디바이스 소셜리티에서의 GPGPU 자원 공유를 위한 오프로딩 프레임워크
Offloading Framework for Sharing GPGPU Resources in Device Sociality

마정현, 박세진, 박찬익
Jeonghyeon Ma, Sejin Park, Chanik Park

포항공과대학교 컴퓨터공학과 (경북 포항시 남구 효자동 산 31번지)
{doitnow0415, baksejin, cipark}@postech.ac.kr

요약 (국문요약)

디바이스 소셜리티 내에는 서버, 데스크탑, 모바일 단말기와 같은 다양한 성능을 가진 디바이스들이 존재할 수 있다. 이러한 환경에서 모바일 단말기의 경우 상대적으로 다른 디바이스들에 비해 성능이 낮고 전력 소모에 민감하다. 따라서 모바일 단말기의 제한점을 해결하기 위해선 다른 디바이스의 고성능 자원의 공유가 필요하다. 공유 대상 자원의 종류 중 GPGPU의 경우 최근에 모바일 단말기에서 지원이 시작되고 있어 성능이 제한적이고 일부 단말기의 경우 지원하지 않는다. 따라서 본 논문에서는 디바이스 소셜리티 내에서 모바일 GPGPU가 장착되지 않은 모바일에서도 GPGPU 수행을 가능하게 하며, GPGPU 기반 연산의 성능을 향상시킬 수 있는 GPGPU 자원 공유 방식인 GPGPU 오프로딩 프레임워크를 제안하였다. 이 프레임워크는 OpenCL에 기반한 GPGPU 응용을 지원하며, OpenCL 소프트웨어 스택 중 프로그래밍 언어 사이의 이식성이 높고 구현의 복잡도가 낮은 레이어를 대상으로 구현되었다. 실험결과로 행렬 곱 512 x 512 워크로드의 경우 오프로딩을 수행한 결과가 로컬 GPGPU를 사용했을 때의 수행 시간보다 약 2.66배 향상되었다.

Abstract(영문요약)

In the device sociality, devices which have various performance may exist such as server, desktop or mobile. In this environment, mobile device has poor performance and is sensitive to power consumption. Therefore, it is required to share other device's high performance resources in order to resolve the limitation of mobile devices. Among the resources, the performance of mobile GPGPU is lower and some mobile devices still do not support GPGPU. In this paper, therefore, we proposed mobile GPGPU offloading framework that enables mobile GPGPU application to run on mobile which has not GPGPU and improves the performance of GPGPU-based application. This framework supports OpenCL-based GPGPU application and is built on the layer which has high portability and low complexity, among OpenCL software stacks. In case of 512 x 512 matrix multiplication, the result have shown that remote execution performs about 13.5 times better than local execution.

키워드: 모바일, GPGPU, 오프로딩, OpenCL
Keyword: Mobile, GPGPU, Offloading, OpenCL

1. 서론

디바이스들 간에 소셜리티가 생성되어있는 환경 아래 다양한 성능을 가진 디바이스들이 존재할 수 있다. 즉 서버, 데스크탑 혹은 모바일 단말기와 같은 디바이스들이 한 소셜리티 내에 존재할 수 있다. 이러한 상황에서 모바일 단말기의 경우 상대적으로 다른 디바이스들에 비해 성능이 낮고 전력 소모에 민감하다. 따라서, 모바일 단말기의 제한점을 해결하기 위해선 다른 디바이스의 고성능 자원의 공유가 필요하다.

공유 가능한 자원의 종류 중 현재 GPGPU 자원이 모바일 단말기에서 지원이 시작되고 있으나 그 성능이 데스크탑이나 서버에 비해 제한적 [1]이고 일부 모바일 단말기 경우에는 GPGPU를 지원하고 있지 않기 때문에 이 논문에서는 GPGPU 자원을 대상으로 자원의 공유 방식을 제안한다. 표 1은 최신 모바일 GPU와 데스크탑 GPU의 성능을 비교한 표로써, 단위 시간 당 부동소수점의 연산수가 약 41배 이상 차이남을 알 수 있다. 또한 표 2의 경우 모바일 GPU의 GPGPU 지원 여부를 나타낸 표로써, 현재 많은 모바일 플랫폼에서 GPGPU를 지원하지 않고 있음을 알 수 있다. 따라서 이러한 문제점을 해결하기 위해서 본 논문에서는 디바이스 소셜리티 내에서 원격지

