Journal of the Korea Academia-Industrial cooperation Society
Vol. 15, No. 7 pp. 4545-4553, 2014

http://dx.doi.org/10.5762/KAIS.2014.15.7.4545
ISSN 1975-4701 / eISSN 2288-4688

멀티코어형 모바일 GPU의 작업 분배 및 효율성 분석

임효정[1], 한동건[1], 김형신[1*]
[1]충남대학교 컴퓨터공학과

Analysis of Job Scheduling and the Efficiency for Multi-core Mobile GPU

Hyojeong Lim[1], Donggeon Han[1], Hyungshin Kim[1*]
[1]**Department of Computer Science and Engineering, Chungnam National University**

요 약 모바일 GPU가 발전함에 따라 멀티코어 GPU를 효과적으로 최적화하는 것은 스마트폰의 성능을 높이는데 있어 중요한 문제가 되고 있다. 하지만 대부분의 모바일 GPU에 관한 연구는 싱글코어 모바일 GPU에 대해 다루고 있거나, GPU 공급자에 의한 제한적인 연구만을 다루고 있다. 따라서 본 논문에서는 멀티코어 GPU의 작업 분배 패턴과 효율성 분석을 통해 성능향상의 가능성에 대한 분석을 수행하였다. 실험은 DS-5 Streamline을 사용하여 시스템 사용자 인터페이스를 조작하였을 때, GPU의 코어 수의 변화에 따른 그래픽 처리 소요 시간을 측정한 실험과 GPU의 코어 수에 따른 작업 분배 패턴에 대한 실험을 수행하였다. 프로파일링 결과, GPU의 코어수가 더 증가했음에도 불구하고 그래픽 애플리케이션을 실행하는데 요구되는 전체 소요시간이 증가하는 경우를 발견하였다. 또한 GPU가 그래픽을 처리할 때, 약 4ms의 오버헤드가 CPU와 GPU 사이의 통신에서 발생하고, GPU 내부 드라이버의 활동으로 인한 지연이 발생했음을 확인하였다. 따라서 본 논문에서 GPU 동작의 비효율성에 대한 분석결과는 앞으로의 모바일 멀티코어 GPU의 연구에 있어 참고가 될 수 있을 것이라 예상한다.

Abstract Mobile GPU has led to the rapid development of smart phone graphic technology. Most recent smart phones are equipped with high-performance multi-core GPU. How a multi-core mobile GPU can be utilized efficiently will be a critical issue for improving the smart phone performance. On the other hand, most current research has focused on a single-core mobile GPU; studies of multi-core mobile GPU are rare. In this paper, the job scheduling patterns and the efficiency of multi-core mobile GPU are analyzed. In the profiling result, despite the higher number of GPU cores, the total processing time required for certain graphics applications were increased. In addition, when GPU is processing for 3D games, a substantial amount of overhead is caused by communication between not only the CPU and GPU, but also within the GPUs. These results confirmed that more active research for multi-core mobile GPU should be performed to optimize the present mobile GPUs.

Key Words : Job scheduling, Mobile GPU, Multi-core GPU, Profiling

1. 서 론

최근 스마트폰의 고화질 디스플레이 발달로 사용자들은 더욱 부드러운 터치감의 사용자 인터페이스와 사실과 같은 선명하고 실감나는 그래픽 영상을 선호하게 되었다. 이러한 사용자의 요구 변화로 디바이스에서 처리해야 할 데이터와 연산의 양이 기하급수적으로 증가하였다. 이러한 문제를 해결하는데 모바일 GPU(Graphic Processing Unit)의 등장은 큰 도움을 주었다. 고속 병렬 처리에 특화된 GPU를 사용함으로써 그 동안 프로세서에 치중되었던 계산 부담을 분산시키고, 연산속도를 가속화하는 효과를 얻을 수 있게 되었다. 모바일 기기의 성능 향상을

이 논문은 2011 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업입니다. (No. 2011-0013115)
*Corresponding Author : Hyungshin Kim(Chungnam National Univ.)
Tel: +82-10-2085-3860 email: hyungshin@cnu.ac.kr
Received March 21, 2014 Revised June 16, 2014 Accepted July 10, 2014

위해 모바일 GPU는 싱글코어에서 8개의 코어를 갖는 멀티코어 GPU까지 발전하였다. 이로 인해 멀티코어 GPU의 효율적 관리는 모바일 기기의 성능 향상에서 중요한 부분을 차지하게 되었다.

