KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.5 pp.153~160 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.5.153

An Efficient Load Balancing Technique in a Multicore Mobile System

Cho Jungseok† · Cho Doosan††

ABSTRACT

The effectiveness of multicores depends on how well a scheduler can assign tasks onto the cores efficiently. In a heterogeneous multicore platform, the execution time of an application depends on which core it executes on. That is to say, the effectiveness of task assignment is one of the important components for a multicore systems' performance.

This work proposes a load scheduling technique that analyzes execution time of each task by profiling. The profiling result provides a basic information to predict which task-to-core mapping is likely to provide the best performance. By using such information, the proposed technique is about 26% performance gain.

Keywords : Smartphone, Multicore, Task Allocation, Code Analysis, Mobile Platform

멀티코어 모바일 시스템에서 효과적인 부하 균등화 기법

조 중 석† · 조 두 산††

요 약

멀티코어 시스템의 효율은 스케줄러가 태스크 할당을 코어들에게 얼마나 효율적으로 분배하느냐에 달려있다. 이기종 멀티코어 플랫폼에서 애플리케이션의 실행시간은 어느 코어에서 실행되느냐에 따라 결정된다. 즉, 태스크 할당의 효율이 멀티 코어 시스템의 성능을 결정하는 중요한 요소 중의 하나이다. 본 연구에서는 프로파일링을 통하여 각 태스크의 실행시간을 분석하고 이를 이용하는 부하 균등화 기법을 제안하고 있다. 프로파일링 결과는 최상의 성능을 제공할 수 있는 태스크 할당을 예측하는 기본적인 정보를 제공한다. 이러한 정보를 이용하여 제안하는 기법을 통해 약 26%의 성능이득을 가질 수 있다.

키워드 : 스마트폰, 멀티코어, 작업 할당, 코드 분석, 모바일 플랫폼

1. 서 론

최근 모바일 기기의 상용화로 인해 이용자의 수가 매해 증가하고 있다. 이러한 증가 추세를 보면, 2018년에는 세계 모바일 인구가 40억 명을 돌파할 것으로 예측된다[1]. 이러한 추세와 더불어 최근 모바일 애플리케이션의 시장규모 확대에 따라 HD 비디오 재생, 스트리밍 AV서비스, 멀티태스킹, 웹 브라우징, 3D 게임 등 싱글코어 프로세서로는 감당하기 어려운 PC와 같은 수준의 높은 성능을 요구하고 있다[2-3]. 하지만 Fig. 1을 보면 급격한 성능의 발전을 보이는 현재의 휴대용 단말기라도 배터리 수명 증가 속도가 따라가지 못하는 현상이 발생하고 있다[4]. 결국 고성능일지라도 프로세서의 전력 소모를 감소시키거나 증가를 최소한으로 억제해야 한다.

애플리케이션의 비효율적인 전력 사용을 막기 위해서는 우선 애플리케이션의 실행시간, 동작주기와 같은 특성에 따른 전력 소비 분석이 선행되어야 한다. 즉, 유사한 서비스를 제공하는 애플리케이션들 간에도 실행 특성에 따른 전력 소비의 차이를 보이는 경우가 있기 때문에, 전력 소비가 더 적은 애플리케이션의 실행 특성을 분석하여 이를 다른 애플리케이션에 적용함으로써 전력 소비를 절감하는 효과를 가져올 수 있다.

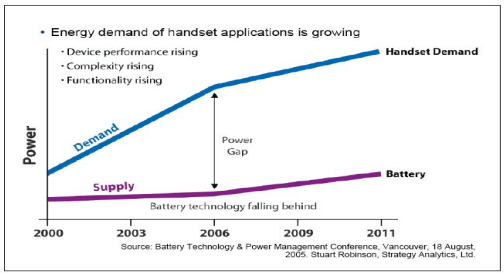


Fig. 1. Improvement Status of Energy Demand and Battery Supply(Courtesy of the Battery Technology & Power Management Conference, Vancouver, August 18, 2005; Stuart Robinson, Strategy Analytics Ltd.)

※ 이 논문은 2010년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업 (No. 2010-0024529), 2013년도 정부(교육부)의 재원으로 한국과학창의재단 (대학생창의융합형 연구과제 지원사업)의 지원을 받아 수행된 연구임.
※ 이 논문은 2014년도 한국정보처리학회 추계학술발표대회에서 '어플리케이션 실행 특성 분석을 통한 모바일 시스템 성능 최적화 연구'의 제목으로 발표된 논문을 확장한 것임.
† 비 회 원 : 순천대학교 전기전자공학부 박사과정
†† 정 회 원 : 순천대학교 전기전자공학부 부교수
Manuscript Received : December 31, 2014
First Revision : March 4, 2015
Second Revision : March 19, 2015
Accepted : March 27, 2015
* Corresponding Author : Cho Doosan(dscho@sunchon.ac.kr)