서버나 데스크탑의 GPGPU를 모바일에서 공유 가능하도록 모바일 GPGPU 오프로딩 프레임워크를 제안한다. 이 프레임워크는 우리의 기존 연구 [2]에 발전된 형태로 x86 기반의 머신뿐만 아니라 ARM기반의 모바일 환경에서도 GPGPU 오프로딩이 가능하도록 발전시켰다.

(표 1) 최신 모바일 GPU와 데스크탑 GPU 성능비교

	모바일 GPU (Tegra 4)	데스크탑 GPU (GTX 680)
코어개수	72	1536
FLOPS (초당 부동 소수점 연산 수)	74.8×10^9	3090×10^9

(표 2) 최신 모바일 GPU의 OpenCL 지원현황

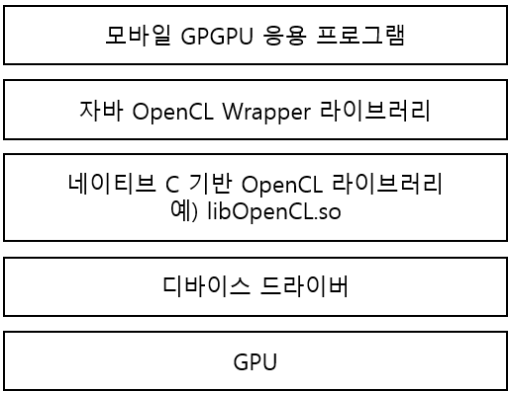
모바일 GPU 모델명	관련 제품	OpenCL 지원여부
Mali-400MP	갤럭시 3, 갤럭시 노트 2	지원안함
Mali-T604	넥서스 10, 넥서스 5	OpenCL 1.1지원
PowerVR SGX535	아이패드, 아이폰4	지원안함
PowerVR SGX543	아이폰4S, 아이폰5	OpenCL 1.0지원

본 논문의 구성은 다음과 같다. 2장에서 모바일 GPGPU 오프로딩과 관련된 배경지식을 설명하고, 3장에서는 모바일 환경의 GPGPU 오프로딩 프레임워크를 설명한다. 4장에서는 모바일 GPGPU 오프로딩 프레임워크의 실제 구현에 대해 설명하고 5장에서는 제안한 프레임워크의 성능 평가를 진행한다. 6장에서 관련연구를 설명하고 마지막으로 7장에서 결론을 맺는다.

2. 배경지식

2.1 GPGPU

GPGPU는 전통적으로 그래픽스를 위한 계산만 다루는 GPU를 CPU가 수행하던 연산량이 많은 워크로드도 수행 가능하도록 확장한 기술이다. GPU 하드웨어 특성상 수백에서 수천 개의 코어를 장착하고 있고, CPU에 비해 비교적 단순한 하드웨어 아키텍쳐를 가지고 있기 때문에, 계산이 단순하면서 높은 병렬성을 요구하는 워크로드를 수행할 때 최적의 성능을 보일 수 있다. 주로 영상처리나 수학 연산 등 다양한 분야에 실제 적용되어 응용 프로그램의 성능이 크게 향상되었다. 현재는 GPGPU 병렬 컴퓨팅의 성능을 슈퍼컴퓨터 분야에도 적용하고 있다. 실제로 2013년도 전 세계 탑 500 슈퍼컴퓨터 중 2위에 랭크된 슈퍼컴퓨터인 Titan는 Nvidia의 K20



(그림 1) 자바 OpenCL 소프트웨어 스택

GPU를 사용하고 있고, 이외의 6위, 11위에 랭크된 슈퍼컴퓨터들 역시 GPU를 이용하여 연산을 수행 하고 있다 [3].