대부분의 GPU에 관한 연구는 데스크톱 환경에서 이루어졌다. 데스크톱 GPU의 성능 개선 연구로 NVIDIA의 경우 GPU 자체의 구조적인 성능 향상 연구와 더불어 CUDA 프로그래밍이라는 GPU의 성능을 최대로 이용하기 위한 병렬프로그래밍 기법을 고안하여 효율적인 GPU의 사용에 대해 꾸준히 연구하였다[1-3]. GPU 자원관리에 관한 연구에는 [4,5]가 있다. [5]에서는 타임그래프(Time Graph)를 이용하여 디바이스 드라이버에서 GPU 명령어 기반의 스케줄러를 통합하는 연구와 명령어 기반 스케줄러 모델의 오버헤드를 다루고 있다. [4]는 사용자영역 실시간 모델인 RGEM을 이용하여 사용자영역의 한계를 극복하여 GPU 자원을 관리하는 것을 연구하였다. 이와 비슷한 연구로 효율적인 병렬처리를 위해 GPU에서 R-Tree 질의 처리 기법을 이용한 연구도 있다[5]. 또한 고성능 GPU의 성능저하 현상에 대한 분석 연구도 존재한다[7]. 이 논문은 GPU의 성능에 부정적인 영향을 미치는 요인들을 다섯 가지로 분류하여, 각 요인들이 성능에 미치는 영향을 분석하였다. 데스크톱 GPU에 관한 연구는 이러한 연구 이외에 운영체제나 아키텍처 관점에서의 접근이나 스케줄링, 데이터 흐름에 관한 연구 등 다양한 측면에서 연구가 이루어져왔다. 하지만 데스크톱 환경의 GPU는 특정 하드웨어의 지원을 받아 동작하고, CUDA나 OpenCL[8]과 같은 GPU 컴퓨팅 프레임워크들은 아직까지 모바일 GPU에 바로 적용이 불가능하다. 따라서 데스크톱 환경에서 GPU에 관한 연구를 모바일 GPU에 적용하기가 어렵다.

데스크톱 GPU의 연구와 함께 모바일 GPU에 관한 연구도 진행되고 있다. 모바일 GPU의 분석을 통한 성능 평가 연구로는 [9]이 존재한다. 이 논문에서는 PowerVR SGX530, Adreno 200, GeForce ULV 세 종류의 모바일 GPU가 탑재된 스마트폰의 3D 그래픽 게임 처리 성능을 비교 분석하였다. 그러나 이 연구에서 비교 분석된 모바일 GPU들은 모두 싱글코어 제품이다. 이와 같이 대부분의 모바일 GPU에 관한 연구들은 싱글코어 GPU를 중심으로 이루어졌기 때문에 현재 상용화된 스마트 기기에서 사용하는 멀티코어 GPU에 곧바로 적용하기 힘들다. 그러므로 여전히 멀티코어 GPU 연구의 필요성은 존재한다.

대부분의 연구가 싱글코어 환경에서 진행 되었지만, 최신 기술에 발 맞춰 모바일 멀티코어에서 연구도 진행되었다. 하지만 이러한 연구들은 GPU 공급자에 의한 자체적인 연구가 대부분을 차지하고 있다. 이러한 연구들은 GPU 공급자의 특성상, 비공개적이고 특정 시스템이나 디바이스에 대해서 다루는 경우가 많다. 그렇기 때문에 이러한 연구는 추후, 모바일 멀티코어 GPU 연구를 위한 참고가 제한적이고, 연구의 실험을 재연하고 재활용하기가 어렵다는 문제가 있다.

따라서 본 논문에서는 모바일 멀티코어 GPU의 프로파일링 환경을 구성하고, 측정한 프로파일링 결과를 바탕으로 멀티코어 GPU의 작업처리 시 작업분배에 대한 분석과 병렬처리가 효율적으로 이루어지고 있는지 분석하였다.

본 논문에서는 작업 분배와 그 효율성 분석을 위해 두 가지 실험을 했다. 첫 번째 실험에서는 시스템 사용자 인터페이스를 조작했을 경우에 모바일 GPU의 코어 수에 따른 전체 그래픽 처리가 완료되는 소요시간을 측정하는 실험을 하였다. 두 번째 실험에서는 첫 번째 실험에서 스와핑을 했을 때, 모바일 GPU의 코어 수에 따른 작업 분배 패턴을 실험하였다. 첫 번째 실험의 결과 코어의 수가 증가함에도 불구하고 전체 소요시간이 그에 비해 줄어들지 않고 오히려 증가하는 경우가 발생함을 확인 할 수 있었다. 두 번째 실험의 결과 버텍스 프로세서(Vertex processor, VP) 작업 사이에 유휴시간이 있었고, 해당 유휴시간동안 CPU의 활동이 증가하는 모습을 확인하였다. 또 코어가 네 개일 경우에는 각 코어별 작업이 동시에 시작했음에도 불구하고 작업이 끝나는 시간에 차이가 발생하였다. 즉, 종합적으로 살펴보면 CPU와 GPU 사이의 통신으로 오버헤드가 발생하여 코어별 작업 처리 시간의 지연이 있었고, 멀티코어 환경에서의 작업처리 효율성이 좋지 않다는 것을 확인하였다.

본 논문의 모바일 멀티코어 GPU의 작업 분배에 대한 효율성 분석 결과는 향후 모바일 멀티코어 GPU 성능 향상 기법 연구의 방향을 설정하는데 참고가 될 수 있고, 효율적인 모바일 멀티코어 GPU의 설계 방법을 연구하는 것에 참고가 될 수 있다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어, 2장에서는 본 논문과 관련된 연구들에 대해 소개하고, 3장에서는 모바일 GPU의 전반적인 구조에 대해 살펴본다.

4장에서는 본 논문에서 제안하는 모바일 GPU 프로파일링 환경 구축에 대하여 설명하고, 5장에서는 프로파일링 실험 결과 및 분석 내용을 보여준다. 마지막으로 6장에서 결론을 내린다.

