논문 13-38C-06-04

한국통신학회논문지 '13-06 Vol.38C No.06
http://dx.doi.org/10.7840/kics.2013.38C.6.521

멀티코어 기반 모바일 플랫폼을 위한 애플리케이션의 태스크 병렬화 시스템

임 근 식◆, 이 세 호*, 엄 영 익°

Task Parallelism System of Application for Multicore-Based Mobile Platform

Geunsik Lim◆, Seho Lee*, Young Ik Eom°

요 약

본 논문은 기존의 소프트웨어가 멀티코어기반의 모바일 디바이스를 인지할 수 있도록 태스크 병렬화 시스템(BioMP)을 제안한다. 애플리케이션 개발자가 기존의 소프트웨어에 병렬화 규약의 코드를 추가하였을 때, 제안 시스템은 호환성 뿐만 아니라 병렬 쓰레드의 수행을 지원한다. BioMP는 기존의 대용량 애플리케이션 소스코드를 단시일에 멀티코어를 인지하는 소프트웨어로 개선한다. 실험 결과, 우리의 아이디어는 쿼드 코어기반의 멀티코어 환경에서 기존의 시스템 대비 애플리케이션 실행속도를 약 64%까지 개선하였다. 또한, BioMP는 독립적인 컴포넌트이기 때문에 어떠한 플랫폼의 추가적인 수정도 필요로 하지 않는다. 그 결과, 애플리케이션 개발자는 멀티코어향 소프트웨어를 애플리케이션 스토어에 배포하였을 때, 사용자는 모바일 디바이스의 어떠한 수정도 없이 즉시 실행을 할 수 있다.

Key Words : Multicore, Mobile platform, Android, OpenMP, Parallel programming, Task parallelism

ABSTRACT

This paper proposes a task parallelism system (BioMP) to improve applications' execution time of multicore based mobile device. When application developers append the functions of parallel specification into the existing software, our proposed system supports the parallel processing of threads as well as a compatibility. BioMP improves the software in order that an existing large-scale source can recognize the multicore architecture. From our experiment, our idea improved the execution time of application until about 64% against the existing system in multicore environment based on quad core. In addition, BioMP does not require any additional modification of a mobile platform because BioMP is independent component. Consequently, when application developers release multicore-aware applications into the application store, users can immediately run without any modification of the mobile device.

Ⅰ. 서 론

과거에 멀티코어 기술에 대한 연구는 소프트웨어의 처리속도를 개선하기 위하여 서버 환경에 집중하여 연

※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행된 연구이다. [10041244, 스마트TV 2.0 소프트웨어 플랫폼].
◆ 주저자 : 성균관대학교 정보통신대학 분산컴퓨팅 연구실, leemgs@skku.edu, geunsik.lim@samsung.com, 학생회원
° 교신저자 : 성균관대학교 정보통신대학 분산컴퓨팅 연구실, yieom@skku.edu, 정회원
* 성균관대학교 정보통신대학 분산컴퓨팅 연구실, loadic@skku.edu, 학생회원
논문번호 : KICS2013-03-136, 접수일자 : 2013년 3월 19일, 최종논문접수일자 : 2013년 5월 27일

구되었다. 최근 들어 애플리케이션 스토어를 내장하고 있는 스마트폰이 일반인들에게 급증하고 있다. 제조사가 개발한 애플리케이션들만 실행할 수 있는 폐쇄적 구조의 모바일 환경이 사용자가 개발한 애플리케이션들도 실행이 가능한 개방형 스마트 모바일 환경으로 빠르게 진화하고 있다. 기존에 데스크톱 컴퓨터에서만 실행을 할 수 있었던 고성능의 애플리케이션[1-3] 들을 (예: 게임, 가상현실, 웹브라우저 등) 스마트폰에서도 실행하고자 하는 요구가 급증함에 따라 멀티코어 CPU가 모바일 환경에서도 중요한 기술 중의 하나가 되고 있음을 그림 1을 통해서 보여주고 있다.

