# CUDA GPGPU와 CPU standard C 기반 연산 성능 비교를 위한 Benchmark 환경의 설계 및 구조

김동훈 O, 이기범, 오상윤*
아주대학교 정보컴퓨터공학과, *아주대학교 소프트웨어특성화학과
{paian, rhlehwls, syoh} @ajou.ac.kr

# Benchmark environment for computational performance CUDA GPGPU and CPU based standard C

DongHoon Kim O, Kibeum Lee, Sangyoon Oh*
Graduate School of Information and Computer Engineering, Ajou University
*Graduate School of Software, Ajou University

**요 약**

하드웨어 기술이 발전함에 따라 CPU를 능가하는 GPU의 연산능력을 바탕으로 컴퓨터 그래픽스 계산에 사용되던 GPU를 CPU를 대신해 범용 계산에 사용하는 GPGPU에 대한 연구가 활발히 진행되고 있다. 이에 따라 CPU와 GPU의 성능을 분석하고 비교해 적절한 환경에 배치하는 것도 매우 중요한 요소로 떠오르고 있다. 본 논문에서는 CPU와 GPU의 성능을 비교하는 요소들에 대한 분석과, 이 분석을 기반으로 새로운 벤치마크 환경을 결정하고 NVIDIA에서 제공하는 소프트웨어 플랫폼인 CUDA를 사용해 두 개의 배열에서 공통된 요소를 찾아내는 프로그램을 구현해 CPU와 GPU의 성능을 비교 분석하였다. 그 결과 주어진 조건에서 GPU를 사용한 프로그램이 CPU의 수행시간보다 약 45배 빠른 것을 확인하였다.

## 1. 서 론

최근 하드웨어 시장이 발전함에 따라 CPU와 GPU 시장과 관련기술도 함께 발전하고 있다. 현재 헥사코어(hexa-core), 옥타코어(octa-core)에 이어 16개의 코어를 가진 헥사데시멀코어(hexadecimal core)가 등장했다. GPU의 경우 CPU의 코어 수 증가트랜드 보다 앞서서 병렬 처리가 강화되고 코어 수를 비약적으로 증가된 매니 코어가 등장했다. [1]

단위 GPU의 연산성능이 CPU의 성능을 능가함에 따라 컴퓨터 그래픽스를 위한 계산을 다루는 용도로 사용되던 GPU를 범용 계산에 활용하는 GPGPU (General Purpose computing on Graphics Processing Units)에 대한 연구가 활발히 진행되고 있다.

하지만 GPU의 구조 특성상 한번에 하나의 instruction만 실행이 가능하고 메모리 접근이 느리기 때문에 무분별한 GPU 사용은 적절하지 않다. 이에 따라 GPU와 CPU성능을 분석하고 비교하여 적절한 환경에 배치하는 것도 매우 중요해졌다. 이 논문에서는 NVIDIA CUDA를 이용한 GPGPU의 성능과 CPU의 성능을 분석하고 비교하는 환경을 구축하는 방법에 대한 연구결과를 설명한다. [2]

본 논문의 구성은 다음과 같다. 2장에서는 CPU와 GPU의 구조와 GPGPU의 개념을 알아보고 GPU를 범용계산에 활용할 수 있는 NVIDIA CUDA와 CPU와 GPU의 성능을 비교 분석하는 방법에 대해 알아본다. 3장에서는 실제 실험한 결과를 토대로 CPU와 GPGPU의 성능을 비교 분석한다. 마지막 4장 결론에서는 본 논문의 전반적인 요약과 향후 계획에 대해 말한다.

## 2. 관련 연구

### 2.1 GPGPU

GPGPU란 GPU를 범용 계산에 활용하는 것으로 CPU가 하는 연산을 GPU가 대신하면서 연산속도를 향상시키는 기술이다. 그림 1. 은 CPU와 GPU 구조를 각각 보여주는데 CPU는 복잡한 제어 로직과 캐시 메모리를 사용하는 반면 GPU는 다수의 ALU(Arithmetic Logic Unit)를 사용해 단순하고 많은 계산을 병렬적으로 빠르게 계산할 수 있다. CPU의 속도가 300GFlops정도인데 비해 대부분의 GPU는 1TFlops를 넘어가고 있다.