2. 관련연구

2000년 후반부터 사용되기 시작 된 모바일 GPU의 짧은 역사와는 달리, 데스크톱 환경에서의 GPU는 몇 십 년 전부터 사용되기 시작하였다. 때문에 기존의 GPU 관련 연구들은 대부분 데스크톱 환경을 중심으로 다양한 이슈에 관한 연구들이 수행되어 왔다.

데스크톱의 GPU 프로파일링 툴에 관한 연구로 NVIDIA에서 무료로 제공하는 CUDA 툴킷이 있다. CUDA 툴킷은 프로파일링 된 데이터의 지능적인 자동 성능 분석 기능을 제공한다[10]. 또한 GPU-Z나 GPU Shark와 같이 일반 데스크톱 사용자들도 간편하게 GPU의 상태를 상세히 모니터링 할 수 있는 프로그램도 일반화 되어있어 데스크톱에서의 GPU 프로파일링은 모바일 환경보다 비교적 쉽다[9].

데스크톱 GPU의 성능 개선 연구 역시 활발히 이루어져왔다. NVIDIA의 경우 GPU 자체의 구조적인 성능 향상 연구와 더불어 CUDA 프로그래밍이라는 GPU의 성능을 최대로 이용하기 위한 병렬프로그래밍 기법을 고안하여 효율적인 GPU의 사용에 대해 꾸준히 연구하였다[1][2][3]. 다른 GPU 컴퓨팅 프레임워크로는 OpenCL[8]이 있다. 그 외의 연구로 데스크톱 환경에서의 GPU의 전력관리를 위한 연구가 존재한다. [12]에서는 GPU 코어의 수의 증가에 따라 증가하는 전력 소모를 측정하고, 효율적인 전력관리를 위한 전력추정모델을 제시하였으며 실험 결과 실제와 비슷한 수준의 전력추정 결과를 보였다.

모바일 GPU의 분석을 통한 성능 평가의 연구로는 [9]이 존재한다. 이 논문에서는 PowerVR SGX530, Adreno 200, GeForce ULV 세 종류의 모바일 GPU가 탑재된 스마트폰의 3D 게임 그래픽 처리 성능을 비교 분석하였다. 특히 GPU의 기능을 세분화 하여 각각의 기능을 사용 했을 때와 그렇지 않았을 때를 나누어 그래픽 처리 성능을 비교하면서 그래픽 파이프라인에서 병목현상이 발생하는 원인을 추적하였다. 그러나 이 연구에서 비교 분석된 모바일 GPU들은 모두 싱글코어 제품이다. 따라서 이 논문의 분석 결과가 현재 상용화된 멀티코어 제품에 적용되기 힘들며, 여전히 멀티코어 GPU 연구의 필요성은 존재한다.

GPU의 성능분석을 위한 GPU 프로파일링 방법에는 성능감시유닛(Performance Monitoring Unit, PMU)을 이용하여 프로파일링 하는 방법, 모바일 그래픽 벤치마크 응용 프로그램을 이용한 방법, 그리고 GPU 공급자가 제공하는 GPU 프로파일링 툴을 이용한 프로파일링 방법이 있다.

성능감시유닛을 이용한 방법은 퍼포먼스 레지스터에 직접 접근하여 데이터를 수집할 수 있어 정확한 프로파일링이 가능하다. 하지만 이 방법은 성능감시유닛의 하드웨어적 구조와 실제 내부 레지스터의 정보 없이는 프로파일러 구현이 불가능하다는 단점이 있다.

모바일 그래픽 벤치마크 응용 프로그램을 이용한 방법은 그래픽의 성능 측정뿐만 아니라 프로세서와 I/O등의 성능 측정 기능까지 제공한다. 성능 측정 후에는 이해하기 쉬운 스코어로 프로파일링 결과를 출력한다. 하지만 프로그램이 각 기기의 기본 해상도 값으로만 성능을 측정하기 때문에 상대적으로 해상도가 높은 모바일 디바이스에서는 정확한 성능 측정이 불가능하다. 또한 높은 레벨에서의 결과를 보여주기 때문에 세밀한 성능 측정이 어렵다. 대표적인 그래픽 벤치마크 프로그램으로는 Quadrant 벤치마크[12]와 GL 벤치마크[13]가 있다.

GPU 공급자가 제공하는 GPU 프로파일링 툴을 이용한 방법은 내부에 성능감시유닛을 이용하는 코드를 포함하고 있기 때문에 세밀하고 정확한 성능 측정이 가능하다. 그리고 GPU 공급자가 제공하는 툴이므로 해당 GPU에 맞는 별도의 DDK(Driver Development Kit)만 준비되어있다면 성능감시유닛의 구조나 내부 레지스터의 정보 없이도 프로파일링이 가능하다.

따라서 본 논문에서는 정밀한 성능 측정이 가능하고, 내부 레지스터의 정보 없이도 프로파일링이 가능한 GPU 프로파일링 툴을 이용하여 프로파일링 실험 환경을 구성하였다.