그림 1. 멀티코어 CPU 의 출현
Fig. 1. The appearance of multicore CPU

멀티코어 CPU가 스마트폰에서 개발된 고성능 애플리케이션들의 처리속도를 개선하기 위하여 모바일 디바이스에 도입되었다. 즉, 서버뿐만 아니라 모바일 디바이스에도 우리는 멀티코어 CPU를 어렵지 않게 접할 수 있다. 하드웨어 기반의 멀티코어 아키텍처를 효과적으로 이용하기 위해서 개발자들은 애플리케이션들의 쓰레드 및 병렬화 프로그래밍을 신중히 해야 한다. 프로그래머가 이미 개발되어 있는 대규모 프로젝트의 소프트웨어코드들을 멀티코어기반의 새로운 디바이스를 위해서 처음부터 다시 개발하는 것은 개발시간 및 개발 비용적인 면에서 상당한 부담이다. 안드로이드와 같은 모바일 플랫폼에서 병렬화를 위한 프로그래밍 규약의 구현 결과인 OpenMP API[4] 를 이용하여 기존의 코드를 병렬화 하는 것이 중요하다. 따라서 모바일 애플리케이션의 쓰레드들이 멀티코어 환경에서 완전하게 병렬로 수행이 가능한 컴포넌트가 필요하다.

우리는 기존의 대용량 애플리케이션 소스코드가 멀티코어 아키텍처를 인지하여 동작할 수 있도록 BioMP (Bionic and OpenMP for Multicore-aware Mobile Platforms)라는 새로운 시스템 독립적인 병렬화 컴포넌트 기술을 제안한다[4]. BioMP는 기존에 개발된 대형 프로젝트의 소스코드를 빠른 시간 내에 멀티코어 모바일 환경에 최적화된 자바 애플리케이션을 재생산할 수 있도록 돕는다. 제안하는 기법의 실효성을 검증하기 위하여 상업적으로 사용 중인 안드로이드 모바일 플랫폼에서 구현 및 평가를 하였다.

이 후의 섹션들은 다음과 같이 구성되어 있다. 섹션 II는 관련 연구들의 제안 기법 및 장단점을 요약한다. 섹션 III는 제안하고자하는 BioMP의 시스템 아키텍처를 설명한다. 섹션 IV에서는 모바일 플랫폼의 애플리케이션이 멀티코어를 인지할 수 있도록 제안하려는 BioMP의 구현을 기술한다. 섹션 V에서는 제안 기법의 효과를 검증하기 위해 실험한 결과를 설명한다. 최종적으로 섹션 VI에서 제안된 아이디어를 결론짓는다.

Ⅱ. 관련 연구

Google의 Bionic[5,6]은 *POSIX (Portable Operating System Interface)*를 준수하는 Pthread를 이용한 쓰레드 프로그래밍을 통해서 애플리케이션의 이식성 및 호환성을 제공한다. 그러나 기존에 작성된 소스코드를 멀티코어 환경을 위해 *Pthread API*를 이용하여 병렬화 하려면 기존의 소스코드 자체를 대폭 수정해야 하기 때문에 많은 개발시간이 필요하다. 더군다나, 기존에 개발된 소스코드의 용량이 많은 경우에는 그 소스코드 개선하기 위하여 분석하는데 엄청난 시간이 필요하다. 또한, *OpenMP API*[7,13] 의 개수보다 대략 3배가 많은 120여 개 이상의 *POSIX Pthread API*들을 정확히 이해하고 사용해야 한다는 문제를 내포하고 있다.

OpenMP[3,13,17] 는 기존의 C/C++ 소스코드를 단시일 내에 재사용할 수 있도록 돕는다. 실제로 *POSIX Pthread API*가 120여개이고, *OpenMP API*가 45개 미만인 점을 고려하였을 때, 개발시간 단축이라는 강점은 매우 크다. 그러나 OpenMP는 서버향 멀티코어 환경에서의 고성능 컴퓨팅에만 집중하고 있기 때문에, 전력관리를 도모해야하는 모바일의 CPU 환경을 고려하여 설계 및 구현되어 있지 않다. 또한, OpenMP는 C, C++, Fortran만을 최적화할 수 있기 때문에 안드로이드 모바일 플랫폼에서 채택하고 있는 달빅이라는 가상머신위에서 실행되는 자바 애플리케이션의 지원을 고려하지 않는다. 따라서 기존의 OpenMP[12] 프레임웍

은 안드로이드 모바일 플랫폼에서 실행이 불가능하다.