2.2 Java OpenCL (JOCL) [4]

OpenCL [5]은 일반 응용 프로그램에서 GPGPU를 이용할 수 있도록 지원하는 프레임워크 중 하나로써, Apple이 제안하고 그래픽 디바이스 제조사들이 참여하는 개방형 범용 병렬 컴퓨팅 프레임워크이다. OpenCL을 지원하기 위한 라이브러리나 드라이버 모두 응용프로그램에게 C언어 기반의 인터페이스를 제공하고 있다. 이러한 C언어 기반의 인터페이스는 안드로이드와 같은 자바로 응용프로그램을 구성하는 모바일 환경에서 바로 사용할 수 없다. 따라서 이러한 모바일 환경에서도 OpenCL을 지원하기 위해 기존 C언어 기반 인터페이스를 제공하는 OpenCL 라이브러리에 Native Development Kit (NDK)를 통해 자바 OpenCL Wrapper 라이브러리를 구성하여 네이티브 코드를 자바 환경에서도 사용가능하도록 지원하고 있다. NDK는 C나 C++로 구현된 네이티브 코드를 자바 환경에서 사용가능하도록 지원해주는 개발 킷이다.

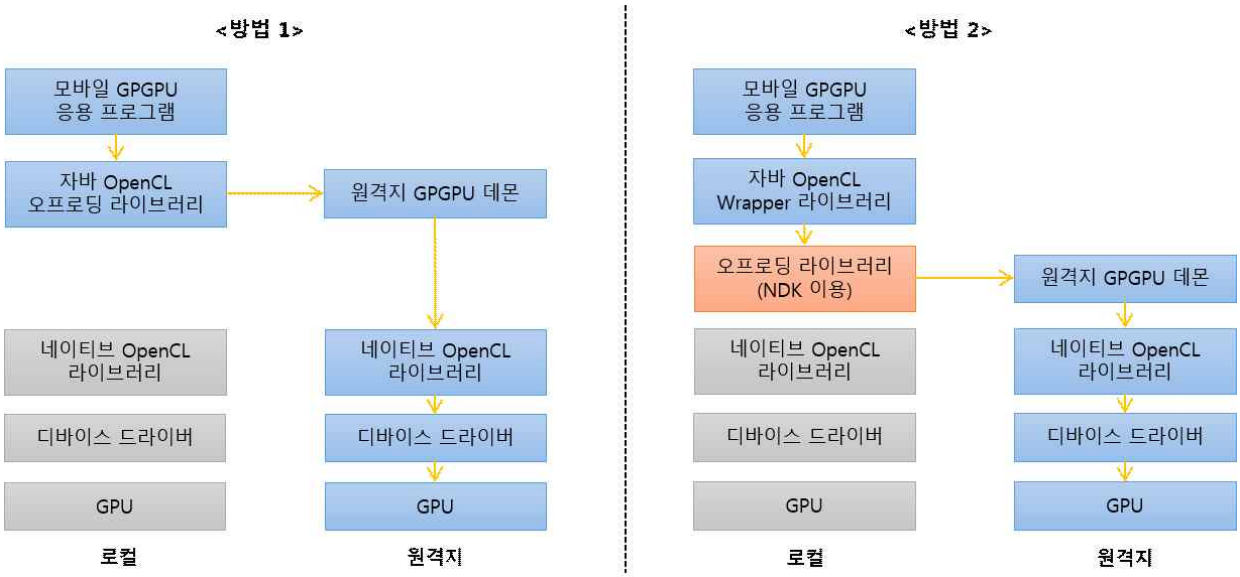
그림 1은 모바일 GPGPU 응용 프로그램이 GPU를 이용할 때 필요로 하는 소프트웨어 스택을 나타낸다. 모바일 GPGPU 응용 프로그램이 네이티브 OpenCL 라이브러리를 통하여 GPGPU를 이용할 수 있도록 네이티브 OpenCL 라이브러리 인터페이스인 자바 OpenCL Wrapper 라이브러리를 응용 프로그램에게 제공한다.

3. 설계

모바일 환경에서 OpenCL 오프로딩을 수행하기 위해서는 그림 1의 소프트웨어 스택을 기반으로 어떤 레이어에서 오프로딩을 수행할 지 결정해야 한다.