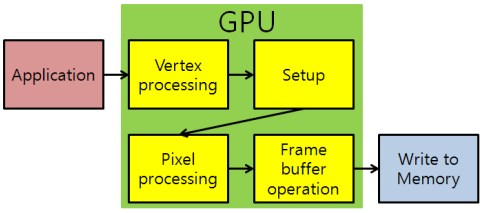
3. 배경지식

이 절에서는 모바일 GPU의 전반적인 구조에 대해 기술한다. 본 논문에서는 Mali-400MP GPU를 대상으로 실

험하였다. 먼저 일반적인 모바일 GPU의 구조와 그래픽 파이프라인의 동작을 삼각형을 그리는 예제를 사용하여 기술한다. 그 후 Mali-400MP의 하드웨어적 구조와 디바이스 드라이버 구조를 다루었다.

3.1 그래픽 파이프라인

모바일 GPU에서의 그래픽 처리는 그래픽 파이프라인을 거치며 수행된다. Fig. 1은 일반적인 모바일 그래픽 파이프라인의 쉐이더(shader) 프로그래밍이 가능한 구조이다[14].



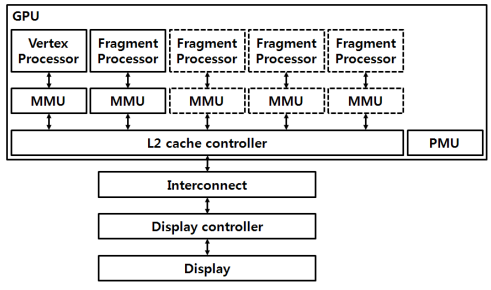
[Fig. 1] Graphic pipeline

OpenGL ES의 API를 사용하여 작성된 응용 프로그램은 프로그램 실행 중 GPU에서 렌더링이 필요한 그래픽 데이터를 GPU에게 전송한다. 데이터를 받은 GPU는 가장 먼저 버텍스 쉐이더(Vertex shader)를 이용하여 삼각형의 각 점들의 위치를 계산한다. 여기서 버텍스 쉐이더는 물체의 정점 정보를 이용하여 수학적 연산을 수행한다. 또한 객체 상의 수학연산을 통해 길이, 높이, 깊이에 해당하는 정점 데이터 값을 결정한다. 그리고 이 정보를 바탕으로 정점의 위치를 결정하고 조명의 특성, 텍스처 좌표의 변환 등의 작업을 한다. 버텍스 쉐이더는 연산한 결과를 베어링(vaying) 배열에 저장한다. 추후 이 배열은 프레그먼트 쉐이더(Fragment shader)에 저장된다. 프레그먼트 쉐이더는 렌더링 될 각 픽셀의 최종 출력 될 색상 값을 결정한다. 프레그먼트 쉐이더는 이를 위해 광원 효과 적용, 텍스처로부터 색 정보 수집, 투명처리 등의 작업을 처리한다. 셋업 유닛은 삼각형 각 점들의 위치를 계산한 후, 점들을 삼각형의 모양으로 만들고 다양한 상수를 이용하여 래스터화 과정을 거친다. 틀이 만들어진 삼각형 내부의 각각의 픽셀들은 픽셀 프로세싱 유닛에 의해 픽셀 값들이 결정되며, 작업이 끝나면 몇 가지 프레임 버퍼 오퍼레이션 과정이 수행된다. 그 후 최종적으로 메모리에 저장되는 과정이 일반적으로 그래픽 파이프라인에서 수행된다.

3.2 Mali-400MP GPU의 하드웨어 구조

ARM사의 Mali-400MP 멀티 코어 GPU는 한 개의 버텍스 쉐이더와 네 개의 프레그먼트 쉐이더를 가진 모바일 멀티코어 GPU이다. Fig. 2는 Mali-400MP의 간단한 내부 구조이다.

VP는 그래픽 파이프라인 정점들의 연산 단계를 담당한다. 이것은 기본 데이터들의 리스트를 생성하고, 프레그먼트 프로세서(Fragment processor, FP)에서 연산에 필요한 폴리곤 리스트, 버텍스 데이터가 담긴 구조체를 구성한다. FP는 Fig. 1의 그래픽 파이프라인에서 래스터화와 프레그먼트 프로세싱 단계를 담당한다. FP는 VP에서 생성한 데이터 구조체와 폴리곤 리스트를 이용하여 프레임 버퍼에 출력할 최종 이미지를 생성한다.



[Fig. 2] Mali-400MP hardware architecture

메모리관리유닛(Memory Management Unit, MMU)은 각 프로세서 별로 할당이 되며, 모든 프로세서들은 메모리에 접근하기 위해 메모리관리유닛을 거쳐야 한다.

Mali-400MP GPU는 L2 캐시를 가진다. 이를 통해 메모리 대역폭 사용량을 줄이고, 메모리 접근에 드는 비용을 줄여 전력 소모를 낮춘다.

성능감시유닛(Power Management unit, PMU)은 L2 캐시 컨트롤러와 각 프로세서들의 전력을 개별적으로 제어하여 GPU의 전력 관리를 돕는 장치이며, 프로그램 작성 가능한 구조로 되어 있다.