Yu-Hao Chang[1] 은 멀티코어 기반의 안드로이드 환경을 위한 소프트웨어 프레임웍을 제안한다. 이 소프트웨어 프레임웍은 2개의 컴포넌트들을 구성한다. 1) 멀티코어 컴파일러 툴킷, 멀티코어 디버거, 퍼포먼스 측정 툴, OpenCL 컴파일러 등을 구성하고 있는 시스템 소프트웨어와 2) 멀티코어 코덱, 멀티코어 C++, SIMD Intrinsic등의 시스템 라이브러리로 구성된다. 제안하는 소프트웨어 프레임웍을 통하여 *Face detection, voice recognition, mobile streaming management* 등의 애플리케이션의 성능을 개선할 수 있다. 이 기법은 기존에 존재하는 멀티코어 기법 및 기능들을 2개의 컴포넌트로 패키징하기 때문에 구현이 용이하다. 이 방법은 2개의 컴포넌트를 잘 사용하여 애플리케이션들을 구현 후 디버깅을 통해 최적화를 수행함으로써 애플리케이션들이 멀티코어 환경에서 최적인 성능을 나타낼 수 있음을 설명하고 있다. 그러나 기존에 개발되어 있는 대용량의 소스코드를 멀티코어 환경에 맞게 재배포하려면 기존의 소스코드를 완전하게 이해해야만 이식할 수 있으며, 제안된 2개의 멀티코어향 컴포넌트를 제대로 활용하려면 일관성 있는 개발자 API 정의가 추가로 필요하다. 또한, 이 방법은 기존의 코드를 멀티코어향에 맞게 최적화 할 수 있는 기법에 대한 연구를 다루지 않는다.

Ⅲ. 설 계

본 논문은 애플리케이션이 멀티코어 모바일 시스템을 인지하여 동작하도록 병렬화 해주는 BioMP 시스템을 제안한다. 본 섹션은 새롭게 제안하고자 하는 BioMP[4] 가 멀티코어향 자바 애플리케이션에 어떻게 포함되어 동작하는지에 대해서 우리의 시스템 아키텍쳐를 상세히 설명한다. 특히, BioMP 기반의 멀티코어향 모바일 애플리케이션은 안드로이드 플랫폼의 어떠한 추가적인 수정도 요구하지 않는다. 즉, 우리의 BioMP를 통해서 수정된 멀티쓰레디드 병렬 애플리케이션들은 안드로이드 모바일 플랫폼에 의존적이지 않다. 또한, 개발자들은 BioMP를 사용하기 위하여 어떠한 플랫폼의 수정도 필요하지 않다. 본 섹션에서는 BioMP가 애플리케이션 스토어에 어떻게 업로드, 다운로드, 설치, 실행 등이 가능한지에 대해 설계 및 구현에 대해 상세히 설명한다.

3.1. 사용자 애플리케이션의 병렬화

기존 시스템은 멀티코어에 최적화된 애플리케이션을 개발하기 위해서 소프트웨어의 세부 동작 구조를 분석해야만 전체 소스코드를 멀티코어향으로 병렬화 할 수 있다. 이 때문에 애플리케이션의 병렬화를 위한 추가적인 소프트웨어 개발비용이 심각해진다. 또한, 개발 프로젝트의 소스코드 양이 클수록 소스코드를 올바르게 분석하기 위한 시간적인 비용은 그에 비례한다.

개발 비용의 관점에서 OpenMP[4] 규약의 프로그래밍방법은 이미 개발되어 있는 기존의 대용량 애플리케이션 소스코드를 단 시일에 멀티코어를 인지하는 소프트웨어로 재생산할 수 있는 효과적인 방법이다. 개발자가 OpenMP를 ARM기반의 안드로이드 모바일 플랫폼에서 실행하려면 OpenMP의 라이브러리 호출 구조를 안드로이드 플랫폼의 Bionic[5,6] 라이브러리에 이식시키기 위해서 추가적인 설계가 필요하다. 시스템 콜과 라이브러리 콜은 Bionic이라고 명명된 C 라이브러리 레벨에서 구현되기 때문이다.