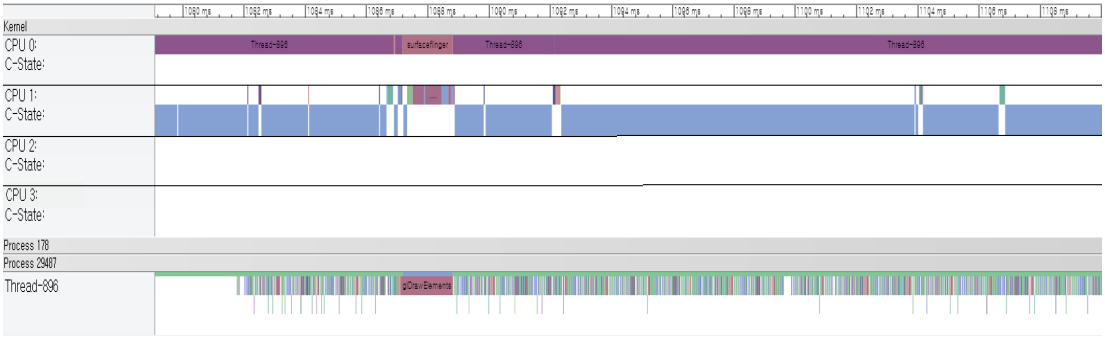


Fig. 2. CPU Utilization in a Mobile Platform

또한, 특정 기능을 구현한 앱의 성능을 개선하는 방법이 단순히 하드웨어의 성능 향상으로 이뤄지기보다는 소프트웨어의 최적화를 통해 동일 기기에서 에너지 소모 감소와 처리 속도 증가 및 성능 향상을 할 수 있도록 개선하는 것이 필요하다. Fig. 2를 보면 애플리케이션 실행 시에 쓰레드(Thread)의 대부분을 CPU 0에서 처리를 하고 사운드나 시스템 UI 및 기타 하드웨어 이벤트 발생 시 CPU 1에서 처리하고 Idle 상태가 되는 것을 알 수 있다. 4개의 CPU를 가지고 있어도 대부분의 쓰레드 처리는 하나의 CPU에서 처리가 된다. 이는 성능을 감소시키고, 에너지 효율 측면에서도 비효율적이다. 따라서 모바일 디바이스의 성능 개선 및 애플리케이션의 효율적인 자원 분배와 최적화를 위해서 애플리케이션 실행 특성의 분석이 필요하다.

애플리케이션의 실행 특성을 분석하여 모바일 시스템의 최적화 기법을 개발하는 데 본 연구의 목적이 있다. 이를 위해 애플리케이션의 실행 특성을 분석하고 그 결과를 바탕으로 효율적인 자원 할당을 위한 알고리즘을 제시하여 그에 따른 시스템 성능 향상을 제시한다. 2절에서는 애플리케이션 실행 분석을 위한 방법을 제시한다. 3절에서는 프로파일된 데이터를 통하여 최적화를 위한 알고리즘을 제시한다. 4절에서는 제시하는 알고리즘의 성능 평가를 하였다. 마지막으로 5절과 6절에서는 관련 연구과 결론을 기술하였다.

2. 애플리케이션 실행 분석

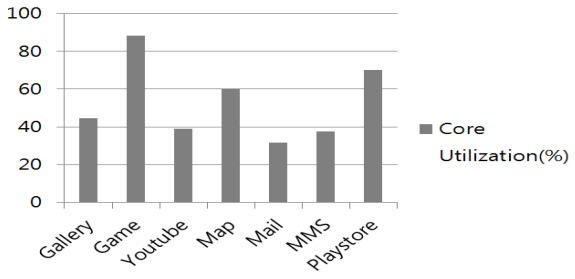


Fig. 3. CPU Utilization of Applications

애플리케이션의 실행 분석 기준은 CPU 사용률을 기준으로 한다. ADB(Android Debug Bridge)를 통하여 디바이스에서 실행되는 애플리케이션을 분석한다. 애플리케이션은 사용 빈도가 높은 앨범, 게임, 유튜브, 지도, 메일, MMS, Playstore로 하였다. 각각 실행시간은 10분으로 하여 각 애플리케이션의 CPU 사용률을 Fig. 3에 나타내었다.

각각의 애플리케이션의 각 CPU 사용률을 측정하기 위해 Systrace를 이용하여 분석한다. Systrace 분석은 애플리케이션 중 CPU 사용률이 가장 높은 Game을 기준으로 분석한다. Systrace 분석 결과를 Fig. 4와 Fig. 5에 나타내었다. 실험의 결과를 바탕으로 4개의 코어를 사용하는 디바이스에서 다양한 애플리케이션 실행 시 사용되는 코어는 보통 2개를 넘지 않음을 알 수 있다. 각 코어별 사용률에 영향을 미치는 요소들을 분석하기 위해 동적 코드 분석 도구를 사용하여 실행 가능한 이진 프로그램에 대한 동적 분석을 실행한다. 동적 코드 분석 도구는 동적 컴파일을 통한 동적 코드 변환 기법을 사용한다[5]. 프로그램 실행을 통하여 하드웨어 이벤트를 측정하여 실제적으로 프로그램이 가지는 쓰레드와 함수, 명령어들이 CPU 사용률에 미치는 영향들을 분석한다. Fig. 6을 보면 각 함수의 CPU time을 알 수가 있다.

