ㅇㅇㅇ

[소프트웨어 종합설계]

Runtime Estimation for Multi-Core Processor

Case study of Exynos 4412

ㅇㅇㅇ

조장 ㅇㅇㅇ

조원 ㅇㅇㅇ　ㅇㅇㅇ

담당 교수 ㅇㅇㅇ교수님

Mobile & Embedded Systems Laboratory

ㅇㅇㅇ

목차

ㅇㅇㅇ

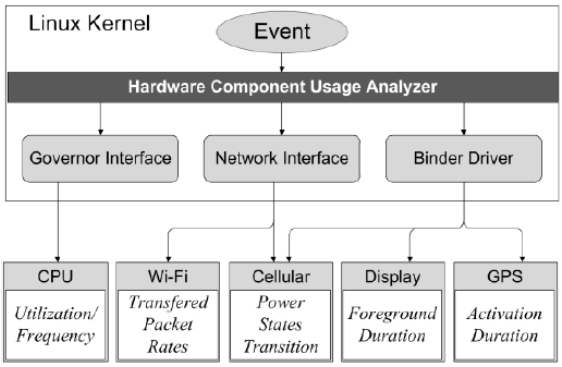
1. 목표

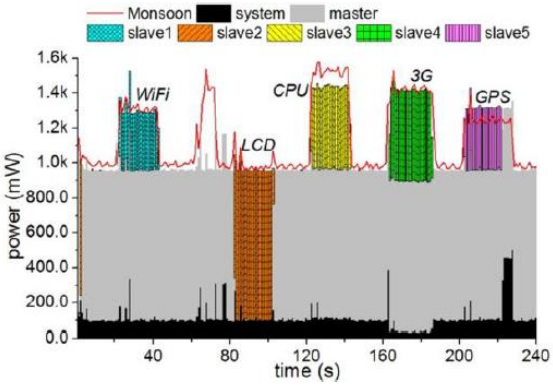
Multi-Core CPU와 GPU의 Power Modeling을 통한 실시간 전력 측정

2. 이전 연구와 연구의 필요성

2.1 AppScope

AppScope는 커널의 동작을 모니터링해서 안드로이드 스마트폰의 사용 전력을 측정하는 어플리케이션이다.





2.2 AppScope의 한계점

AppScope는 Single-Core에 맞춰져 있고 GPU를 포함하지 않고 있다. 그래서 Multi-Core를 사용하는 기기 혹은 GPU를 사용하는 경우는 전력을 정확하게 측정할 수 없다.

2.3 연구의 필요성

최근의 기기는 대부분 Multi-Core를 탑재하고 있고 안드로이드의 경우 ICS버전 이후로는 기본 런쳐에서도 GPU를 사용하는 정도로 GPU의 사용이 증가하고 있다. 그래서 이 두 가지를 AppScope에 적용시켜야 할 필요가 있다.

3. 연구 내용

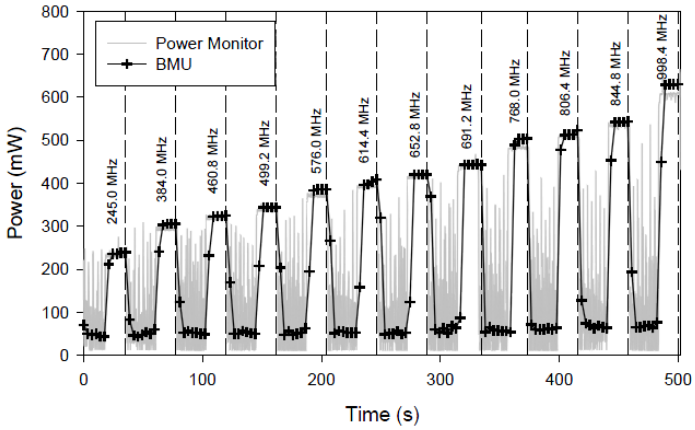
3.1 Target Devices

운영체제	안드로이드 OS v4.0.4 (ICS)
Chipset	ARMv7 Exynos 4412 Quad
CPU	Quad-core 1.4GHz Coretex-A9
GPU	Mali-400MP