3.2. BioMP의 지원 범위

사용자 애플리케이션이 멀티코어 기반의 모바일 플랫폼에서 병렬화 가능하도록 본 논문에서 제안하고자 하는 기술은 크게 2가지의 범주로 나뉜다. 첫째, OpenMP[7] 가 안드로이드 모바일 플랫폼의 *Bionic C library*[5] 와 통합되어 애플리케이션의 쓰레드들이 병렬로 수행되고, 안드로이드 *EABI (Embedded Application Binary Interface)*에 호환되도록 설계한다. 둘째, 항상 전력 소모를 중요하게 고려해야하는 모바일 디바이스 환경은 *CPU-Hotplug* 와 *CPU-DVFS (Dynamic Voltage & Frequency Scaling)*을 사용하기 때문에, 이러한 모바일 환경의 특성을 올바르게 인지하여 병렬로 쓰레드 수행하는 구조를 추가한다.

3.3. BioMP의 전체 아키텍쳐

그림 2는 우리가 새롭게 설계한 BioMP의 전체 아키텍쳐를 보여준다[8]. 그림 2에서 BioMP으로 표시된 점선의 네모 박스는 안드로이드 모바일 플랫폼이 멀티코어 CPU를 올바르게 인지할 수 있도록 OpenMP 규약 기반으로 구성된 애플리케이션들을 해독 및 실행하는 역할 담당한다. 점선의 네모 박스들은 기존의 안드로이드 플랫폼이 멀티코어 CPU를 인지할 수 있도록 추가되어야 하는 BioMP의 주요 기능들을 나타내고 있다. 점선의 네모박스로 그려진

BioMP는 OpenMP의 프로그래밍 규약 API를 *Bionic C library*[5,6] 의 POSIX 쓰레드 API에 연결시키고, 안드로이드 플랫폼의 EABI와 호환시키는 핵심 코어 역할을 한다.

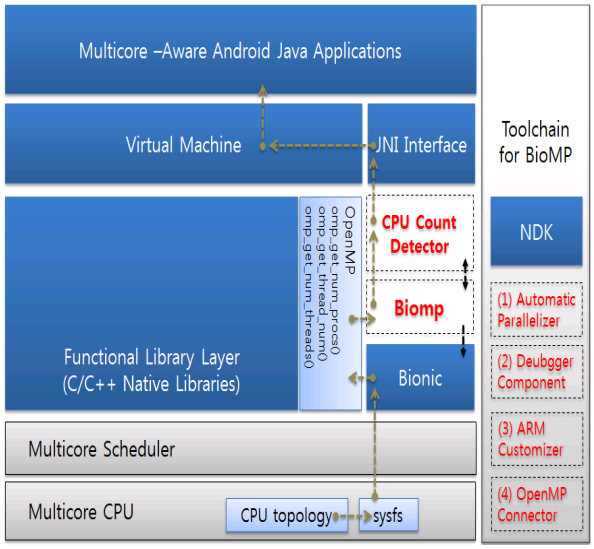


그림 2. BioMP의 아키텍처
Fig. 2. The BioMP architecture

3.4. BioMP 툴체인

OpenMP[7,9] 규약의 API가 추가된 기존의 애플리케이션 소스는 컴파일 시점에 BioMP를 포함한다. 그림2의 툴체인 컴파일러는 BioMP가 수정된 기존의 소스코드를 컴파일 시점에 정적 링킹 또는 동적 링킹한다. BioMP가 컴파일시점에 JNI 인터페이스로 구현되는 자바 애플리케이션들과 결합하기 위해서 툴체인 컴파일러는 아래와 같이 4가지의 주요 기능들을 구성한다.

3.4.1. Automatic Parallelizer

컴파일러가 애플리케이션 소스코드를 실행 가능한 바이너리 포맷으로 변환하는 시점에 루프문의 코드를 자동으로 병렬화하는 작업을 수행한다. 애플리케이션의 Hotspot 구간은 대부분 반복문 (loop statement)이다. 따라서 개발자가 이 반복문을 최적화하기 위해서 *-ftree-parallelize-loops=n* 형식으로 루프문의 병렬화 쓰레드 개수를 컴파일 시점에 지정할 수 있다.