3.1 오프로딩 레이어

모바일 GPGPU 오프로딩을 위한 첫 번째 방법은



(그림 2) 로컬에서 원격지로 모바일 GPGPU 응용프로그램을 오프로딩하기 위한 두 가지 방법

모바일 GPGPU 응용 프로그램 내부에서 소스코드를 직접 구현하여 원격서버와 연결하는 통신 모듈을 구성하여 원격지의 GPGPU 자원을 사용하는 것이다. 하지만 이러한 오프로딩 방식은 기존 로컬의 GPGPU를 이용하기 위해 JOCL을 사용하던 응용 프로그램의 원격 수행을 위해선 수정이 불가피하다. 이는 특정 응용 프로그램을 작성할 때, 원격 수행을 위한 응용과 로컬 수행을 위한 응용이 따로 작성되어야함을 의미하기 때문에 응용 프로그램의 이식성이 없는 형태이다.

두 번째 방법으로 디바이스 드라이버 수준에서 Virtual File System (VFS) 인터페이스를 수정하여 오프로딩을 수행하는 방법이 있다. 이 방법은 VFS 인터페이스 내부에 오프로딩 수행 모듈을 추가해야 하기 때문에 소스코드의 수정이 필수적이다. 하지만 현재 GPU 제조사에서 제공하는 디바이스 드라이버는 바이너리 형태로 제공되기 때문에 일반적인 방법으로 구현이 어렵다.

세 번째 방법으로 네이티브 OpenCL 라이브러리를 수정하여 오프로딩을 수행하는 방법이 있다. 이 방법은 네이티브 OpenCL 라이브러리 내부에서 시스템 콜이 수행되기 전에 GPGPU 수행을 위한 관련 정보들을 원격지로 전송하여 원격지에서 전송된 정보를 가지고 시스템 콜을 호출하여 원격지 디바이스 드라이버를 통해 GPGPU를 이용하는 방식이다. 이 방법 역시 네이티브 OpenCL 라이브러리 내부에 오프로딩 전송 모듈이 추가 구현되어야하기 때문에 소스코드 수정이 필수적이다. 하지만 현재 GPU 벤더에서 디바이스 드라이버뿐만 아니라 라이브러리도 바이너리 형태로 제공하기 때문에 일반적인 방법으로 구현이 어렵다.

따라서, 이 논문에서는 그림 2에서와 같이 자바 OpenCL Wrapper 라이브러리를 자바 OpenCL 오프로딩 라이브러리로 수정하여 오프로딩을 수행 하는 방식과, 네이티브 OpenCL 라이브러리 위에 NDK를 이용한 오프로딩 라이브러리를 추가하는 방식 두 가지를 고려한다.

그림 2의 방법 1과 방법 2 방식 중 더 나은 오프로딩 레이어를 결정하기 위해서 두 방식간의 비교 분석이 필요하다.

첫 번째 비교 관점으로 이식성에 대해 살펴본다면, 방법 1의 경우가 방법 2의 경우보다 범용 오프로딩 프레임워크 구현 관점에서 이식성이 적다. 즉, 방법 1의 경우, 안드로이드 기반 모바일 응용 프로그램만을 지원하는 오프로딩 프레임워크로써 C 기반의 모바일 응용 프로그램은 지원할 수 없다. 하지만, 방법 2의 경우 오프로딩 라이브러리가 NDK를 이용하여 C를 기반으로 구현되기 때문에, C 기반의 모바일 응용 오프로딩 프레임워크를 구성한다는 관점에서 방법1보다 이식성이 높다.

두 번째 비교 관점으로 구현의 복잡도 측면에서 살펴본다면, 응용프로그램이 네트워크 문제로 오프로딩을 수행하지 못할 경우에 모바일 GPGPU 응용 프로그램은 로컬 GPGPU를 사용하여야 하는데, 이를 지원하기 위해선 자바 OpenCL 오프로딩 라이브러리에서 네이티브 OpenCL 라이브러리를 사용할 수 있도록 추가 구현이 필요하다. 이는 구현의 복잡도를 더 높이는 결과를 초래한다. 반면에 방법 2의 경우 네이티브 OpenCL 라이브러리와 NDK를 이용하여 기존에 구성이 완료되어있는 레이어에서 오프로딩 작업이 이루어지기 때문에 로컬 네이티브 OpenCL 라이브러리와의 연동 작업이 상대적으로 방법1에 비해 복잡도가 낮다.