3.2 Multi-Core CPU

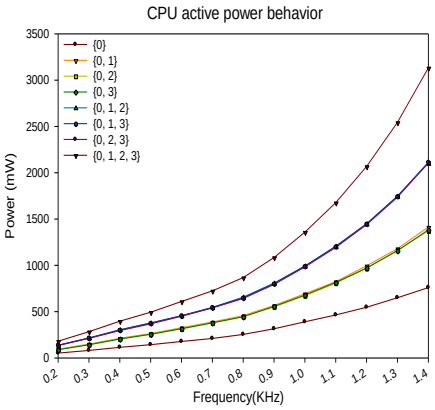
3.2.1 Utilization 기반 모델링

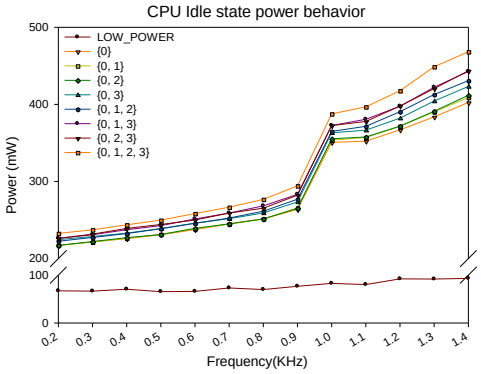
$$P^{CPU} = \beta_{freq}^{CPU} \times u + \beta_{freq}^{idle}$$
u: utilization, $0 \le u \le 100$
$freq$: frequency index, $freq = 0,1,2 \cdots, n$

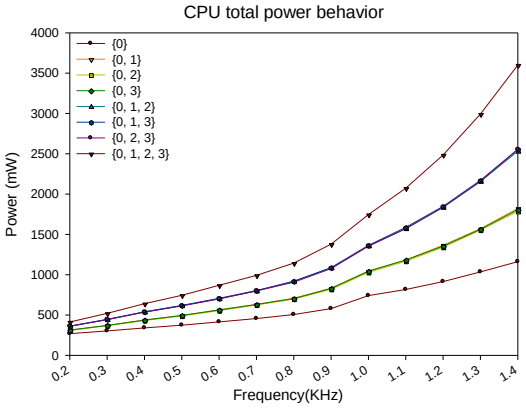


이전 모델링은 CPU가 기본적으로 사용하는 idle state의 소비 전력과 Utilization에 따라 소비되는 active state의 전력 값의 합으로 이루어진다. Utilization이 0%일 때 전력 값을 측정해 idle state의 coefficient값으로 설정하고 utilization이 100%일 때 active state의 coefficient 값으로 설정해서 위의 식에 적용한다. 각각의 frequency에 따라 이 coefficient값은 차이가 난다. 이 모델을 바탕으로 Multi-Core에 대한 전력을 측정해보았다.

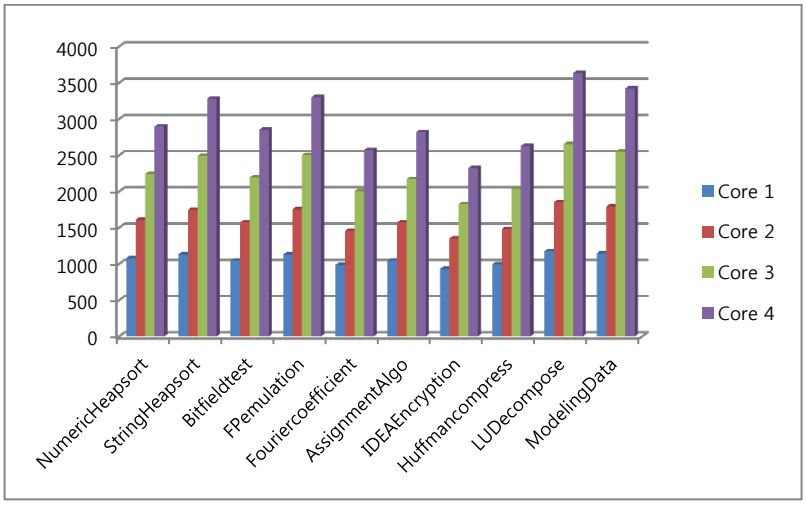
$$P_{CPU} = \sum_{j=0}^{frequencies} \left(\beta_{i,j}^{freq} * u_j^{freq} + \sum_{k=0}^{idle\ levels} \beta_{i,j,k}^{idle} * u_{j,k}^{idle} \right),$$
$$i = num\ of\ active\ core$$