3.4.2. Debugger Component

컴파일 된 코드가 실행이 될 때 라이브러리 콜의 비용과 시스템 콜의 비용이 어떻게 되는지 분석하기 위한 콜 트레이서이다. 애플리케이션 개발자는 이 컴포넌트를 이용하여 애플리케이션의 수행시간을 최적화하기 위해서 작업해야 하는 함수 정보들을 쉽게 알 수 있다.

3.4.3. ARM Customizer

이 컴포넌트는 소스코드를 바이너리 코드로 생성할 수 있도록 돕는다. 해당 아키텍처 및 코어에 대한 정보를 명시함으로써 최적화된 CPU 인스트럭션을 생성한다. CPU의 코어를 정확히 지정하는 것은 애플리케이션의 실행속도를 향상시킬 수 있음을 의미한다. 그러나 특정 CPU 코어에 집중되는 최적화는 하위 CPU 코어에서도 프로그램이 실행될 수 있는 호환성을 손상시키는 문제를 같이 내포한다.

3.4.4. OpenMP Connector[7]

이 컴포넌트는 OpenMP 규약으로 다시 작성된 기존의 소프트웨어가 실행할 때, 안드로이드 모바일 플랫폼이 멀티코어 CPU를 인지할 수 있도록 돕는다. 애플리케이션의 병렬 쓰레드 수행이 가능하도록 BioMP 컴포넌트에 연결시키는 작업을 담당한다. OpenMP 규약의 API가 추가된 기존의 소스코드는 BioMP 툴체인과 BioMP을 통해서 병렬로 실행 가능한 최종적인 바이너리 파일로 생성된다.

3.5. BioMP Core

우리의 제안 시스템에서 병렬로 실행 가능한 바이너리 파일은 안드로이드 모바일 플랫폼 레벨에서 아래와 같이 2개의 주요 컴포넌트를 통해서 멀티 쓰레드화된 자바 애플리케이션으로써 병렬 수행을 한다.

3.5.1. BioMP

OpenMP[9] 규약으로 작성된 소스코드를 Bionic[6] 기반의 안드로이드 모바일플랫폼에서 호환하여 동작시키는 컴포넌트이다. OpenMP 자체는 POSIX 와 호환하지 않는 라이브러리 콜이다. 따라서 모바일 플랫폼이 OpenMP의 API를 해독할 수 있도록 시스템 콜 API가 구현되어 있는 Bionic[5] 라이브러리와 연결을 해야 한다. BioMP Core는 Bionic Library가 OpenMP 규약으로 작성된 코드를 올바르게 인식하여 수행할 수 있도록 병렬화 문장을 변환하는 주요 역할을 담당한다.

3.5.2. CPU Count Detector

배터리 수명은 모바일 디바이스에서 가장 중요한 이슈 중의 하나이다. 모바일 디바이스들은 배터리를 최대한 장시간 사용하기 위하여 CPU가 동작하지 않아

도 될 때는 disable시키는 CPU-Hotplug와 CPU의 frequency를 동적으로 조절하는 CPU-DVFS를 이용하여 배터리 소모량을 최소화한다. *CPU Count Detector*는 Online 상태의 CPU 정보, Offline되어 있는 CPU 정보, 실제로 실행 가능한 CPU 정보 등을 모니터링 함으로써, 병렬화를 위한 최적의 쓰레드 개수를 결정할 수 있다. 사용자 공간에서 결정된 쓰레드 개수들은 커널 레벨의 멀티코어 로드 밸런서를 통해서 스케줄링된다. *CPU Count Detector*는 멀티코어에 최적화된 쓰레드 개수를 산출함으로써 커널 레벨에서 멀티코어 로드밸런싱 동작 시 과도한 태스크 마이그레인션으로 인한 락킹비용 증가 문제를 최소화한다. 이를 위해 실시간으로 CPU의 On-line 및 Off-line여부를 감지해야 하며, *CPU Count Detector*가 이러한 정보들을 수집 및 파싱한다.

Ⅳ. 구 현

일반적으로 모바일 디바이스의 리눅스 커널 및 플랫폼의 수정은 Root권한을 필요로 한다. 플랫폼을 수정하여 애플리케이션을 실행하는 방법은 보안상의 취약점이 발생할 수밖에 없다. 또한, 이러한 수정작업들을 하려면, 사용자는 시스템 지식들이 필요하기 때문에 플랫폼을 수정해야만 개선된 애플리케이션을 실행할 수 있는 방식으로 사용성이 떨어진다. 따라서 우리가 제안하는 BioMP는 기존의 안드로이드 기반의 모바일 디바이스를 수정하지 않고도 실행이 가능하도록 구현한다.