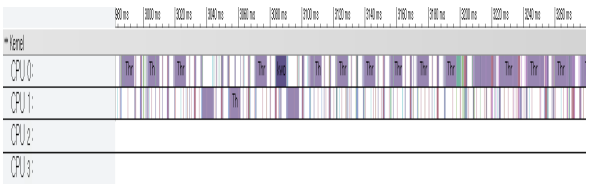


Fig. 4. Core Utilization Result of a Game Application

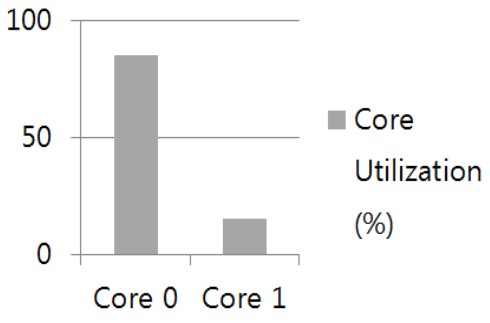


Fig. 5. Core Utilization(%)

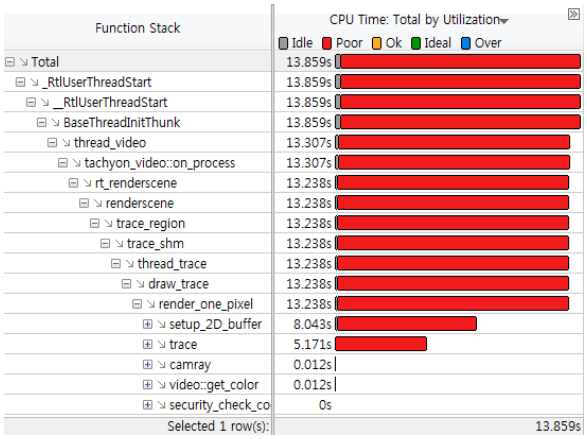


Fig. 6. CPU Utilization Analysis at Function Level

3. 알고리즘

3.1 The Proposed Approach

애플리케이션 프로파일의 결과를 바탕으로, 효과적인 CPU 사용을 위한 알고리즘을 제시한다. 자원 할당, 즉 병렬 처리에는 크게 명령어 단위 병렬 처리와 태스크 단위 병렬 처리가 있다. 명령어 수준 병렬 처리는 데이터 의존성이 존재하지 않는 여러 개의 명령어들을 동시에 수행하는 병렬 처리를 말하며, 태스크 단위 병렬 처리는 기능에 따라 분할한 작은 태스크 단위로 병렬 처리하는 것을 말한다. 병렬 처리란 다수의 프로세서들이 여러 개의 프로그램들 혹은 한 프로그램의 분할된 부분들을 분담하여 동시에 처리하는 기술을 말하는데, 이때 분할된 부분을 그레인이라고 한다. 그레인의 크기가 매우 작은 명령어 수준 병렬 처리 같은 경우 많은 병렬성을 얻을 수 있지만 애플리케이션 단위에서 동기화와 스케줄링에 과부하가 발생하게 된다. 그렇기 때문에 태스크 단위 병렬 처리를 목적으로 애플리케이션의 자원 할당을 실행한다. 본래 태스크는 다양한 범위로 정의될 수 있다. 본 연구에서는 애플리케이션의 함수 단위로 태스크를 정의하였다. 애플리케이션을 구성하는 함수들을 대상으로 복수의 프로세서 코어를 이에 할당하는 스케줄링 기법으로 제안한다. 제안하는 스케줄링 기법은 복수의 프로세서 코어의 실행시간이 최소가 되도록 애플리케이션의 함수들에 할당하는 것을 목적으로 한다.

nvidia white paper[6] 중 워크로드 기반 동적 CPU 코어 가용의 내용을 보면, 애플리케이션의 실행에 있어서 CPU 가버너와 CPU 매니지먼트 로직에 의해 애플리케이션이 요구하는 성능에 따라 코어의 사용을 조절한다. 이를 바탕으로 본 연구에서는 코어의 사용을 애플리케이션 단위가 아닌 애플리케이션 프로파일에 의한 애플리케이션이 동작하는 각 함수들의 실행시간 및 사용이 되는 각 코어들과 사용되지 않는 코어들에 대한 데이터를 바탕으로 알고리즘을 설계한다.

제안하는 시스템 프레임워크는 Juraj Feljan, Jan Carlson[7]의 시스템 프레임워크를 바탕으로 설계하였다. [7]은 멀티코어 임베디드 시스템에서의 최적화 태스크 할당 방법으로, 소정한계 이하의 데드라인 미스의 수를 유지하면서, 작업 체인 종단 간 응답 시간을 최소화하는 것으로 추정에 의한 지연 정보들에 기초하여 지연이 적은 추정값을 선택하여 최적화하는 방법이다. 이를 바탕으로 시스템 프레임워크를 설계하였으며 Fig. 7과 같다.

애플리케이션이 선택되면, 하드웨어, 즉 디바이스에서 애플리케이션이 실행되고, 실행이 종료되면 애플리케이션이 실행하는 동안의 태스크들을 수집하여 분석하고, 각 태스크들이 차지하는 자원의 소모에 따라서 분배된다. 모든 태스크들의 분배가 되었을 시 작업을 종료한다.

Fig. 7에서 Target Application은 실행이 되는 애플리케이션이며, HW Specification은 하드웨어가 가지는 코어의 수로 이는 애플리케이션이 실행하면서 종료되는 시점에 측정이 된다. 애플리케이션이 실행, 분석, 할당, 검사를 순차적으로 진행하여 모든 태스크들이 분할이 될 시 프로세스가 종료된다.