이 실험을 통해서 전력은 켜진 코어의 번호와는 관련이 없고 켜진 개수만 관련이 있다는 것을 확인했고 Utilization 모델이 멀티코어에도 잘 적용되는 것을 확인할 수 있었다. 하지만 이것은 같은 종류의 연산의 경우만 적용이 가능하다는 점에서 문제가 있다.



Nbench의 각 연산의 소비 전력을 측정해봤다. 모두 Utilization은 100%로 동일하지만 소비 전력은 차이가 났고 이 차이는 코어가 1개 일때는 245.12mW 정도인데 코어가 4개일 때는 1308.27mW로 큰 차이를 보인다. 그러므로 Utilization 모델을 Multi-Core CPU에 적용시키는 것은 큰 오차를 발생시킬 우려가 있다.

3.2.2 HPC 기반 모델링

About HPC

HPC, Hardware Performance Counter는 하드웨어의 성능을 측정하기 위한 회로로, 지원하는 이벤트들의 발생횟수를 측정하는 방법으로 성능을 알 수 있게 해주는 기능을 한다. 본 연구에서 사용하는 HPC는 Galaxy S3 의 AP인 Exynos4412의 내부에 들어가있는 HPC를 이용할 것이다.

HPC approach

- PERF

리눅스 커널소스에 기본적으로 탑재되어 있는 Performance monitoring tool이다. 리눅스와 마찬가지로 오픈소스 소프트웨어로 ARM아키텍쳐를 위한 버전도 준비되어있다. 동작방식은, HPC 값을 기반으로 성능을 측정하며, hpc값에 접근할 수 없는 경우에도 커널에서 제공하는 다양한 정보를 통해 성능에 관련된 다양한 정보를 얻는다.

하지만, PERF를 통해서는 실시간으로 CPU의 동작정보를 알기 힘들고, 또한 단순한 HPC값을 가져오는 기능외에도, 다른 기능들이 들어가 있기 때문에 OVERHEAD가 크고, 실험에 사용하기 힘든 단점이 존재한다.

Direct approach

HPC는 PERF와 같은 툴을 통해서도 접근할 수 있지만, ARM에서는 HPC를 이용할 수 있도록 인라인 어셈블리 문법을 제공하여 직접 접근 또한 가능하다. 실험을 위해서 HPC값을 유저모드에서

읽어올 수 있도록 해주는 커널 모듈을 작성하였고, NDK를 이용하여 최종적으로는 유저 어플리케이션 수준에서 HPC값을 읽을 수 있도록 하였다.