4.1. BioMP의 자료구조

BioMP는 커널레벨에서 제공되는 *sysfs*라는 기존의 시스템 파일시스템의 정보들을 수집한 후 해독한다. 해독된 정보들은 정수타입의 간단한 자료구조 타입을 통해 메모리 사용을 최소화한다. 그리고 다른 함수들이 CPU정보를 열람할 수 있도록 하나의 글로벌 변수로써 관리한다. CPU정보를 수집하기 위해 발생하는 비용을 최소화하기 위하여 사용자 공간인 */sys/*으로 정보들을 제공하는 *sysfs*를 이용하는 방법으로 구현한다. 그리고 모바일 디바이스의 플랫폼을 수정해야하는 상황을 회피하기 위하여 컴파일러가 BioMP의 라이브러리를 사용자의 애플리케이션 코드에 포함하는 구조를 동작함으로써, 기존의 모바일 디바이스에서 문제없이 실행 가능하도록 애플리케이션을 배포할 수 있다.

4.2. 애플리케이션의 병렬화

OpenMP[7,10] 규약으로 수정된 기존의 소프트웨어가 안드로이드 모바일 플랫폼에서 실행되기 위하여 아래와 같이 3단계의 작업이 순차적으로 실행되도록 구현한다. 그림 3은 사용자의 자바 애플리케이션을 BioMP와 어떻게 결합시키는 지에 대해서 세부적인 구현 방법을 나타내고 있다. OpenMP 규약으로 수정된 소프트웨어가 멀티코어 CPU를 최대한 활용하여 처리속도를 개선하기 위해서는 안드로이드 플랫폼이 멀티코어를 인지하여 애플리케이션을 실행할 수 있어야 한다. 이를 위해서 BioMP를 이용하는 애플리케이션들은 아래와 같이 3 단계의 주요 과정을 순차적으로 수행한다.

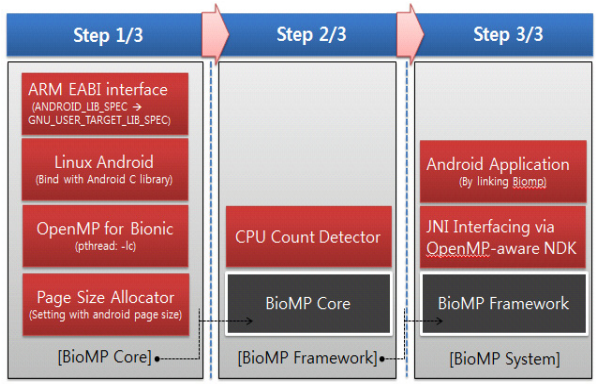


그림 3. BioMP와 애플리케이션의 결합 순서
Fig. 3. Combination sequence of Biomp and application

Step 1) BioMP Core : BioMP 코어는 사용자의 애플리케이션들을 병렬로 수행하기 위해서 OpenMP API[11]를 안드로이드 플랫폼의 Bionic API에 맵핑하는 작업을 수행한다. 제일 먼저, OpenMP 규약의 API으로 작성된 실행 가능한 바이너리 파일이 안드로이드 플랫폼에 수행될 수 있도록 *ARM EABI interface*을 구성한다. 그리고 나서, OpenMP API 가 Bionic 라이브러리에 정상적으로 결합되어 실행되도록 Android 플랫폼의 동작구조와 호환시키기 위하여 *Linux Android*라는 기능을 구성한다. 그 다음, OpenMP 규약의 코드가 Bionic 라이브러리 내에서 호환하여 동작이 가능하도록 *OpenMP for Bionic*를 구성한다. 마지막으로 안드로이드 플랫폼을 위해 실행되는 리눅스 커널의 페이지 사이즈를 할당하기 위한 *Page Size Allocator* 기능을 구성한다. *BioMP Core*는 OpenMP[12] 규약으로 재 작성된 기존의 소스코드가 안드로이드 플랫폼에서 실행이 가능하도록 필요한 작업들을 수행한다.