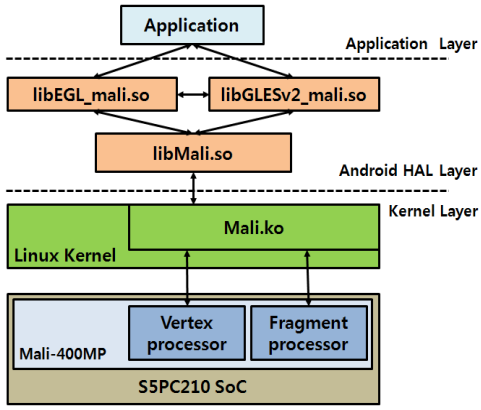
3.3 Mali-400MP의 디바이스 드라이버 구조

Mali-400MP를 위한 디바이스 드라이버는 크게 안드로이드 시스템과의 통신, VP를 위한 작업 처리, FP를 위한 작업 처리, 메모리 관리 그리고 성능감시유닛을 통한

전력 관리 기능을 갖는다.

안드로이드 시스템과 Mali-400MP 디바이스 드라이버의 통신은 일반적인 디바이스 드라이버와 동일한 방식인 ioctl(I/O Control) 인터페이스를 이용한다. 안드로이드 응용 프로그램으로부터의 GPU사용 요청은 ioctl 명령을 통해 GPU에게 데이터를 전송한다. Fig. 3은 Mali 드라이버의 전체 스택이다.

ARM사에서는 Mali 디바이스 드라이버를 두 가지로 분류한다. 하나는 ARM에서 오픈소스로 제공하고 있는 커널 영역 디바이스 드라이버이고, 나머지는 안드로이드 HAL(Hardware Abstraction Layer)에 해당하는 사용자 영역 디바이스 드라이버다. Fig. 3에서 안드로이드 HAL 영역은 세 가지의 동적 라이브러리로 이루어져있다. HAL 영역의 드라이버는 그래픽 처리 요청을 위해 mali.ko에게 ioctl 명령어로 데이터를 전송한다. 명령을 받은 mali.ko는 각 프로세서에게 작업의 시작을 알리고 데이터를 넘겨준다.



[Fig. 3] Mali device driver stack

4. 프로파일링 환경 구축

실험을 위한 환경과 프로파일링에 사용된 도구는 다음과 같다. 실험을 위한 가상화 환경으로 VMware Player v3.14에서 호스트 운영체제로 Ubuntu10.10을 사용하였다. 안드로이드 운영체제로 Android 4.0.4 ICS를 사용하였고, 실험 보드로 HBE-SM5-S4210을 사용하였다. 실험을 위한 모바일 멀티코어 GPU는 애플리케이션 프로세서 Exynos4210에 탑재된 Mali-400MP GPU를 프로파일링 대상으로 선정하였다. Mali-400MP는 최근 국내에서 많이 사용되고 있는 GPU이다. 프로파일링 방법으로 앞선 2장에서 제시한 GPU 프로파일링 툴을 이용한 방법을 사용하였다. GPU 프로파일링 툴 중, DS-5 Streamline[15]을 이용하여 연구를 진행하였다. DS-5 Streamline은 Mali-400MP GPU의 모든 코어에 대한 개별 프로파일링 기능을 지원하고, 각 코어별 활용률과 그래픽 파이프라인에서 단계별로 사용한 사이클 등을 측정할 수 있다.

Table 1은 Mali-400MP GPU를 위한 프로파일링 환경 정보이다.

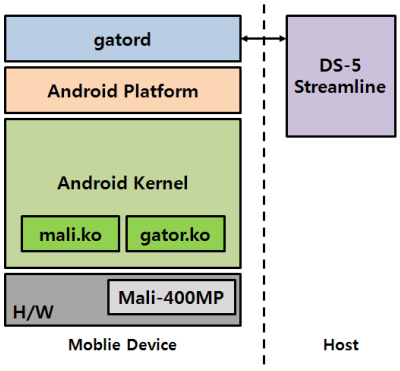
[Table 1] Profiling environment

Component	Description
VMware	VMware Player v3.14
Host OS	Ubuntu 10.10
Experiment Board	HBE-SM5-S4210
Application processor	Exynos4210
Moblie GPU	Mali-400MP 267MHz
Android OS	Android 4.0.4 ICS
Profiling tools	DS-5 Streamline

본 논문의 Mali-400MP의 프로파일링은 두 가지 형태로 수행되었다. 첫 째는 ARM사의 DS-5 Streamline이며, 두 번째는 DS-5 Streamline에서 제공하지 않는 정보를 위해 드라이버 내에 프로파일링 코드를 삽입하는 방법이다.

Fig. 4는 DS-5 Streamline을 이용하여 구축한 프로파일링 환경의 간략한 구조이다. DS-5 Streamline을 이용한 프로파일링 환경에서 모바일 디바이스에서 데이터를 지속적으로 수집하기 위한 두 개의 프로파일링 보조 프로그램을 설치해야한다. 프로파일링 보조 프로그램인 gatord와 gator.ko는 모바일 디바이스 상에서 동작하며 프로파일링을 돕는다.

gatord는 안드로이드 디바이스에서 프로파일링을 위해 동작하는 NDK(Native Development Kit) 응용 프로그램이다. 또한 gatord는 DS-5 Streamline과 TCP 통신을 하여 프로파일링 데이터를 전송하는 기능을 한다. gator.ko는 프로파일링에 필요한 리소스들을 준비하고 수집할 데이터를 정의한다. 그리고 gator.ko는 Mali 디바이스 드라이버의 프로파일링을 위한 엔트리 포인트로써 프로파일링 할 내용을 전송하고 프로파일링을 시작하게 하는 역할을 한다. 또한 프로파일링 된 데이터를 디바이스에 임시로 저장했다가 작업이 끝나면 gatord를 통해 호스트로 전송한다.