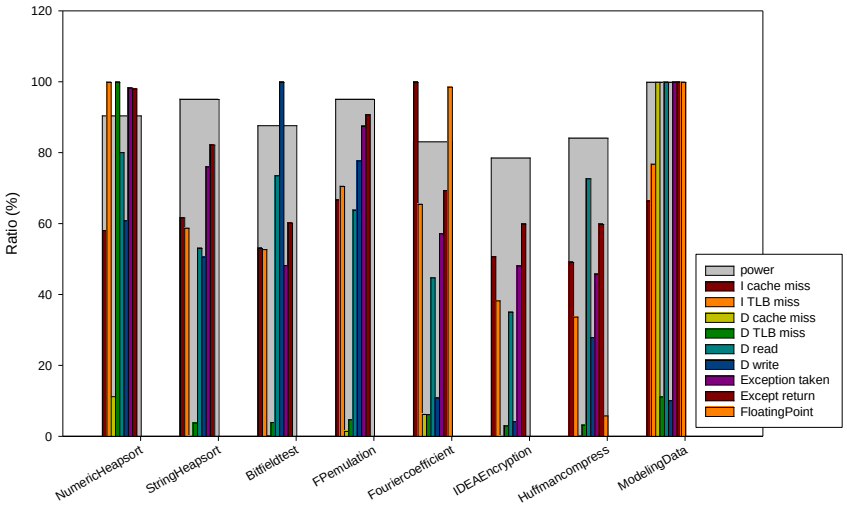
HPC Data

Exynos4412 내부의 HPC는 커널코드 상에서는 PMU(Performance Monitoring Unit)으로 불리며, Exynos4412의 베이스인 ARM Cortex A9 아키텍쳐의 PMU에서 지원하는 총 58가지 이벤트를 측정할 수 있게 설계되어 있다. 이 이벤트들은 Instruction cache miss, Data read/write, Floating point unit operation 등 cpu에서 수행되는 대부분의 기능들을 측정할 수 있도록 설계되어있다.

하지만, HPC는 1개의 cycle register와 6개의 counter register로 이루어져, 동시에 측정할 수 있는 이벤트의 수가 최대 6개로 제한되어 있다.

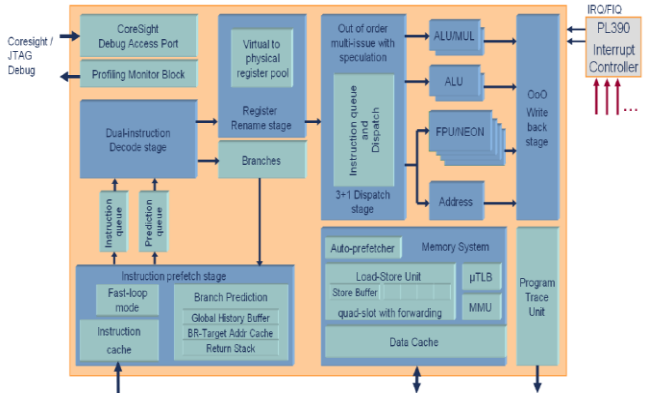
HPC Data Analysis

동시에 측정 가능한 이벤트의 수가 6개로 제한됨에 따라서, Power Modeling에 사용할 변수로서의 이벤트의 종류를 결정해야 할 필요성이 존재한다. 따라서 먼저 58가지의 이벤트중 cpu의 동작 및 반복횟수에 관련된 이벤트들을 선정하여 벤치마크를 수행하는 동안 hpc값의 결과를 얻었다.



위 그래프는 해당 실험의 결과이다. 위 그래프에서도 알 수 있듯이 소비전력은, 적은 수의 이벤트에 의해 지배적으로 영향 받지 않으며, 좀 더 많은 수의 변수에 의해 결정될 것이라 예측된다.

따라서 먼저 단순한 회귀분석을 통한 Power model을 만드는 것보다, 구조적인 분석을 통한 접근이 필요하다고 생각된다. 아래의 블록 다이어그램은 ARM Cortex-A9의 CPU구조이다.