세 번째 비교 관점으로 성능의 관점에서 살펴본다면, 방법 1의 경우 방법 2의 경우보다 레이어가 적어 오버헤드도 그만큼 적다는 장점이 있지만, 실험을 통해 오버헤드를 살펴보면 사실 그렇지 않다. 표 3은 방법 2의 추가 오버헤드의 정도를 실험한 결과이다. 실험환경으로 Intel Core i7-3770k (3.50GHz) CPU, 16GB RAM, Ubuntu 12.06 (64bit)를

사용하였다. 워크로드로 32 x 32 사이즈의 매트릭스 곱과 NBody를 사용하였다. NBody중 연산에 필요한 노드의 개수는 256개를 이용하였다. 표 1의 실험결과로부터 알 수 있듯이, 32 x 32 매트릭스 곱의 경우 전체 수행시간은 약 200ms의 수행시간이 소요되고 이 때 추가 레이어로 인한 오버헤드는 약 0.4ms로 워크로드의 성능영향이 거의 없고 이는 수행성능의 오차범위에 들어가는 수준이다. 뿐만 아니라, 32 x 32 매트릭스 곱은 워크로드의 사이즈가 작아 연산량도 적은 워크로드이다. 매트릭스곱 뿐만 아니라 Nbody의 경우도 비슷한 결과를 나타낸다. 연산에 사용되는 노드가 256개 인 것은 NBody 워크로드 중 연산량 정도가 낮은 것으로 이 때 발생하는 오버헤드도 전체 수행시간에 비해 약 4%정도로 전체 수행성능에 크게 영향을 미치지 않는 정도의 오버헤드가 발생한다.

(표 3) 방법2의 추가적인 레이어로 인한 두 워크로드(매트릭스 곱, Nbody)의 오버헤드 측정

	매트릭스 곱 (32x32)	Nbody (N=256)
전체수행시간 (ms)	200	2
오버헤드 (ms)	0.4	0.08

따라서, 방법 1과 방법2의 장단점을 비교한다면, 방법2가 방법1에 비해 오버헤드 발생하지만 그 정도가 미미하고(표 3), 또한 방법2가 방법1에 비해 이식성이 높고 구현 복잡도가 낮기 때문에 모바일 OpenCL 오프로딩 수행 방법으로 정한다.

3.2 오프로딩 구조

오프로딩의 원격지 GPGPU 데몬이나 오프로딩 방법은 이전 우리의 연구와 같은 형태를 갖는다.

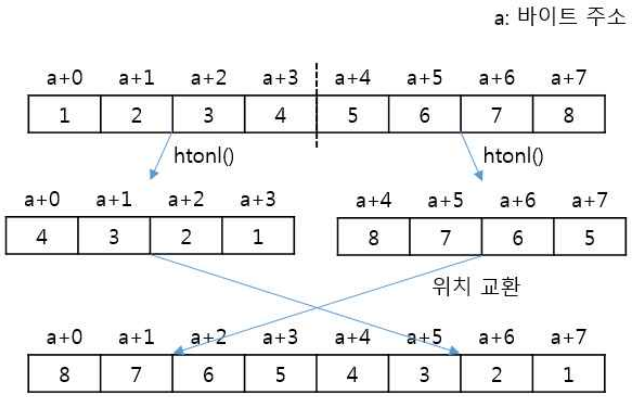
4. 구현

본 논문에서 구현한 모바일 GPGPU 오프로딩 프레임워크는 기본적으로 우리의 이전 연구와 같은 형태를 갖는다.

