thread를 지원하는 모델임을 감안하면 각 스레드별로 25B의 cache공간이 주어지는 셈이다. 이 적은 공간의 cache를 여러 어플리케이션들이 효율적으로 공유하는 방법이 없다. NVIDIA에서 제공하는 GPU 언어인 CUDA에서는 cache 제어 접근을 지원 할 정도로 cache는 GPU에서 성능을 좌지우지하는 요소이다. 하지만 양날의 검처럼 잘못 사용할 경우 성능의 향상보다는 끔직한 성능 저하를 마주하게 된다. 자원 공유를 해야 하는 multi-tasking에서의 캐시 활용을 위한 연구가 산발적으로 이루어지고 있으나 저마다의 원인규명을 가지고 원자적인 연구가 진행되고 있다. 이에 Multi-tasking이 요구되는 시대의 요구에 어떠한 애플리케이션과 알고리즘이 GPU 에서 다중으로 실행되어 질지를 파악 하여 그에 대한 현재 cache 활용에서의 문제점을 파악하고자 한다.

2. 선행연구 및 기술현황

GPU 멀티어플리케이션의 최근 기술 동향은 다음과 같다.

GPU에서의 Multiple application execution는 temporal이나 spatial 방법으로 수행되고 있다. Temporal 방법은 time multiplexing을 사용해서 여러 유저들에게 자원을 배분하는 가장 흔한 GPU virtualization 테크닉이다. 그러나 이런 공유 방법은 자원들을 효율적으로 경제적으로 활용하지 못하고 퍼포먼스도 매우 떨어진다.

[6]에서는 더 큰 warp를 사용하여 warp내의 branch divergence의 성능 향상을 이끌어냈다. 하지만 이 테크닉은 어플리케이션이 직접 thread-level parallelism을 구현하지 않으면 아직 gpu기기 자원의 full utilization을 이끌어내지 못하고 있다. [7]에서는 elastic kernel을 사용하여 커널의 동시성을 높이는 기술을 제안하고 있다. 하지만 이 기법은 모든 커널에 적용될 수 있는 것이 아니며, 특히 하드웨어 id와 소프트웨어 id가 다르면 이 기법은 사용이 불가하다. [8]에서는 warp-aware memory scheduling을 하여 warp끼리의 contention을 줄이기를 제안하고 있다.

이 중에서 졸업 논문과 관련하여 조금 더 지엽적인 연구로는 다음과 같은 선행 연구가 있다.

최근 GPU 아키텍쳐들은 MPS를 사용하여 Huper-Q feature를 소프트웨어 레이어 에서 돌려서 여러 어플리케이션들이 돌아 갈 수 있는 환경을 제공해 주고 있다. 그러나 이 방식은 cache thrashing에 취약하여 캐시의 올바른 활용법이 되지 못한다.

적은 양의 캐시를 여러 어플리케이션들이 공유하기 위해서 현재 사용되고 있는 대표적인 기술로는 cache bypassing과 cache partitioning이 있다. 현재까지 나와 있는 GPU 아키텍쳐들은 cache bypassing을 이용하거나

partitioning을 이용하는 방법을 채택하고 있다. 두 기술이 상호적으로 이용되어지지 않고 독립적으로 이용되고 있어, 최근 선행 된 연구로는 이 두 가지 기법을 융합하는 기법으로 성능 향상을 꾀하고 있다. Cache partitioning을 하면 task별 bypassing decision을 내리는데 첫째 문제가 생긴다. Cache bypassing은 데이터 리퀘스트와 쓰레드가 캐시를 접근 할지 bypass할지를 결정하는데, 이 결정이 각 task의 캐시 capacity에 영향을 준다.
Cache bypassing과 partitioning을 동시에 적용하는데 생기는 어려움을 적용 단계를 나누어 해결하고 있다. [5]

3. 작품/논문 전체 진행계획 및 구성

3. 1 기본적인 컴퓨터구조 복습

GPU도 기본적으로는 프로세서의 일종이기 때문에 기존의 것들과 일정부분 구조를 공유한다. 그러므로 전에 전공수업에서 학습한 기본적인 프로세서의 구조와 메모리 계층에 대해서 다시 한 번 정리하고 복습할 예정이다.

3. 2 Parallel computing 학습

GPU는 Parallel computing에 근간을 두기 때문에 그것에 대한 기본적인 지식을 학습할 필요가 있다. 그러므로 기존에 존재하던 Parallel computing에서 참고한 것이 많기에 X86의 Multimedia extensions 과 Vector processor등을 공부하면서 Parallel computing의 기본적인 개념을 학습할 예정이다.