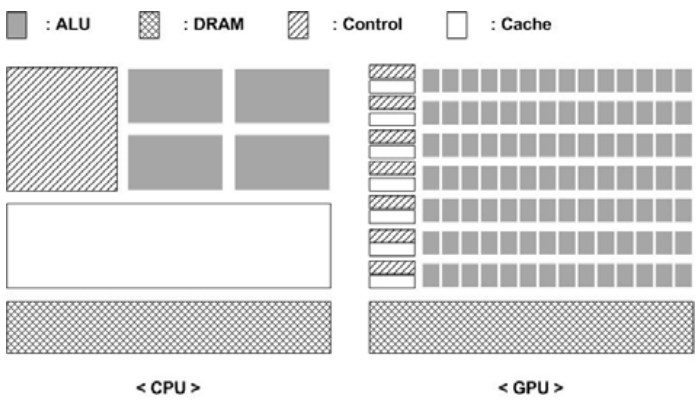


그림 1. CPU와 GPU 구조

GPU는 연산수행속도는 CPU의 연산수행속도 보다 빠르지만 CPU에 비해 Control이 부족하기 때문에 GPU에서 단독으로 작업을 처리할 수 없다. 그렇기 때문에 CPU에서 GPU를 제어해야 한다.

2.2 CUDA

CPU보다 더 좋은 연산능력이 있는 GPU를 범용적인 프로그램에 사용하기 위해 nVIDIA사는 2007년 C언어를 기본으로 별도의 라이브러리를 추가하여 고급언어의 추가 지원이 가능한 소프트웨어 플랫폼인 CUDA를 발표하였다.

그림 2. 에서 보여지듯이, 프로그램이 실행되면 CPU(Host)에서 프로그램이 수행이 되다가 연산이 필요하다고 판단되면 GPU(Device)를 호출한다. GPU(Device)가 호출되면 하나의 Grid가 생성되고 이 Grid는 여러 개의 Block을 가지고 있으며 각 Block은 여러 개의Thread를 사용해 CPU(Host)가 요청한 연산을 수행한다. 따라서 사용자는 프로그램의 특성과 GPU와 GPU의 성능을 고려하여 Grid와 Block, Thread를 적절히 조절해 프로그램을 최적화 시켜야 한다.

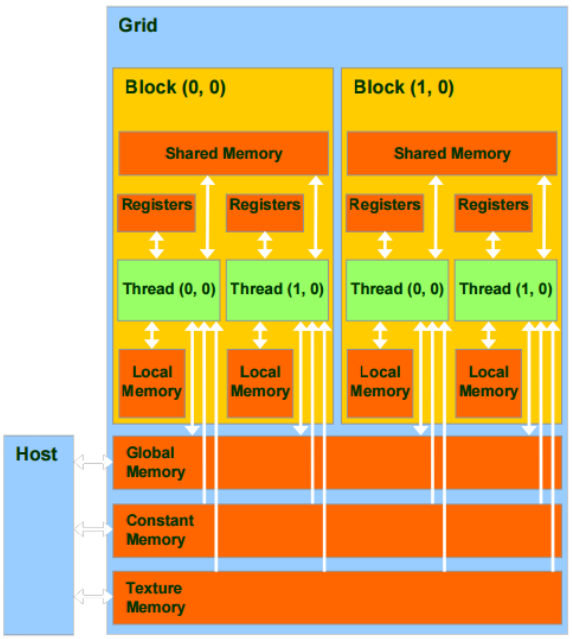


**그림 2. CUDA 관점에서의 메모리 계층구조**

2.3 비교 환경

일반적으로 하드웨어 시스템 성능을 비교, 평가하는 방법으로 벤치마크(Benchmark)를 사용한다. 벤치마크란 실제의 적용업무에 맞춘 표준문제를 설정해 비교대상인 컴퓨터로 실행시켜 처리시간을 기본으로 점수를 측정하고 측정된 이 점수는 누구라도 인정할 수 있도록 하드웨어의 성능을 표준화시키는 작업이다. 그러므로 벤치마크의 점수는 소비자가 하드웨어를 구매 하거나 프로젝트를 진행할 때 어떠한 하드웨어를 선택하고 적용할지 도와주는 가이드 역할을 한다. 하드웨어 벤치마크를 하기 위해서는 동일한 플랫폼에서 동일한 input을 사용해 각각의 output을 확인하고 처리시간과 정확성을 비교해 점수를 측정한다.