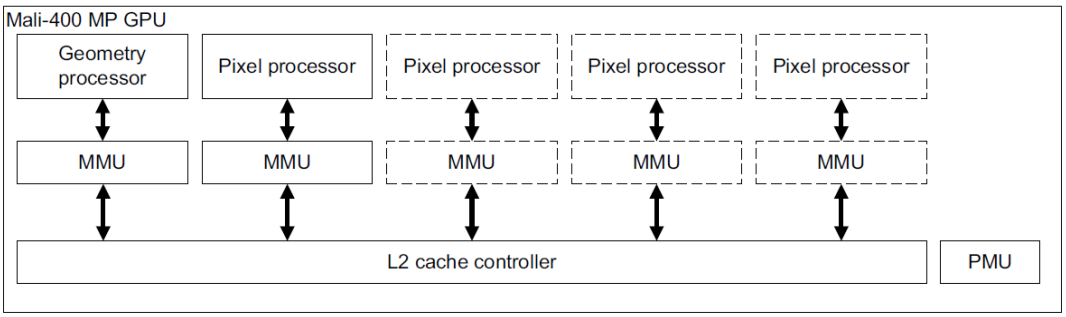


3.3 GPU

3.3.1 Mali-400MP

Mali-400MP는 Exynos 4412에 채택된 ARM사의 GPU이다. ARM의 Mali GPU는 이와 같이 자사의 CPU와 함께 사용되는 경우가 많으며, 모바일 기기나 임베디드 시스템에서 타일 기반의 렌더링 처리를 담당한다. 이 Mali-400MP는 Exynos 4412의 Cortex-A9 CPU와 마찬가지로 코어가 여러 개인 multi-core 형태이며, CPU는 모든 코어가 동일한 homogeneous multi-core인데 비해 GPU는 1개의 Geometry Processor(Vertex Processor)와 4개의 Pixel Processor(Fragment Processor)로 이루어져 있다. 이 외에도MMU(Memory Management Units)와, PMU(Power Management Unit), 그리고 L2 cache도 함께 Mali-400MP를 구성하고 있다.

주요 구성 요소들에 대해서 살펴보면, Geometry Processor는 vertex 생성 및 처리를 담당하며 primitive list, polygon list, packed vertex data를 생성하여 Pixel Processor에게 넘겨주는 역할을 한다. Pixel Processor들은 fragment 처리를 담당하며 Geometry Processor에서 생성한 list들과 여러 data structure들을 활용하여 이미지를 생성하는 작업을 담당한다. 마지막으로 L2 cache는 일반적인 cache 기능과 마찬가지로 memory bandwidth usage와 power consumption을 줄이는 역할을 하며 특히 memory에 접근하는 cost를 줄이기 위한 것이 주목적이다.