[Fig. 4] Profiling structure for DS-5 Streamline

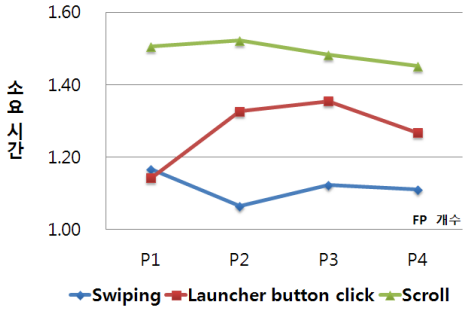
DS-5 Streamline을 사용하기 위해서는 커널의 설정 파일 수정이 필수적이다. 커널의 설정 정보가 제대로 설정되어 있지 않으면 gatord와 gator.ko의 생성이 불가능하기 때문에 커널 설정을 수정한 후 커널을 새로 컴파일 한다. 그리고 두 개의 프로파일링 보조 프로그램인 gatord와 gator.ko를 빌드한다. 이때 빌드를 위해 gatord는 안드로이드 NDK를 필요로 하고, gator.ko는 커널소스 정보와 Mali 디바이스 드라이버소스 정보를 필요로 한다. 우리는 오픈 소스인 Mali DDK를 이용하여 Mali-400MP의 디바이스 드라이버와 DS-5 Streamline을 위한 프로파일링 인터페이스를 포팅하였다. Mali 디바이스 드라이버의 프로파일링을 위해서 드라이버소스 내에 프로파일링에 필요한 코드를 삽입하였다. 삽입된 코드로는 _mali_osk_profiling_start(), _mali_osk_profiling_stop(), 그리고 _mali_osk_profiling_get_event()가 있다. _mali_osk_profiling_start()는 프로파일링을 시작하는 동작을 하고, _mali_osk_profiling_stop()은 프로파일링을 멈추는 동작을 한다. 그리고 _mali_osk_profiling_get_event()는 프로파일링에서 시간 이벤트 정보를 가져온다.

또한 DS-5 Streamline에서 제공하지 않는 기능의 프로파일링을 위해 Mali 디바이스 드라이버 내에 코드를 추가하였다. Mali 디바이스 드라이버에 삽입한 프로파일링 코드는 각 프로세서가 할당받은 작업들의 비율을 측정하기 위한 코드이다. 드라이버에서 코드 삽입 위치는 실제 프로세서에게 작업 시작을 지시하는 함수인 mali_core_renderunit_register_write() 함수 바로 다음에 위치한다.

5. 프로파일링 결과 및 분석

5.1 GPU 코어 수에 따른 그래픽 처리의 오버헤드

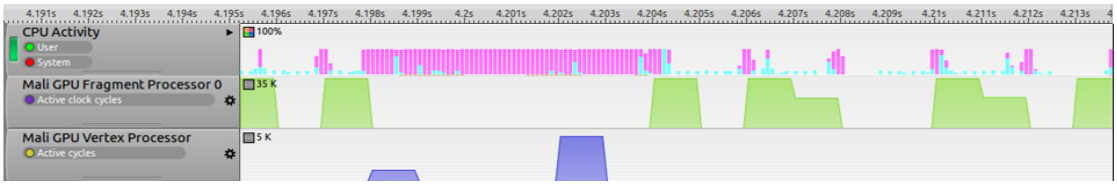
실험에 측정 할 항목은 시스템 사용자 인터페이스이다. 시스템 인터페이스의 세부 실험 항목으로 홈 화면에서 화면 넘김 동작(스와핑), 전체 프로그램 목록을 보여주는 버튼을 누르는 동작, 시스템 설정 메뉴의 상단부터 하단까지 스크롤을 조작하는 것이 있다. 이때 매 실험 시, 캐시를 비워 다른 요소로 인한 오버헤드의 발생을 방지했고, 동일한 결과 화면을 출력하도록 실험하였다. 각각의 세부 항목을 프로파일링 한 후 GPU의 코어 수에 따라 전체 완료에 소요되는 시간을 비교하였다. 실험은 다음과 같이 진행하였다. 우선 _mali_osk_profiling_start()함수를 사용하여 프로파일링을 시작한다. _mali_osk_profiling_stop()를 통해 프로파일링을 마친 뒤, _mali_osk_profiling_get_event()를 통해 각 세부항목의 소요시간을 측정하였다.



[Fig. 5] User interface execution time with varying GPU cores

Fig. 5는 시스템 사용자 인터페이스의 그래픽 처리에 필요한 전체 소요시간을 코어 수에 따라 비교한 그래프다. 시스템 사용자 인터페이스에서 앞서 설명한 세 개의 항목에 관하여 전체 동작이 완료되는데 소요된 시간을 측정하였다. Fig. 5의 세로축은 전체 소요시간을 의미하고, 가로축의 P1, P2, P3, P4는 각각 프로세서가 한 개, 두 개, 세 개, 네 개일 경우를 의미한다.