4.1 NDK를 이용한 포인터 처리

OpenCL의 경우 응용 프로그램에서 GPGPU를 컨트롤 할 수 있도록 구조체 변수를 제공하고 있다. 하지만 구조체 내부 정보를 공개하지 않기 위해서 OpenCL에서는 C의 포인터를 이용하여 구조체를 포인팅하고 포인팅된 메모리 주소를 라이브러리 인터페이스에 넘김으로써 GPGPU를 컨트롤하게 된다. 즉 GPGPU 컨트롤은 포인터를 통하여 이루어진다. 하지만, 자바에서는 포인터 개념이 없기 때문에 원격지의 구조체에 담긴 정보를 가지고 GPGPU를 컨트롤하기 위해선 원격지의 포인터를 처리하기 위한 방법이 필요하다. 이를 처리하기 위해 오프로딩 라이브러리 내에서 NDK에서 제공하는 인터페이스 클래스인 JNIEnv의 멤버함수를 이용한다. SetNativePointer() 함수를 이용하여 포인터 정보를 저장하고 이후에 해당 포인터 정보를 이용하여 연산이 필요할 경우 GetLongField()를 사용하여 포인터 정보를 가져온 후 원격지 GPGPU 컨트롤에 이용한다.

4.2 네트워크 바이트 순서 처리



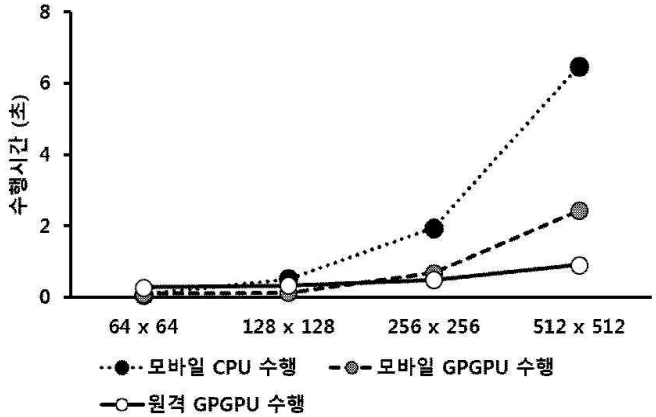
(그림 3) 리틀엔디안 8바이트 데이터로부터 빅엔디안 8바이트 데이터로의 변환과정

모바일 GPGPU 응용 프로그램의 오프로딩 수행시, 원격지 GPGPU를 컨트롤하기 위한 정보가 원격지로 전달되고 수행 결과가 원격지에서 로컬로 전달된다. 이 때 로컬 모바일 환경의 CPU와 원격지 서버 환경의 CPU가 다른 방식으로 데이터를 처리할 수 있다. 즉, 각 CPU가 데이터를 관리하는 방식이 리틀엔디안과 빅엔디안으로 서로 다를 수 있다. 따라서 오프로딩을 수행할 때, 바이트 순서를 고려하여 로컬과 원격지간의 데이터가 교환되어야 한다. 이미 소켓에서, hton(), ntoh() 계열 함수를 통하여 다른 엔디안 방식을 가진 CPU간에 소켓 통신을 지원하고 있지만 해당 함수들은 2바이트와 4바이트 데이터에 대해서만 지원하고 있기 때문에 8바이트 정보를 전송하기 위해선 추가적으로 네트워크 바이트 순서 처리를 위한 구현이 필요하다. 이미 앞 4.1 절에서 GPGPU 컨트롤을 위해 포인터 정보를 주고받음을 언급하였다. 64bit 머신에서 포인터의 사이즈가 8바이트이기 때문에 위의 구현은 필수적이다. 구현은 다음과 같다. 먼저 데이터를 보내는 주체가 되는 호스트가 리틀엔디안인지 빅엔디안인지 확인한다. 만약에 빅엔디안이라면 해당 데이터를 네트워크로 바이트 순서 처리 없이 전송한다. 만약 리틀엔디안이라면, 4바이트 단위의 두 파트로 나눈 후 각각에 대해 그림 3과 같이 htonl() 함수를 통하여 바이트 순서를 조절한다. 각 파트에 바이트 순서 조절 후 최종적으로 두 파트의 위치를 바꾼다.