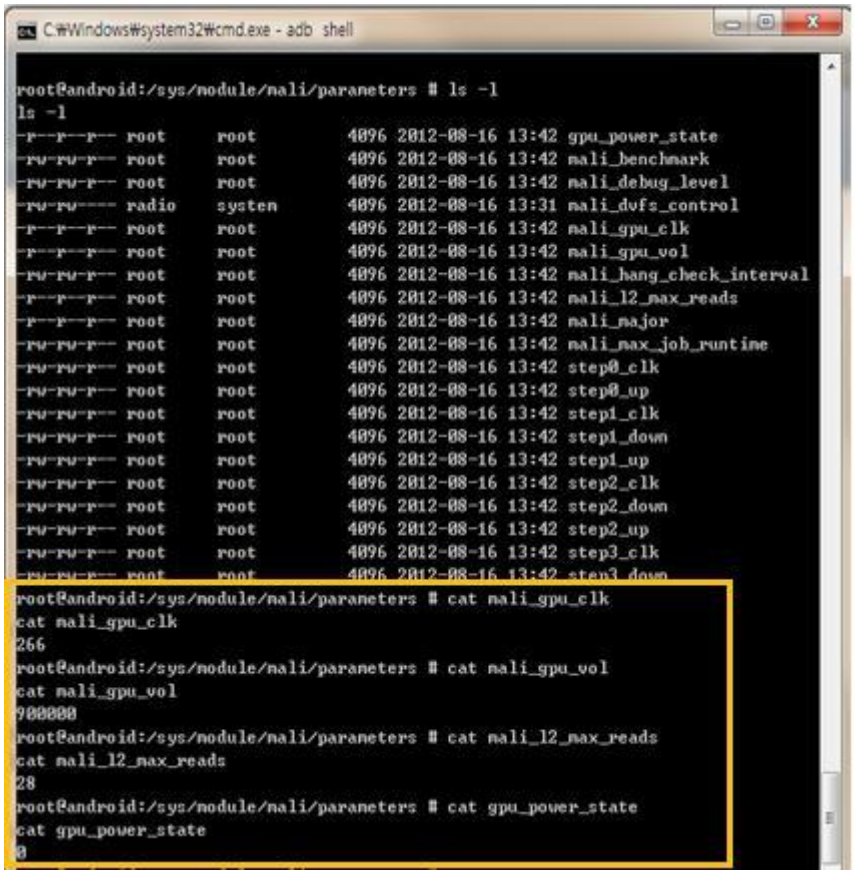


3.3.2 Mali GPU Utilization

GPU의 전력 소비를 모델링하기 위해서는 기본적으로 GPU만의 소비 전력은 물론 GPU가 언제, 얼마나 쓰이는지에 대한 정보들이 반드시 필요하다. 이 정보는 크게 GPU의 Utilization에 기반하는 데이터와 앞서 CPU 부분에서 설명한 HPC(Hardware Performance Counter)에 기반 하는 데이터로 나뉘는데, HPC 기반의 데이터로 모델링하는 것이 GPU에서도 보다 정확한 결과를 가져다 줄 것으로 예상되나 대상 기기에서는 GPU 관련 HPC를 사용하는 것이 실질적으로 불가능하여(HPC로 하여금 GPU관련 정보를 수집하도록 커널을 수정하였을 경우 기기가 부팅되지 않는 문제 발생) Utilization 기반으로 모델링을 하기로 결정하였다. CPU의 경우는 이미 Utilization 기반의 모델링이 존재하여 이를 HPC 기반으로 개선하는데 의미를 두었다면, GPU는 이전에 스마트폰 GPU에 대한 제대로 된 모델링이 존재하지 않았으므로 Utilization 기반의 모델링이라도 그 연구에 의의가 있다 할 수 있겠다.

Mali GPU를 사용하는 안드로이드 스마트폰 상에는(/sys/module/mali/parameters/) 아래 이미지와 같은 파일들이 존재한다. 이는 커널 옵션 중 'mali profiling'에 의해 생성된 것으로 실제 커널 코드 상에도 이와 관련한 소스들이 존재한다. 이 parameter들은 Mali GPU와 관련된 여러 데이터들을 각각 가지고 있다. step0부터 step3로 시작하는 파일들은 Mali GPU의 DVFS(Dynamic Voltage and Frequency Scaling)에 관련된 것인데, Mali-400MP의 경우 총 4개의 step이 존재한다. 각 step은 서로 다른 frequecny 및 voltage를 가지는데, Exynos 4412의 경우 이 4개의 step을 활용하여 각 상황마다 가장 적은 전력 소모로 작업을 처리할 수 있는 최적의 step을 선택한다.

참고로 각 step의 frequency를 살펴보자면, step0 : 160Mhz, step1 : 260Mhz, step2 : 350Mhz, step3 : 440Mhz이고, 기본 대기화면인 런쳐나 일반적으로 사용하는 어플리케이션의 경우 GPU를 사용하긴 하지만 GPU가 처리해야할 양이 적으므로 상대적으로 낮은 클럭에서 동작한다(step0, 1). 한편, GPU의 성능을 한계까지 시험하는 GPU benchmark 어플리케이션(GPU-T, NenaMark1, NenaMark2, BaseMark GUI Free, BaseMark Taiji Edition 등)이나 3D 그래픽이 많이 포함된 고사양의 게임 어플리케이션을 실행하는 경우에는 높은 동작 클럭(step2, step3)를 사용하는 것을 mali parameter 중 'mali_gpu_clk'를 통해 확인할 수 있었다. step으로 시작하는 것을 제외한 mali paramter들은 모두 실시간으로 바뀌는 데이터들이므로 필요할 때 유효한 데이터로 사용할 수 있다.



이전 연구(Appscope)의 경우 모든 하드웨어 컴포넌트들의 모델링이 Utilization 기반으로 이루어졌고, 사용 시간 등 다양한 정보를 바탕으로 각각의 Utilization을 직접 구해 모델링에 사용하였다. 따라서 이번에 Utilization 기반의 GPU 모델링을 하게 되면서 직접 Utilization을 마찬가지로 직접 계산하려 했으나, 커널 소스 중 Mali 관련한 경로들을 모두 확인한 끝에 Utilization을 계산하는 함수를 찾아 이를 활용하게 되었다. Mali GPU의 Utilization을 계산하는 이 함수는 말 그대로 'calculate_gpu_utilization'이라는 이름을 가지고 있으며 drivers/media/video/samsung/mali/common/mali_kernel_utilization.c 라는 경로에 위치해 있다. calculate_gpu_utilization 함수는 커널 상에 정해진 기준 시간(1000ms = 1초, 수정 가능)마다 GPU의 현재 Utilization을 계산해 전용 핸들러인 mali_gpu_utilization_handler에 계산된 Utilization 값을 넘겨준다. 참고로 이 Utilization 값은 0 이상 256 이하의 범위를 가진다.

```
93      utilization = work_normalized / period_normalized;
94
95      accumulated_work_time = 0;
96      period_start_time = time_now; /* starting a new period */
97
98      _mali_osk_lock_signal(time_data_lock, _MALI_OSK_LOCKMODE_RW);
99
100     _mali_osk_timer_add(utilization_timer, _mali_osk_time_mstoticks(MALI_GPU_UTILIZATION_TIMEOUT));
101
102     mali_gpu_utilization_handler(utilization);
103 }
```