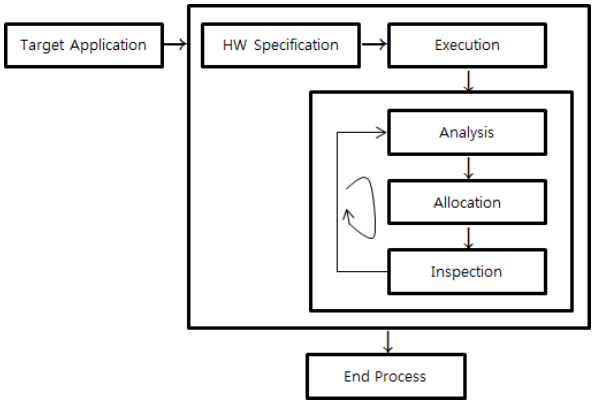


Fig. 7. Task Allocation

3.2 Task Allocation

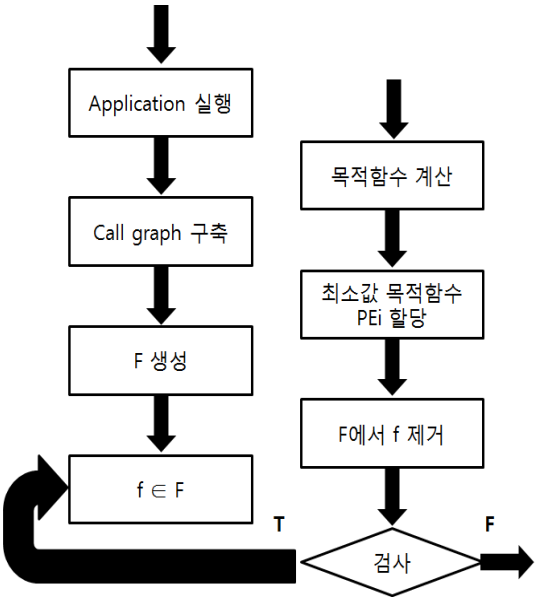
시스템 프레임워크를 바탕으로 효과적인 CPU 사용을 위한 알고리즘을 제시한다. 코어의 균일한 할당을 위하여 코어의 할당은 실행 프로그램의 함수 실행 시 CPU 점유율을 기준으로 한다.

$$PE = 1, 2, 3, \ldots, n$$
$$F = 1, 2, 3, \ldots k$$
$$Load(PE_i) = normalized execution time(allocated(f, PE_i)) / speed(PR_i) \quad (1)$$

$$AveLoad = \frac{\sum_{i=1}^{n} Load(PE_i)}{n} \quad (2)$$

$$Objective\ function = Minimize\left(\sqrt{\frac{\sum_{i=1}^{n}(Load(PE_i) - AveLoad)^2}{n}}\right) \quad (3)$$

PE는 MPSoC 프로세서 코어 집합이며, F는 애플리케이션 함수 집합, Load(PEi)는 정규화 실행시간 함수의 부하를 나타내며 allocated(f,PEi)는 할당된 함수 집합 f ∈ F to PEi를 나타낸다. 이종 MPSoC에서, 부하는 해당 워크로드 PEi, i는 코어의 수를 뜻한다. 제안하는 기법의 목적은 복수의 CPU를 충분히 활용하여 시스템 성능이 최상이 되도록 하는 데 있다. 따라서 특정 코어에 태스크가 집중되지 않도록 작업 분배가 고르게 되도록 해야 한다. 동일한 성능의 코어로 구성된 동종 시스템과 다양한 성능의 코어로 구성된 이기종 시스템 양쪽에서 이를 달성하기 위해서는 태스크의 작업량을 정규화하여 정의해야 한다. 본 연구에서는 애플리케이션의 작업량을 정규화하기 위하여 일반 PC에서 동일한 환경을 구성하고 함수단위로 실행시간을 반복 측정하여 그 평균값을 함수의 작업량으로 정의하였다. 제안하는 기법의 최종 목적은 코어별로 작업량을 균등하게 하여 시스템 성능을 최적으로 하는 데 있다. 이를 달성하기 위하여 목적 함수는 코어들에 할당된 작업량의 표준 편차를 최소화하도록 설계하였다. 알고리즘 동작은 Fig. 8과 같다.



[Algorithm]
0. 애플리케이션을 실행하고 함수 호출 트리를 구축
1. 할당하지 않은 함수 선택
2. PEi에 f를 할당하여 목적 함수를 계산
3. 최소 결과를 가지는 목적함수 f를 선택하여 PEi 할당
4. PEi에 f를 할당하고 F에서 f를 제거
5. 할당하지 않은 함수 검사
6. 할당 되었다면, 종료
7. 할당 되지 않았다면, 다시 1번으로

Fig. 8. Algorithm

제안하는 알고리즘은 안드로이드 OS에서 기존의 스케줄러와 연동하여 실행된다. 먼저 타깃 모바일 시스템에서 구동하는 한 개의 애플리케이션을 실행하고, 해당 애플리케이션의 프로파일 정보 파일을 미리 특정 폴더에 저장해놓고 스케줄러가 구동할 때 읽어 들인다. 해당 파일은 프로파일링 결과로 함수 호출 트리, 함수 리스트, 각 함수별 실행시간 등의 정보로 구성된다. 제안하는 스케줄링 기법은 함수 리스트와 정규화된 함수별 작업량을 참조하여 진행된다.