벤치마크의 대표적인 예로 TPC(Transaction Processing Performance Council) 벤치마크를 꼽을 수 있다. TPC란 시스템의 성능 평기 척도의 마련과 각 회사의 벤치마크 값을 검증하기 위하여 만들어진 비영리 단체로 TPC-C, TPC-H, TPC-R, TPC-W등이 있다.[3]

하지만 본 논문에서는 CPU와 GPU의 서로 다른 하드웨어 플랫폼의 연산 성능을 비교하는 것이 목적이기 때문에 동일한 하드웨어 플랫폼을 비교하는 일반적인 벤치마크는 사용할 수 없다. 그렇기 때문에 CPU와 GPU의 연산 성능을 비교 평가하기 위해서는 새로운 벤치마크를 설계해야 한다.

두 개의 서로 다른 approach를 비교하기 위해서는 프로그램이 실행되는데 필요한 요소들을 동일하게 해야 한다. 하드웨어의 성능이나 특성에 따라 동일하게 하지 못하는 요소들은 각자 주어진 환경에서 최대의 효율을 낼 수 있어야 한다. 이에 따라 본 논문에서는 사용자가 제어할 수 있는 input과 알고리즘을 동일하게 하였고 하드웨어의 특성상 같게 할 수 없는 Thread의 수는 CPU와 GPU에서 사용할 수 있는 최대 Thread를 사용하여 벤치마크 환경을 설계하였다.

## 3. GPGPU와 CPU의 성능 비교 분석을 위한 환경 설계 및 구조

본 절에서는 GPU와 CPU의 성능 비교 분석에 대해서 설명하고 이를 위한 전송과 환경 설계 및 구축에 대한 고려요소들과 본 연구그룹의 결론 배경에 대해 서술하고, 구축된 환경에 대해서 설명한다.

3.1 GPGPU와 CPU의 성능 비교 분석

본 논문에서는 데이터 비교 프로그램을 통해 CPU와 GPGPU의 성능을 비교 분석한다. 데이터 비교를 위한 프로그램은 unsigned int 타입의 데이터 값들이 정렬되어 들어가 있는 2개의 배열들 중에서 중복된 값들을 추출해 새로운 배열에 저장하고 중복된 값의 개수가 몇 개 인지 찾아내는 프로그램이다. CPU 환경을 테스트하기 위해서 standard c sequential logic에서 BST(Binary Search Tree)알고리즘을 사용하였고 GPGPU 환경을 테스트 하기 위해 CUDA multi thread logic에서 BST알고리즘을 사용하였다. 사용한 CPU와 GPU의 성능은 표 1. 과 같다.

| | CPU | GPU |
|---|---|---|
| 프로세서 | Intel i7-5930k | NVIDIA GTX 970 |
| 코어의 개수 | 6 | 1664 |

| 최대 성능 | 289 GFLOP | 3494 GFLOP |
|---|---|---|
| 전력량(W) | 140W | 145W |
| 성능 / 전력(W) | 2.06 | 24 |
| 가격 / 성능 | ₩2197.2/GFLOP | ₩110.9/GFLOP |

**표 1. CPU와 GPU SPEC**

3.2 GPGPU와 CPU의 성능 비교 요소

CPU와 GPGPU의 연산 성능을 비교하기 위해서는 프로세서를 제외한 모든 환경이 동등해야 한다. 따라서 프로그램에 사용되는 알고리즘과 input data의 크기를 동일하게 만든 후에 프로그램을 실행하고 결과를 추출하였다. 프로그램에 사용한 환경은 표 2. 와 같다.

| 배열의 크기 (N, M) | 133,705,728 |
|---|---|
| 배열의 타입 | Integer |
| 사용한 알고리즘 | BST |
| Time complexity | O( N * log(M) ) |

**표 2. 프로그램 환경**

실험은 CPU 내에서 single Thread와 12개의 multi Thread로 BST알고리즘을 사용해 두 개의 배열에서 동일한 데이터를 찾는 프로그램을 실행시켜 수행시간을 측정하였고, CUDA를 사용해 CPU에서 GPU를 호출한 후 input크기에 같은 수의 Thread를 사용해 위와 동일한 작업을 수행하는 프로그램의 수행시간을 측정하였다. 실험의 결과는 표 3. 과 같다.