3. 3 GPU관련 기반 지식 학습

GPU는 일반 CPU와 다른 부분이 많기 때문에 GPU 구조와 용어를 공부 할 예정이다. 먼저 GPU 구조로는 프로세서가 어떠한 구조로 되어 있는지 기존의 CPU와 비교를 하면서 공부를 하고 캐시의 구조와 용도가 다를 수 있기 때문에 이 또한 기존 CPU와 비교를 하면서 공부를 할 예정이다. 그리고 용어가 기존에 사용하던 것과 많이 다르기 때문에 기존에 습득한 용어와 뜻이 통하는 점을 비교하면서 공부를 할 것 이다.

3. 4 Ndivia GPU microarchitecture 구조 이해

이번 논문은 Ndivia 기반으로 진행할 예정이기 때문에 먼저 그것에 대한 구조의 이해가 필요하다. 학습 예정인 GPU microarchtecture로는 Tesla, Fermi, Kepler, Maxwell, Pascal, Volta가 있다. 위와 같이 많은 microarchtecture를 학습하는 이유는 프로세서와 캐시의 구조가 한 세대를 넘길 때 마다 많이 변경되고 발전하는 경향을 보이기 때문이다. 그러므로 발전하는 추세를 보면 구조에 대한 이해 뿐 만이 아니라 개선방향을 찾는데

많은 도움이 될 것으로 예상한다.

3. 5 CUDA 프로그래밍 학습

Ndivia GPU를 사용하기 위해서는 필수적으로 CUDA에 대한 이해가 필요하다. 개발자는 패스스케일 오픈64 C 컴파일러로 컴파일 된 'CUDA를 위한 C' (C언어를 Ndivia가 확장한 것) 를 사용하여 GPU 상에서 실행시킬 알고리듬을 작성할 수 있다. CUDA를 통해 개발자들은 CUDA GPU 안 병렬 계산 요소 고유의 명령어 집합과 메모리에 접근할 수 있다. CUDA를 사용하여 최신 Ndivia GPU를 효과적으로 개방적으로 사용할 수 있다. 그러나 CPU와는 달리 GPU는 병렬 다수 코어 구조를 가지고 있고, 각 코어는 수천 threads를 동시에 실행시킬 수 있다.

3. 6 GPGPU-Sim 구조 분석

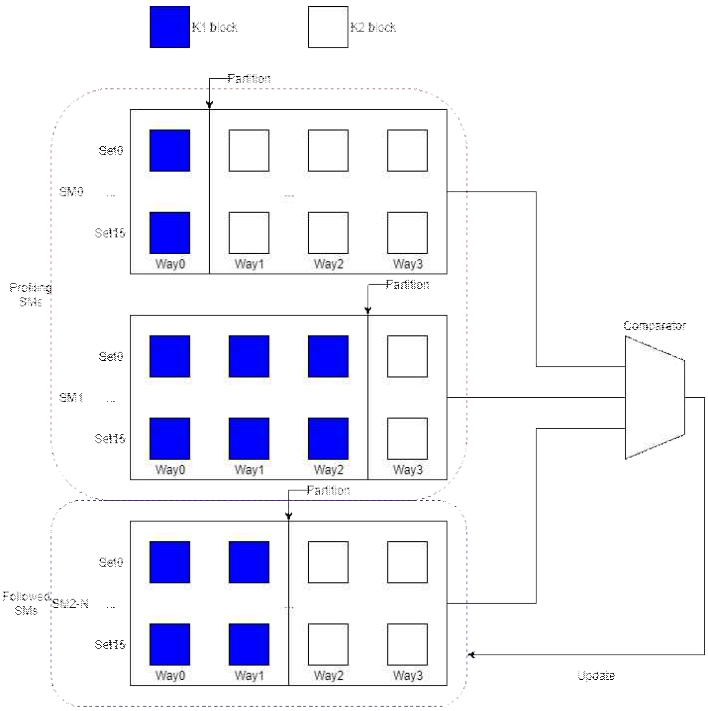
구조를 개선하여 성능향상을 측정할 환경으로 GPGPU-Sim을 사용할 예정이기 때문에 해당 구조와 소스코드를 이해하는 것은 필수적이다. 실제로 구조를 개선할 때 GPGPU-Sim의 소스코드를 수정하여 구현할 예정이기 때문에 각 소스코드가 무슨 동작을 하고 어떻게 다른 소스코드와 연계되는지 정확하게 알 필요가 있다. 다만 GPGPU-Sim은 복잡한 프로그램이므로 구조와 소스코드를 분석하고 이해하는데 상당히 많은 시간이 필요할 것이다.