우선 함수리스트 F를 검사하여 할당되지 않은 f가 있다면 이를 선택한다. f를 모든 PE에 한 번씩 할당하여 목적 함수 값을 계산한다. 목적 함수는 각 프로세서 코어가 할당받는 작업량의 표준 편차로서 모든 프로세서 코어가 균등하게 평균에 가까운 작업량을 할당받도록 유도하는 역할을 한다. 최소의 값을 갖는 PE에 f를 할당한다. 이때 최소값을 갖는 코어가 여러 개일 경우 순차적으로 할당한다. 할당된 f를 함수 리스트 F에서 제거한다. F를 다시 검사하여 남 있는 함수가 있다면 알고리즘 라인 1부터 반복 진행하여 모든 함수를 할당하고 알고리즘을 종료한다.

4. 실험 결과

제시하는 알고리즘을 검증하기 위해, Microsoft Visual Studio 2012와 Intel Vtune Amplifier XE 2015 버전을 사용하였다. 모바일 디바이스 상의 애플리케이션의 디버깅 제한으로, PC에서 실험 환경을 구성하여 측정하였다.

알고리즘을 적용시키기 위해 영상처리 프로그램과 데이터 버퍼에 이미지를 그리는 프로그램과 사진 보기 프로그램, 사진 편집 프로그램, 시스템 UIAClient 프로그램, Sudoku 프로그램 총 6개의 프로그램에서 실행되는 함수들의 CPU 시간을 측정한다. Fig. 9는 영상처리 프로그램에 대한 각 함수들의 CPU 사용 시간을 나타낸다.

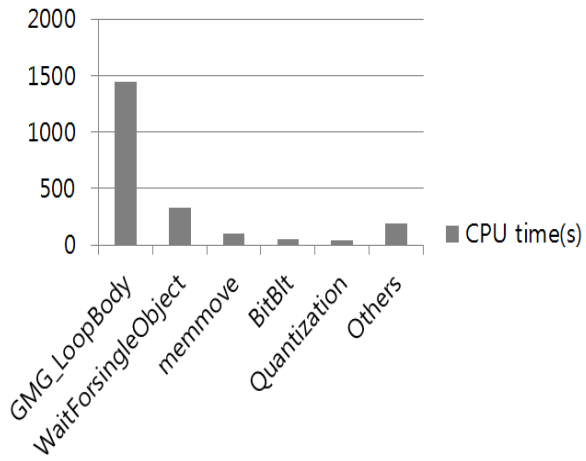


Fig. 9. Image Processing Program

프로그램의 분석 결과를 바탕으로, 상대적으로 CPU 사용률이 높은 함수들을 우선적으로 분류하고, CPU 사용률에 미치는 영향이 상대적으로 적은 함수들을 하나의 처리영역으로 하여 Fig. 9처럼 함수를 분류한다. 영상처리 프로그램의 실행 분석 결과, 각 왼쪽부터 약 1448, 325, 101, 50, 41, 187초의 CPU time을 가진다. 프로그램의 프로파일 결과 프로그램 내의 함수가 차지하는 CPU time이다. 위 결과를 토대로 함수의 CPU 점유율을 분석하고, 함수의 호출 관계를 분석하여 알고리즘에 적용시킨다. 함수의 호출 관계는 Fig. 10에 나타난다. CPU Total 점유율 100%를 기준으로 tmainCRTStartup이 23.6%, RtlUserThreadStart가 76.4%의 점유율을 가진다.

tmainCRTStartup에서 GMG::process와 Others는 각각 18.4%, 5.2%의 점유율을 가지며 RtlUserThreadStart의 GMG_Loopbody::operatot()와 dispatch는 각각 63.6%, 12.8%의 점유율을 가지며 Loopbody::operator()의 Loop@0x567672 와 Others는 각각 44.9%와 18.7%의 CPU 점유율을 가진다. 이는 CPU Total의 호출되는 최상위의 함수와 최상위 함수들의 최하위 호출 함수의 관계를 나타낸 것이다. 이 점유율의 분석값을 바탕으로 제안하는 알고리즘에 적용하여 코어 분할을 할 시, Fig. 11과 같이 나타나게 된다. 코어는 4개의 코어를 기준으로 분할한다.