그러나 이는 커널 내부에서만 계산하고 활용하는 데이터일 뿐 mali parameter처럼 직접적으로 쉽게 확인하는 것은 불가능하다. 따라서 Kprobes라는 커널 메소드를 사용하여 리눅스 커널 모듈을 작성하였다. 모듈로 작성하였기 때문에 따로 커널 컴파일이 필요하지 않다는 장점이 있고, 또 Kprobes 중에서 jprobe를 사용하면 hooking하는 함수가 인자로 받는 parameter 값을 그대로 읽어 올 수 있으므로 용도에도 가장 적합하다. 작성한 test.ko라는 모듈은

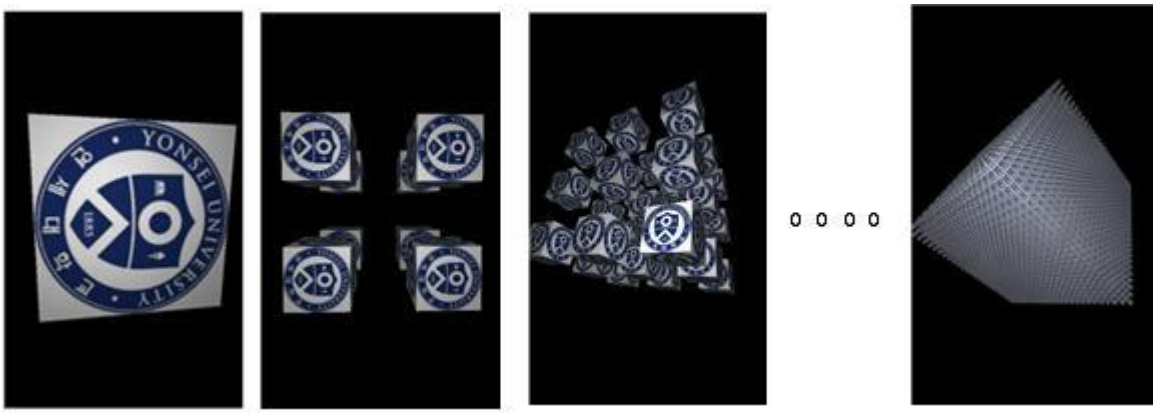
mali_gpu_utilization_handler를 hooking 해서 그 parameter로 전달되는 Utilization 값을 읽어와 이를 커널 메시지로 출력한다. 다음은 해당 출력 부분만 따로 출력한 결과이다.

```
<4>[  610.805205] c0 [GPU] Utilization = 1
<4>[  611.805199] c0 [GPU] Utilization = 0
<4>[  618.685054] c0 [GPU] Utilization = 62
<6>[  618.710106] c0 [TSP] DVFS Off!
<4>[  619.685106] c0 [GPU] Utilization = 75
<4>[  787.420063] c0 [GPU] Utilization = 255
<4>[  788.420061] c0 [GPU] Utilization = 255
<4>[  789.420057] c0 [GPU] Utilization = 255
<4>[  790.420049] c0 [GPU] Utilization = 249
<4>[  791.420061] c0 [GPU] Utilization = 255
<4>[  792.420049] c0 [GPU] Utilization = 243
<4>[  793.420056] c0 [GPU] Utilization = 255
<4>[  794.420060] c0 [GPU] Utilization = 255
```

3.3.3 Experiment

이제 Utilization 기반의 모델링에 필수적으로 필요한 데이터들(단위 시간마다 계산되는 Utilization 값, 동작 클럭 및 전압 값)을 수집하는 방법을 마련하였으므로 실제로 유용한 데이터를 수집할 차례이다. 앞서 Mali-400MP의 동작 클럭 및 Utilization 값의 변화 등을 확인하기 위해 benchmark를 비롯해서 여러 가지 어플리케이션들로 테스트를 해보았지만, 이는 GPU에 맡기는 연산의 양을 직접 조절할 수 없고 이미 만들어진 대로만 사용해야 하기 때문에 유용한 정보들을 뽑아내는 데는 무리가 있다. 수집한 정보들로 전력 소비에 관한 공식을 유도해내기 위해서는 매우 적은 연산량부터 점차적으로 그 크기를 늘려가면서 테스트하는 과정이 필수적이기 때문이다. 따라서 본 연구를 위해 다음과 같은 어플리케이션을 새로 만들게 되었다.