5. 성능 평가

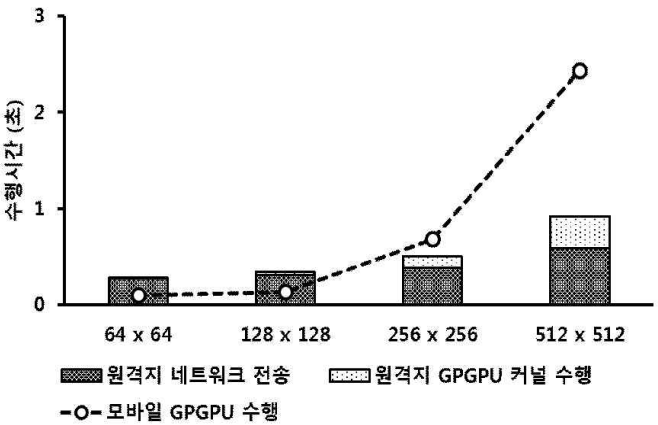
본 논문에서 사용한 실험환경으로 모바일의 경

우 Nexus 10 태블릿 디바이스 (CPU: ARM Cortex 15, GPU: Mali-T604, RAM: 2GB)를 사용하였고, 원격지 PC의 경우 CPU: intel I7-3770k (3.60Ghz), GPU: ATI Radeon 7970HD, RAM: 16GB 를 사용하였다. 모바일과 AP간의 네트워크 연결은 IEEE 802.11n 을 사용하였고, AP와 원격지 PC와의 연결은 Ethernet 100Mbps LAN을 사용하였다.



(그림 4) 매트릭스 곱에 대하여 모바일 CPU, 모바일 GPGPU, 원격 GPGPU 수행 성능 비교 (모바일 CPU 수행결과는 모바일에 GPGPU가 없는 경우의 결과임)

그림 4는 64x64부터 512x512까지 다양한 사이즈의 매트릭스 곱 워크로드에 대해서 모바일 CPU, GPGPU 수행과 원격 GPGPU 수행의 성능을 비교한 결과이다. 그림에서 살펴보면 매트릭스 곱의 사이즈가 작은 경우 모바일 수행과 원격 수행의 성능차이가 거의 없지만 매트릭스 곱의 사이즈가 커질수록 모바일 수행과 원격 수행의 성능 차이가 커진다. 이는 원격 수행 시 발생하는 추가 네트워크 전송 시간이 원격 수행으로 인한 커널 수행 시간의 이득보다 작아야 원격 수행의 이득이 생기기 때문이다.



(그림 5) 매트릭스 곱에 대하여 원격지 오프로딩 수행 시 네트워크 전송시간에 대한 프로파일링 결과와 모바일 GPGPU 수행시간의 비교

그림 5에서 살펴보면 이득 시점은 256 x 256 사이즈의 매트릭스 곱 이상인 경우부터 발생한다. 실제로 모바일 GPGPU 수행과 원격 GPGPU 수행 성능을 비교해봤을 때, 256x256 매트릭스 곱의 경우 수행시간이 약 1.36배, 512x512 매트릭스 곱의 경우 수행 시간이 약 2.66배의 차이가 발생한다.

6. 관련연구

오프로딩을 통한 원격수행의 지원은 크게 CPU 응용을 대상으로 하는 프레임워크와 GPGPU를 대상으로 하는 프레임워크로 나눌 수 있다. CPU 응용을 대상으로 한 연구의 예로 MAUI [6], CloneCloud [7], COMET [8]을 들 수 있다. MAUI의 경우는 원격수행을 위해 소스코드의 수정을 필요로 하고, CloneCloud의 경우엔 쓰레드의 일부분을 클라우드로 오프로딩하여 원격수행을 진행한다. COMET의 경우는 원격지와 로컬 간에 분산 공유 메모리를 지원함으로써, 한 쓰레드 전체를 원격으로 오프로딩하여 수행시킨다. 위 세 연구가 CPU 응용에 관련된 연구라면 rCUDA [9], GViM [10], VOCL [11]은 GPGPU 응용을 대상으로 한 연구이다. rCUDA의 경우 CUDA를 대상으로 한 연구이고, GViM의 경우 한 물리 머신 내에서의 오프로딩에 관한 연구이다. 즉, VMM의 지원을 받아 VM간의 오프로딩을 지원한다. VOCL의 경우 OpenCL 대상으로 한 오프로딩 지원 프레임워크이지만 모바일 환경을 지원하지 않는다.