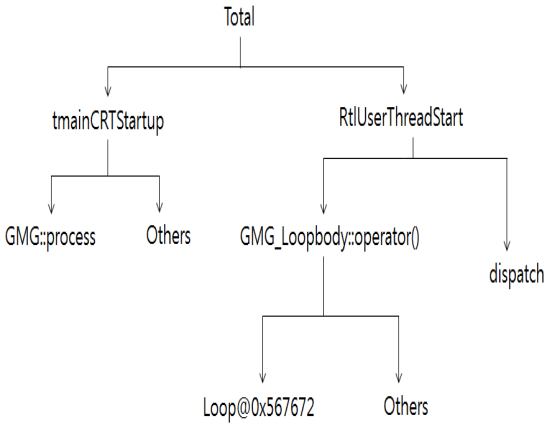


Fig. 10. Function Call tree

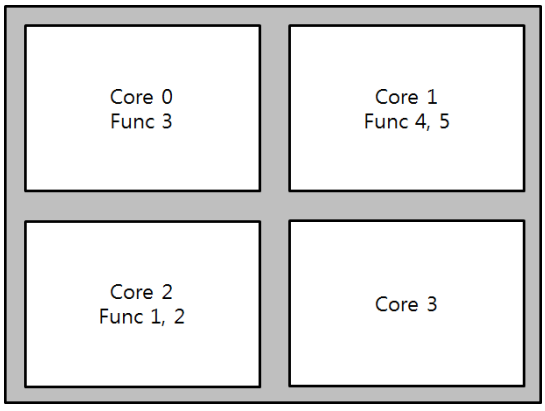


Fig. 11. Function Core allocation

함수 호출 트리에서 최소 단위 함수를 기준으로 왼쪽부터 Function 1~5까지 지정 후 알고리즘을 적용시키면 Fig. 11과 같이 분할된다. 코어의 Maximum time을 기준으로 분할하여 상대적으로 적은 CPU time을 가지는 함수의 경우 가장 높은 점유 시간을 가지는 코어를 제외한 나머지 Core에 순차적으로 배분하며 Core 0~2까지 함수의 CPU 점유율은 각각 44.9%, 31.5%, 23.6%가 된다. 분할되지 않은 Core 3의 경우 유휴상태, 즉 Idle 상태가 된다.

제시하는 알고리즘을 적용시킨 결과와 적용시키지 않은 결과를 비교하면 Fig. 12와 Fig. 13과 같다.

Fig. 12를 보면 적용 전 함수의 사용 시간은 1130초이며 적용 후의 함수의 사용 시간은 940초이다. 이는 wait time과 spin time 등의 하드웨어 이벤트의 시간을 포함한 총 CPU time 2093초에서 순수 함수가 차지하는 CPU time만을 기준으로 Maximum time이 1130초이며 제안하는 알고리즘을 적용시켰을 시 940초의 시간 소비가 된다.

이는 시간적으로 비교하였을 시 약 200초의 시간 감소가 있다. 이를 바탕으로 CPU 점유율을 나타내면 Fig. 13과 같이 나타낼 수 있다. 하드웨어 이벤트를 제외한 순수 CPU time만을 기준으로 기존의 스케줄러에서 코어에 가장 높은 점유율을 가지는 함수가 53.9%이며 알고리즘을 적용시켰을 때 점유율

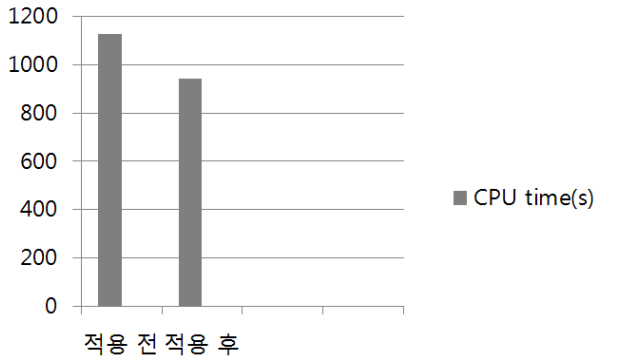


Fig. 12. Execution time

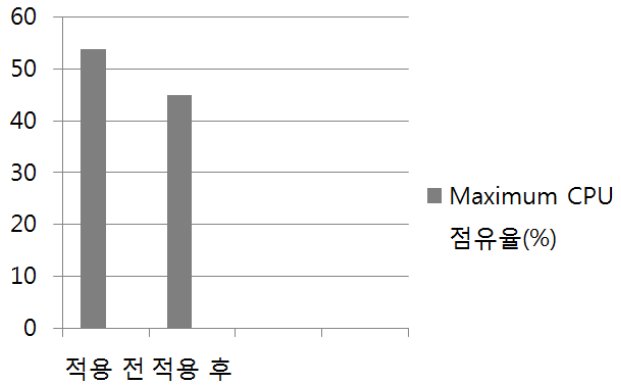


Fig. 13. Percentage of Maximum CPU

이 44.9%가 되는 것을 확인할 수 있다. 이는 기존의 스케줄러와 비교했을 시 약 17%의 성능 이득이 있음을 알 수 있다.

Table 1. Experimental Results(%)

	Before	after	Gain
Data buffer	78	57	27
Gallery	46	30	35
Image tool	46	38	18
System UIAClient	54	36	34
Sudoku	45	34	25

마찬가지로 데이터 버퍼에 이미지를 그리는 프로그램과 사진 보기 프로그램, 사진 편집 프로그램, 시스템 UIAClient 프로그램, Sudoku 프로그램을 분석하고 제시하는 알고리즘을 적용시켰을 시 Table 1과 같이 성능 이득이 있음을 알 수 있다.

Table 1을 보면 프로그램의 측정 결과 기존의 스케줄러에서 각 프로그램이 가지는 함수의 CPU 점유율이 가장 높은 함수가 각각 79%, 46%, 46%, 54%, 45%로 나타난다. 이와 비교하여 제안하는 알고리즘을 적용시킬 시 가장 높은 CPU 점유율이 57%, 30%, 39%, 36%, 34%가 되며 이것은 각각 27%, 35%, 18%, 34%, 25%의 성능 이득이 있다는 것을 알 수 있다.

5. 관련 연구

5.1 부하 균등화

부하 균등화는 프로세서에 부하의 재분배를 통해 병렬 및 분산 시스템의 성능을 향상시키는 과정이다. 부하 균등화의 주요 목표는 처리량을 최대화하기 위해 프로세서에 작업을 분산, 안정성을 유지하고 무결성을 가져야 한다.