어플리케이션 이름은 GPUtest이다. GPUtest는 OpenGL ES 2.0을 기반으로 만들어졌는데, n^3의 큐브(정육면체)를 생성하여 특정 속도로 일정하게 회전시키는 작업을 수행한다. 처음 어플리케이션을 실행하면 간단한 타이틀 화면이 나오는데, 여기에 전체 큐브 수를 결정할 한 변 당 큐브 수, 회전 주기(ms)를 입력할 수 있다. 여기서 설정한 값들에 따라 큐브의 수 및 큐브의 회전 속도가 바뀌므로 원하는 데이터를 뽑아내기 위한 실험을 편리하고 반복적으로 수행할 수 있다.



4. 현재 진행 상황

4.1 Multi-Core CPU

ARM Cortex-A9의 아키텍쳐와 HPC의 관계 분석 및 실험 진행 중

HPC 기반의 모델링 구상 중

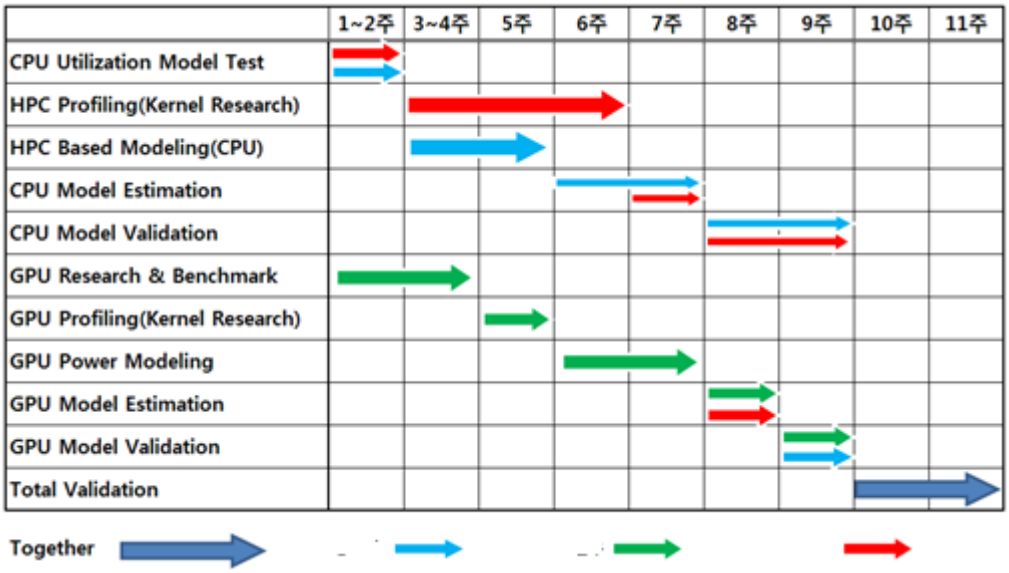
4.2 GPU

어플리케이션을 바탕으로한 다양한 실험 진행 중

보다 정확한 실험 환경을 위해 고려해야하는 부분을 고민

수집한 실험 데이터들을 정리하고 모델링

5. 일정 및 역할 배분



현재 6주차까지 진행된 상태이다.

지금까지 주로 진행했던 파트는 다음과 같다.

O O O / O O O Multi-Core CPU 담당

O O O Utilization 기반 모델을 멀티코어에 적용 및 문제점 도출

O O O HPC 값을 직접 읽어오는 NDK application 작성 및 HPC 값 측정

O O O GPU 담당

O O O Mali-400MP에 대한 정보 수집 및 실험 어플리케이션 작성