결과 그래프를 보면, 전체적으로 코어의 수가 증가함에도 불구하고 전체 소요시간은 그에 비해 크게 줄어들지 않는 결과를 보여주었다. 특히 전체 프로그램 목록을 보여주는 버튼을 누르는 동작의 경우 코어가 네 개일 때



[Fig. 6] GPU activity with 1 FP core

[Fig. 7] GPU activity with 2 FP cores

[Fig. 8] GPU activity with 4 FP cores

가 한 개일 때보다 오히려 시간이 더 소요되는 모습을 보였다.

코어의 수가 증가함에 따라 성능이 향상되어 그래픽 처리를 하는데 소요되는 전체 시간이 줄어들 것이라는 예상에 반하는 실험 결과를 통해, 코어 수의 증가와 함께 그래픽 처리 속도에 치명적인 영향을 끼치는 오버헤드가 있음을 추측할 수 있다.

5.2 GPU 코어 별 작업 분배 패턴

GPU 코어 수에 따른 그래픽 처리의 오버헤드를 측정한 실험 결과를 통해 확인된 오버헤드의 원인을 파악하기 위해 DS-5 Streamline을 이용하여 실험하였고, 프로파일링 결과를 통해 구체적인 원인을 분석하였다. 5.2절의 실험은 멀티코어 GPU를 사용했을 때 GPU 코어의 수에 따른 작업 분배 패턴에 관한 실험을 하였다. 실험은 안드로이드 홈 스크린에서 동일한 속도의 스와핑 동작을 취했을 동안의 GPU 동작을 측정하였다.

Fig. 6는 FP 코어가 한 개일 때의 스와핑을 수행했을 때의 GPU 동작 측정 결과이다. 해당 그래프 가로축은 시간의 흐름을 표현한 것이다. 세로축의 첫 번째 영역은 프로세서의 사용률을 나타낸다. 두 번째와 세 번째 영역은 각각 FP와 VP의 활동을 나타낸 그래프이다. 두 번째와 세 번째 영역의 세로축은 FP와 VP에서 소요된 사이클의 수를 나타낸 것이다. Fig. 6에서 프로세서가 급격히 사용률이 증가 하였을 때, VP와 FP의 활동을 보면, 4.199s에서 4.202s 까지 3ms 동안의 유휴 상태가 있다. 이것은 데이터를 전송하기 위해 프로세서와 GPU 사이의 통신이 발생하였고 그로 인해 작업 지연이 발생했다고 볼 수 있다.

Fig. 7은 FP 코어가 두 개일 때의 GPU 동작 측정 결과이다. 첫 번째 영역은 프로세서의 사용률을 나타낸 것

이다. 두 번째와 세 번째 영역은 각각 시간흐름에 따른 FP0, FP1에 대한 활동 그래프이고, 네 번째 영역은 VP의 시간흐름에 따른 활동 그래프이다. 이때, 각 영역의 세로축은 FP와 VP에서 작업을 처리할 때 소요된 사이클의 수를 나타낸다. FP 코어가 두 개일 때도 코어가 한 개일 때와 마찬가지로 VP 처리 구간 사이에 작업이 없음을 확인할 수 있었다. 또한 각각의 FP에서 두 개의 작업을 처리한 후 4ms의 시간 동안 모두 유휴상태에 들어갔다가 다시 작업을 할당받아 처리하는 모습을 볼 수 있다. 코어가 세 개인 경우에도 앞선 결과와 마찬가지로 VP 작업 사이의 모든 FP가 유휴상태였다.

코어가 네 개일 때의 프로파일링 결과는 Fig. 7과 같다. Fig. 8의 첫 번째 영역은 프로세서의 사용률을 나타낸 것이다. 두 번째부터 다섯 번째 영역까지는 FP0부터 FP3 가지의 시간대 별 활동 그래프이다. 여섯 번째 영역은 VP의 활동을 시간에 따라 나타낸 그래프이다. 이때 각 영역의 세로축은 Fig. 6, Fig. 7과 동일하게 각 프로세서에서 작업을 처리하는데 소요된 사이클의 수를 나타낸다. Fig. 7과 Fig. 8에서, 유휴시간 B에서는 CPU와 GPU 간의 통신으로 인한 오버헤드가 발생하였다. 유휴시간 A 영역에서는 CPU와 GPU 사이의 통신 오버헤드가 없지만 지연이 발생했는데, 이 원인으로 GPU 드라이버 내부에서의 처리를 위한 작업으로 인해 지연이 발생한 것으로 추정된다. FP0, FP1, FP2, FP3의 앞선 작업은 동일한 시간에 시작하여 같은 시간에 작업을 마쳤다. 하지만 5ms 동안의 유휴상태 이 후, 다시 작업을 할당 받은 작업에 대한 종료시간에 차이가 발생하였다. 이러한 현상으로 보아, 내부적으로 또 다른 작업 지연 요인이 발생했음을 추정할 수 있다.