부하 균등화는 정적 부하 균등화와 동적 부하 균등화로 구분할 수 있다. 정적 부하 균등화는 프로세서의 실행의 시작 부분에 결정된다[8, 9]. 그런 다음 성능에 따라 작업 부하는 마스터 프로세서에 의해 시작에 분포한다. 동적 부하 균등화는 런타임 프로세서에 분산된다는 점에서 정적 알고리즘과 다르다. 마스터 프로세서는 수집된 새로운 정보에 기초하여 슬레이브 프로세서에 새로운 프로세스를 할당한다[10, 11]. 정적 부하 균등화와 달리, 하나의 프로세서가 로드되었을 때 동적 알고리즘은 동적으로 프로세서를 할당한다. 대신에 프로세스들은 메인 호스트 큐에 버퍼링 되고 원격 호스트의 요청에 따라 동적으로 할당된다.

이러한 부하 균등화 알고리즘은 분산 시스템에 초점을 맞추고 SoC 시스템에 고려한 알고리즘이 없다. SoC의 중요한 특성은 분산 시스템의 속성과 다르다. 주요 요인은 자원의 수에 따른 자원 분석 오버헤드의 고려가 필요하다.

분산 시스템에 대한 부하 균등화 알고리즘은 일반적으로 가벼운 부하, 일반적인 부하, 무거운 부하의 세 가지 부하 상태의 통계를 사용한다[12]. 그러나 SoC에서 단독으로 세 개의 상태 시스템 부하 메커니즘은 최적이 아니다. SoC는 보통 다른 성능 및 리소스의 다양한 자원의 성능을 가지고 있기 때문에 자원의 부하 상태로 변화한다. 따라서, SoC에 대한 로드 밸런싱 알고리즘은 최적의 시스템 성능을 위해 성능과 자원의 부하 상태를 고려해야 한다[13].

5.2 멀티프로세서에서 작업 분배

MPSoC(멀티프로세서 시스템 온 칩)은 VLSI 시스템의 중요한 클래스로 지난 10년간 등장했다. 단일 칩 다중 프로세서 시스템 구성 요소로서 다수의 프로그램 프로세서를 사용하는 응용 프로그램에 대한 대부분 또는 모든 구성 요소에 필요한 통합 VLSI 시스템이다. MPSoC는 다른 애플리케이션 간의 네트워크 통신, 신호처리 및 멀티미디어에 사용된다[14]. 멀티프로세서는 대칭형 프로세서와 비대칭형 프로세서로 나눌 수 있다. 대칭형은 동종의 두 개 또는 그 이상의 동일한 프로세서가 서로 작업량을 나누어 처리하는 방식이다. 따라서 쓰레드 수준 병렬화(Thread Level Parallelism)와 명령어 수준 병렬화(Instruction Level parallelism)에 적합한 결합 형태이다. 비대칭형은 서로 다른 아키텍처의 프로세서를 사용하는 방식으로, 프로세서의 설계 목적에 따라 적절히 작업을 분담한다. 현재 임베디드 프로세서에서 멀티코어의 보편화된 형태이다. 보통 하나의 프로세서는 마스터로 설정되어 전반적인 시스템의 리소스를 담당하며, 나머지 프로세서들은 마스터로부터 할당받은 프로세스를 처리한다. 특히 서로 다른 아키텍처의 프로세서를 결합하는 방식은 서로의 아키텍처에 특화된 작업을 할당함으로써 작업 효율을 극대화시킬 수 있다.

MPSoC 시스템은 작업의 크기에 따라 적절히 프로세서 코어에 작업을 할당하여 병렬 처리함으로써 시스템 처리 속도를 높이고자 한다. 멀티프로세서에서 사용되었던 기법으로는 대표적으로 리스트 스케줄링 기법[15]과 클러스터링 기법이 있다[16]. 리스트 스케줄링 기법은 태스크들에 우선순위를 부여하여 그 일련의 순서대로 스케줄링 하는 방식이다. 우선순위의 결정 방법은 다음과 같다. 우선 프로그램을 태스크 그래프로 표현한다. 태스크 그래프를 분석하여 프로그램의 실행시간을 결정짓는 가장 긴 구간(critical path)에 포함된 태스크들부터 우선순위가 높은 것으로 부여한다. 클러스터링 기법은 최소 스케줄 시간이 될 수 있도록 태스크

서로 간의 통신비용이 많은 태스크들을 하나의 그룹으로 묶어서 이 그룹 단위로 프로세서 코어에 할당하는 방식이다. 멀티프로세서의 스케줄링 기법들은 우선 높은 통신비용을 해결하는 데 초점을 맞추어 설계되었다. 따라서 통신비용이 상대적으로 낮은 모바일 시스템의 시스템 온 칩에서는 이러한 기술을 그대로 사용하는 데 효율적인 측면에서 어려움이 있다.

리눅스 기반인 안드로이드에서도 프로세서 자원을 효율적으로 활용하여 사용자 응답성을 높이기 위한 스케줄링 기법이 사용되고 있다. 하지만 그럼에도 불구하고 사용자 응답성 측면에서 부정적인 평가가 보고되었다[17]. CFS 같은 경우 태스크들에게 공평한 CPU 할당 시간을 주도록 설계되어 있다. 하지만 리눅스 서버 환경에 특화되어있어 대화형 애플리케이션 환경에서는 효율적인 작업 할당이 되지 않는 것으로 알려졌다[18]. 현재 안드로이드 버전까지도 탑재된 MPSoC의 프로세싱 코어들의 효율적인 사용을 위한 스케줄링 기법이 적용되지 못하고 있는 실정이다.