7. 결론

본 논문에서는 OpenCL 기반 모바일 GPGPU 응용 프로그램의 오프로딩 수행을 지원하는 프레임워크를 제안하였고, 프레임워크의 오프로딩 레이어 결정 시 각 레이어의 장단점을 살펴봄으로써, 가장 적합한 레이어를 대상으로 프레임워크를 구성하였고, 구현 시 발생할 수 있는 이슈들을 살펴보았다.

이 프레임워크는 모바일 GPGPU 응용이 없는 상황에서도 GPGPU를 이용 가능하도록 지원할 뿐만 아니라 로컬에서 모바일 GPGPU 응용 프로그램의 수행 성능을 크게 향상시켰다. 실험결과 워크로드의 연산량이 증가할수록 이 프레임워크를 통한 이득이 증가함을 알 수 있다.

Acknowledgement

본 연구는 2013년도 미래창조과학부 산업원천기술개발 사업의 지원을 받아 수행 중인 디바이스 소셜리티를 이용한 무설정 방식 이종사간 상호연동 기술 개발 (10041801) 과제의 일환으로 수행하였음.

Ⅱ 참고문헌

[1] CES 2012, http://www.nvidia.com/object/ces2012.html

[2] 박세진, 마정현, 박찬익, “GPGPU 응용의 고성능 원격수행을 위한 OpenCL 기반 오프로딩 프레임워크”, 한국차세대컴퓨팅학회 논문지 Vol.9 No.2, pp.44-53, 2013년 4월

[3] Top 500 List, http://www.top500.org/list/2013/11/
[4] JOCL, http://www.jocl.org/
[5] OpenCL, http://www.khronos.org/opencl/
[6] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. “MAUI: making smartphones last longer with code offload.”, In proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)
[7] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. “CloneCloud: elastic execution between mobile device and cloud.” In proceedings of the sixth conference on Computer systems (EuroSys '11)
[8] Gordon, Mark S., et al. "COMET: code offload by migrating execution transparently.“ In proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI '12)
[9] Jose Duato, Antonio J. Pena, Federico Sila, Rafael Mayo and Enrique S. Quintana-Orti, “rCUDA: Reducing the number of GPU-based accelerators in high performance clusters.”, In proceedings of International Conference on High Performance Computing and Simulation, pp.224-231, June 2010.
[10] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharche, Niraj Tolia, Vanish Talwar, Parthasarathy Ranganathan, “GViM: GPU-accelerated virtual machines”, In proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, pp.17-24, 2009
[11] Shucai Xiao, Balaji, P., Qian Zhu, Thankur, R., Coghlan, S., Heshan Lin, Gaojin Wen, Jue Hong, Wu-chun Feng; “VOCL: An optimized environment for transparent virtualization of graphics processing units”, In proceedings of Innovative Parallel Computing, pp.1-12 2012

Ⅱ 저자소개

◆마정현

• 2012년 아주대학교 정보 및 컴퓨터공학과(학사).
• 2012년~현재 포항공과대학교 컴퓨터공학과 통합과정
• 관심분야: 운영체제, GPGPU, 가상머신

◆박세진

• 2007년 금오공과대학교 소프트웨어공학과(학사).
• 2007년~현재 포항공과대학교 컴퓨터공학과 통합과정
• 관심분야: 운영체제, 가상머신, 임베디드 시스템, GPGPU

◆박찬익

• 1983년 서울대학교 전자공학과(학사).
• 1985년 한국과학기술원 컴퓨터공학과(석사).
• 1988년 한국과학기술원 컴퓨터공학과(박사).
• 1989년~현재 포항공과대학교 정교수
• 관심분야: Storage systems, System security, Cloud computing & Virtualization, Embedded Linux