6. 결 론

본 논문에서는 모바일 멀티코어 GPU의 효율을 확인하기 위해 모바일 GPU 코어의 작업 분산 형태와 코어의 오버헤드에 대하여 분석하였다. 본 논문에서는 Mali-400 MP GPU를 사용하여 시스템 사용자 인터페이스의 동작을 처리하는데 소요된 전체 시간을 측정하는 실험을 수행하였다. 실험 분석 결과 시스템 사용자 인터페이스 처리에 필요한 전체 소요시간이 코어가 증가함에 따라 줄어들지 않고, 오히려 더 많은 시간이 소요되는 부분이 발생하는 모습을 보였다. 이러한 원인 분석을 위해 사용자 인터페이스를 조작하였을 때, 이를 처리하기 위한 멀티코어 GPU의 동작을 분석한 결과, CPU와 GPU사이의 통신으로 인한 오버헤드가 발생하였음을 확인할 수 있었고, 그 외에 내부적인 작업으로 인한 추가적인 지연이 발생하였음을 발견 할 수 있었다.

본 연구에서의 모바일 멀티코어 GPU의 프로파일링 환경 구성과 그래픽 처리에서 발생하는 비효율적인 GPU 동작에 대한 분석은 모바일 멀티코어 GPU연구의 기초로써, 앞으로의 모바일 멀티코어 GPU의 연구에 있어 참고가 될 수 있을 것으로 예상한다.

향후 멀티코어 GPU에서 발생하는 통신 오버헤드와 내부 작업으로 인한 추가적인 지연의 정확한 요인을 조사하는 실험을 하고, 해당 지연을 제거하고 멀티코어의 효율적인 동작이 가능하도록 연구를 진행 할 예정이다.

References

[1] Sunpyo Hong, Hyesoon Kim, “An Integrated GPU Power and Performance Model.”, In ACM SIGARCH Computer Architecture News, 2010.
DOI: http://dx.doi.org/10.1145/1816038.1815998

[2] NVIDIA, “NVIDIA CUDA Programming Guide”, 1, 2.2, 7, 2011.

[3] J.-H. Kim, J.-S. Kim “Implementation of Efficient Power Method on CUDA GPU.” Journal of The Korea Society of Computer and Information, Vol. 16, No. 2, pp. 9-16, February 2011.
DOI: http://dx.doi.org/10.9708/jksci.2011.16.2.009

[4] KATO, S., LAKSHMANAN, K., KUMAR, A., KELKAR, M., ISHIKAWA, Y., AND RAJKUMAR, R., “RGEM: A responsive GPGPU execution model for runtime engines.”, In Proc. of IEEE Real-Time Systems Symposium, pp. 57-66, 2011.
DOI: http://dx.doi.org/10.1109/RTSS.2011.13

[5] KATO, S., LAKSHMANAN, K., RAJKUMAR, R., ISHIKAWA, Y., “TimeGraph:GPU scheduling for real-time multi-tasking environments.”, In Proc. of USENIX Annual Technical Conference, 2011.

[6] M. Kim, W. Choi, “Range Query Method of R-tree for Efficient Parallel Processing on GPU”, Journal of KIISE : Computing Practices and Letters, vol.18, no.5, 409-413, May. 2012.

[7] H. choi, H. Jeon, C. Kim, “Quantitative Analysys of the

Negative Factors on the GPU Performance", Journal of KIISE : Computing Practices and Letters, vol.18, no.4, pp.257-350, Apr. 2012.
[8] Khronos Group, "The OpenCL Specification", Version 1.0, 7, 2009.
[9] Mian Dong, Lin Zhong, and Zhigang Deng, "Performance and Power Consumption Characterization of 3D Mobile Games", In IEEE Computer Society, 2011.
DOI: http://dx.doi.org/10.1109/MC.2012.190
[10] NVIDIA, CUDA Toolkit,
Available:https://developer.nvidia.com/cuda-downloads
[11] TechPowerUp, GPU-Z,
http://www.techpowerup.com/gpuz
[12] Aurora Softworks, Quadrant Benchmark,
http://www.aurorasoftworks.com/
[13] GFXBench, GLBenchmark,
http://www.glbenchmark.com/
[14] Akenine-Moller, T. and Strom, J., "Graphics processing units for handhelds", Proceedings of the IEEE 96(5), 779 - 789, 2008
DOI: http://dx.doi.org/10.1109/JPROC.2008.917719
[15] ARM, DS-5 Streamline, www.arm.com

임 효 정(Hyojeong Lim) **[정회원]**

- 2009년 2월 : 광운대학교 멀티미디어학과 (학사)
- 2013년 2월 : 충남대학교 컴퓨터공학과 (공학석사)
- 2013년 3월 ~ 현재 : (주)Contela 입사

<관심분야>
컴퓨터공학, 모바일 저전력 컴퓨팅

한 동 건(Donggeon Han) **[정회원]**

- 2013년 8월 : 충남대학교 컴퓨터공학과 (학사)
- 2013년 9월 ~ 현재 : 충남대학교 컴퓨터공학과 (석사과정)

<관심분야>
컴퓨터공학, 임베디드 소프트웨어, 모바일 저전력 컴퓨팅, 상황인지 컴퓨팅

김 형 신(Hyungshin Kim) **[정회원]**

- 1990년 12월 : Univ. of Surrey, 위성통신공학 (석사)
- 2003년 2월 : 한국과학기술원 전산학과 (박사)
- 2003년 4월 ~ 2004년 2월 : Carnegie Mellon Univ. 박사후 연구원
- 2004년 2월 ~ 현재 : 충남대학교 교수

<관심분야>
내장형 시스템, 시스템 소프트웨어, 우주용 컴퓨터