6. 결 론

본 논문에서는 애플리케이션의 실행 분석을 통한 멀티코어 모바일 시스템 성능 최적화 기법에 대해 제시하였다. 모바일 디바이스에서 사용자 상호작용이 높은 7개의 애플리케이션의 실행 특성의 분석을 통하여 애플리케이션의 비효율적인 실행에 대해 검증하였고, 효율적인 자원 분배 및 성능 향상을 위한 알고리즘을 제시하였다.

제시하는 알고리즘의 검증을 위해 6개의 프로그램의 함수의 실행시간 및 CPU 점유율을 분석하고, 분석한 함수의 호출관계 분석을 통하여 알고리즘을 적용, 각 코어에 함수들을 분할 시 각 프로그램은 17%, 27%, 35%, 18%, 34%, 25%의 성능 이득이 있음을 확인하였고, 이는 기존 스케줄러의 방식보다 약 26%의 성능 이득이 있음을 확인하였다.

따라서, 본 논문에서 제시하는 알고리즘을 통하여 시스템 성능 향상 및 에너지 소모 면에서 성능 이득이 있을 것으로 본다.

References

[1] KOCCA, Korea Creative Content Agency “World Mobile Application Market Status and Forecast”.

[2] Korea Communications Commision, “Study on Security for New Mobile Devices” 2011. 12.

[3] KETI “Mobile CPU technology trends and industry Forecast”.

[4] Stuart Robinson, “Energy demand of handset applications is growing,” Battery Technology & Power Management Conference, pp.8, 18, Aug., 2005.

[5] Hesham El-Rewini, T.G. Lewis, “Scheduling parallel program tasks onto arbitrary target machines,” *Journal of parallel and distributed computing*, Vol.9, No.2, pp.138-153, Feb., 1990.

[6] nVIDIA tegra-whitepaper-0911b, "Variable SMP—A Multi-Core CPU Architecture for Low Power and High Performance".

[7] Feljan, J., Carlson, J. “Task Allocation Optimization for Multicore Embedded Systems,” Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, pp.237-244, 2014.

[8] Derek L. Eager, Edward D. Lazowska, and John Zahorjan, “Adaptive load sharing in homogeneous distributed systems,” *IEEE Transactions on Software Engineering*, Vol.12 No.5, p.662-675, May., 1986.

[9] R. Motwani, P. Raghavan, “Randomized algorithms,” ACM Computing Surveys (CSUR), Vol.28, No.1, pp.33-37, 1996.

[10] S. Malik, “Dynamic Load balancing in a Network of Workstation,” 95.515 Research Report, 19, Nov., 2000.

[11] Y.Wang, R. Morris, “Load balancing in distributed systems,” IEEE Trans. Computing. C-34, No.3, pp.204-217, Mar., 1985.

[12] Gil-Haeng Lee, “An Adaptive Load Balancing Algorithm Using Simple Prediction Mechanism,” Database and Export Systems Applications, pp.496-501, 1998.

[13] Shinwon Lee, Meka, V., Mingu Jeon, Nagoo Sung, and Jeongnam Youn, “Dynamic load balancing algorithm for system on chip,” SoC Design Conference (ISOCC), 2012 International, pp.208-211, 2012.

[14] Wolf, W., Georgia Inst. of Technol., Atlanta, GA, Jerraya, A. A., and Martin, G., “Multiprocessor System-on-Chip (MPSoC) Technology,” Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, pp. 1701-1713, Oct., 2008.

[15] Hesham El-Rewini, T.G. Lewis, “Scheduling parallel program tasks onto arbitrary target machines,” *Journal of parallel and distributed computing*, Vol.9, No.2, pp.138-153, Feb., 1990.

[16] A. Gerasoulis, T. Yang, “On the granularity and clustering of distributed acyclic task graphs,” *IEEE Trans. parallel and distributed systems*, Vol.4, No.6, pp.686-701, Jun., 1993.

[17] http://techland.time.com/2011/12/07/is-android-oomed-to-lag-more-than-ios/

[18] L.A. Torrey, J. Coleman, and B. Miller, “A comparison of interactivity in the linux 2.6 scheduler and an MLFQ scheduler,” *Software: Practice and Experience*, Vol.37, No.4, pp.347-364, 2007.

조 중 석

e-mail : icaroosion@sunchon.ac.kr
2012년 순천대학교 전자공학과(학사)
2014년 순천대학교 전자공학과(석사)
2014년~현 재 순천대학교 전자공학과 박사과정
관심분야 : 임베디드 시스템 설계/최적화, 모바일 시스템 최적화, 병렬/분산 컴퓨팅

조 두 산

e-mail : dscho@sunchon.ac.kr
2001년 한국외국어대학교 전자정보공학부 (학사)
2003년 고려대학교 전기전자공학부 (석사)
2009년 서울대학교 전기컴퓨터공학부 (박사)
2008년~2010년 ㈜ 리코어스 이사
2010년 삼성전자 System LSI 연구원
2010년~현 재 순천대학교 전기전자공학부 부교수
관심분야 : 임베디드 시스템 설계/최적화, 메모리 시스템 최적화, 저전력설계, 최적화 컴파일러, 클라우드 컴퓨팅, 병렬/분산 컴퓨팅