| | CPU | | GPU |
|---|---|---|---|
| Thread | 1 | 12 | 133,705,728 |
| 수행시간 (ms) | 44.743 | 5.077 | 1.021 |

**표 3. 실험결과**

GPU에서 Thread는 그 수의 제한이 없이 사용자가 원하는 개수를 만들 수 있다. 따라서 실험에서는 input data의 크기와 동일한 약 1억 3천개의 thread를 만들었다. GPU내에서 Thread의 개수는 사용자가 원하는 만큼 만들 수 있지만 GPU에 block의 개수가 고정되어 있기 때문에 실제로 GPU 내에서 동시에 작업할 수 있는 Thread의 수는 대략 2000개 에서 3000개 사이이다. 실험결과 GPU를 사용한 프로그램의 수행속도가 CPU single Thread를 사용한 프로그램의 수행속도 보다 약 44배 정도 빠르고 12개의 multi Thread를 사용한 프로그램의 수행속도보다 약 5배 빠른 것을 알 수 있다. 이 실험은 사용한 비교방법이 복잡한 알고리즘 수행이 아닌 비교적 단순한 unsigned int 값을 BST비교 알고리즘을 사용했으므로 각 Architecture의 Thread를 이용한 연산 수행 방식의 차이라고 할 수 있다.

## 4. 결론

최근 CPU 연산을 GPGPU가 대신하는 대한 연구가 활발히 진행되어 왔다. GPU의 구조를 연구하고 GPGPU와 CPU의 성능을 비교해 효율적인 프로그램을 만드는 연구와 CPU와 GPU간의 커널을 연결하는 방법 등 많은 연구가 있었고 NVIDIA CUDA와 같은 High Level Language를 사용할 수 있게 해주는 라이브러리들이 등장하면서 좀더 간편하게 프로그램을 만들 수 있게 되었다. 이와 같이 애플리케이션과 이를 적용하는 플랫폼구조 및 수행방식은 매우 중요한 연구 주제이다. 이와 함께 각 방법들의 성능과 장단점을 비교 분석하는 것도 중요한 연구주제이다.

본 논문에서는 비교분석 환경 구축에 있어 입력 환경 구축, 연산 능력을 극대화, 알고리즘 검토가 비교분석 환경 설계 및 구조를 중요한 요소로 보고 이에 대해 연구를 진행하였다. 위에 실험결과에 근거하여 적절한 방법의 적용에 따라 큰 성능향상을 얻을 수 있음을 환경구축과 비교분석을 통해 보였다. GPGPU와 CPU의 성능을 분석하고 어떠한 방법으로 분석하는 것이 효율적인지 알아보는 방법에 본 논문이 도움이 될 것으로 기대된다.

하지만 GPU가 사용하고 있는 SIMT(Single Instruction Multiple Threads) 모델이 한번에 하나의 instruction만 실행 가능하기 때문에 실행 시간이 더 많이 걸리고 메모리 접근이 느리기 때문에 무조건적인 GPU사용은 적절하지 않을 수 있다. 따라서 향후 본 논문에 기반하여, GPU와 CPU를 혼합한(Hybrid) 사용 방안과 이를 비교 분석하는 환경에 대해 연구를 계속 진행하고자 한다.

## 참고문헌

[1] 안진호, 매니 코어 시스템 신뢰성 향상을 위한 부하 분산 기법. 한국정보기술학회논문지 제11권 제2호 143-149쪽. 2013년 2월

[2] NVIDIA Corporation. http://www.nvidia.com.

[3] TPC http://www.tpc.org/

[4] 이호영, 박종현, 김준성. CUDA를 이용한 FDTD 알고리즘 병렬 처리. 전자공학회 논문지 제 47권 CI편 제 4호 82쪽. 2010.

[5] 이세연, 허의남. GPU 공유 메모리 크기에 따른 최적화 기법. 추계학술발표회 논문집. 2013.

[6] Sunpyo Hong, Hyesoon Kim. Memory-level and Thread-level Parallelism Aware GPU Architecture Performance Analytical Model. 2009.