3. 7 기반 논문의 구현 및 추가적인 개선방향 탐색

참고 문헌에 있는 기반 논문을 실제로 구현하고 논문에서 사용한 벤치마크를 사용하여 성능을 측정 할 것이다. 구현 도중 별도의 개선방향 아이디어가 생각나면 그것 또한 구현을 하고 기반 논문의 것과 같은 벤치마크를 통해서 성능 비교를 할 예정이다. 그리고 실제로 성능이 개선되었으면 그 이유도 분석할 것이다.

4. 기대효과 및 개선방향 (1페이지 내외)

암호화폐나 머신러닝 등 최근에 GPU computing이 점차 널리 쓰이고 있다. 그러므로 자연스럽게 점차 multitasking하는 상황이 증가하는 추세이다. Multitasking은 여러 applications이 한 GPU에서 동시에 resource을 공유하면서 작동하는데 이것은 cache와 같은 자원에 대한 경쟁을 유발한다. 하지만 cache를 이용하는 것은 비교적 어렵기 때문에 잘못 사용된다면 성능이 오히려 저하될 가능성이 높다. 이러한 상황에서 cache partitioning과 cache bypassing 기법을 적용하면 전반적인 system throughput이 상당부분 증가할 것이다.



이번 연구에서의 목표는 함께 작동하는 tasks들 간의 cache contention을 줄이고 각 task들의 data locality를 보존해서 전체적인 system throughput을 증가시키는 것이다. 그러기 위해 앞에서 언급한 cache partitioning과 cache bypassing을 계층적으로 적용할 것이다. 먼저 cache partitioning으로 task 마다 cache 안에서 구역을 정하는데 static 방식과 dynamic 방식이 있다. 기반이 되는 논문에서는 static 방식이 좀 더 좋은 결과를 보여주는데 위의 scheme에서 좀 더 개선된 dynamic 방식을 고안한다면 다른 결과를 보여줄 수 있을 것이다.

5. 기타

홍재완

기본적인 컴퓨터구조, Parallel computing, GPU관련 기반 지식, Ndivia GPU microarchitecture 등 연구를 진행하는데 필요한 이론적 배경과 지식들을 중점적으로 습득하고 팀원과 공유한다.
습득한 내용의 공유는 팀원 간 세미나를 통해 이루어질 예정이며 진행 도중에 필요한 지식들이 있는지 지속적으로 탐색한다.

조상훈

팀원과 같이 습득한 배경과 지식을 바탕으로 CUDA 프로그래밍 학습. GPGPU-Sim 구조 분석, 기반 논문의 구현 및 추가적인 개선방향 탐색 등 연구를 실질적으로 구현하고 testing하며 개선방향을 제시하는데 집중한다.
구현하는 내용을 팀원이 공유해야 하기 때문에 지속적으로 구현 내용을 피드백 받으면서 진행한다.

세부 항목	소요 경비(₩)
컴퓨터 본체 (CPU, RAM, SSD, Mainboard 등)	700,000 * 2 = 1,400,000
컴퓨터 주변기기 (키보드, 마우스 등)	50,000 * 2 = 100,000
모니터 (24inch 이상)	300,000 * 2 = 600,000
그래픽 카드 (Pascal microarchitecture 이상)	1,000,000 * 2 = 2,000,000
합계 (팀원 모두 포함)	4,100,000

6. 참고문헌

[1] KB 금융지주 경영 연구소 "4차 산업혁명과 GPU(Graphics Processing Unit)의 성장"
[2] Shinpei Kato† ǂ, Karthik Lakshmanan† , and Ragunathan (Raj) Rajkumar†, Yutaka Ishikawa "TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments"
[3]Jacob T. Adriaens, Katherine Compton, Nam Sung Kim "The Case for GPGPU Spatial Multitasking"
[4]Srinvasa Reddy Punyala, Theodoros Marinakis, Arashi Komaee "Throughput Optimization and Resource Allocation on GPUs under Multi-Application Execution"
[5]Yun Lian, Xiuhong Li, Xialong Xie "Exploring Cache Bypassing and Partitioning for Multi-Tasking on GPUs"
[6]A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of gpu memory system for multi-application execution," in Proceedings of the 2015 International Symposium on Memory Systems. ACM, 2015.
[7]S. Pai, M. J. Thazhuthaveetil, and R. Govindarajan, "Improving gpgpu concurrency with elastic kernels," in ACM SIGPLAN Notices, vol. 48, no. 4. ACM, 2013, pp. 407–418.
[8]N. Chatterjee, M. O'Connor, G. H. Loh, N. Jayasena, and R. Balasubramonian, "Managing dram latency divergence in irregular gpgpu applications," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2014, pp. 128–139.