Step 2) BioMP Framework : BioMP Framework는 *BioMP Core*와 배터리 수명을 인지하는 멀티코어 CPU를 모니터링 하는 *CPU Count Detector*로 구성된다. 대부분의 모바일 디바이스는 배터리 수명을 연장하기 위하여 CPU-Hotplug 또는 CPU-DVFS 기법을 사용한다. "CPU Count Detector" 기능은 실시간으로 CPU 정보들을 (예: Online CPU정보, Offline CPU정보, 실제 물리 CPU정보)을 모니터링하기 위하여 리눅스 커널의 *sysfs (system file system)* 를 이용하여 수집된 정보를 해독한다. 이때, *sysfs*를 이용한 CPU 정보들은 *CPU topology*가 가지고 있는 계층적 트리구조에 기반을 두어 정보를 수집하므로 수집을 위한 비용이 기존의 */proc/cpuinfo* 또는 */proc/stat* 방법보다 훨씬 뛰어나다.

Step 3) BioMP System : *BioMP System*은 BioMP Framework을 이용하여 안드로이드 자바 애플리케이션을 패키징한다. 이 단계에서 애플리케이션이 실행될 수 있는 최종적인 상태가 된다. 이때, BioMP의 내부 동작 함수들은 JNI 인터페이스를 통하여 최상위의 자바 애플리케이션들과 연결된다.

4.3. 모바일 디바이스에서 CPU 카운트 검색

모바일 디바이스는 그 특성상 CPU를 항상 Online상태로 유지하지 않는다. 즉, 모바일 디바이스의 배터리 소모를 최소화하기 위하여 CPU가 사용되지 않는 시점에 일시적으로 Offline시키는 작업을 계속적으로 반복 수행한다.

모바일 디바이스의 CPU마다 상이한 아키텍쳐 구조를 가지고 있기 때문에 다른 DVFS값과 Hotplug 동작 기준점을 가진다. 예를 들어, 각각의 모바일 디바이스는 CPU의 성능과 배터리 소모의 상관관계에 따라, DVFS와 Hotplug가 동작되는 기준 시점을 다르게 통제한다.

기존의 시스템은 CPU의 개수에 상응하는 쓰레드를 생성하여 애플리케이션이 병렬로 수행되도록 처리한다. 이때, 기존의 방법들은 CPU의 정보를 얻기 위해서 */proc/cpuinfo* 또는 */proc/stat* 등의 proc 파일시스템을 이용한다. 이 방법의 문제는 실제로 실행 가능한 물리적인 CPU의 개수 정보가 아닌 Online 상태에 있는 CPU의 정보만을 수집할 수 있는 동작 구조를 가지고 있다. 일예로써, 그림 4와 같이 기존의 시스템에서 2개의 CPU가 Online이고, 2개의 CPU가 Offline이라고 가정한다. 이때, */proc/cpuinfo* 또는 */proc/stat*으로부터 얻을 수 있는 정보는 Online 상태에 있는 CPU들이다. 즉, 이 상황에서 CPU의 개수는 2개가 되고, CPU를 100% 사용하는 쓰레드를 2개를 만들게 되며, 이로 인해 나머지 2개의 Offline CPU는 영원히 사용되지 않는다. 그 결과, 멀티코어 CPU을 사용하고 있음에도 불구하고 애플리케이션의 성능은 멀티코어 CPU 도입의 효과를 얻지 못하게 된다.

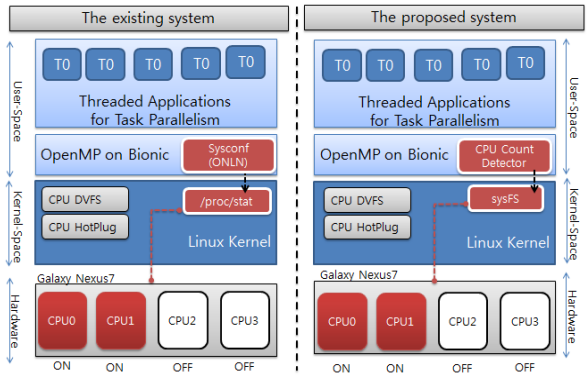


그림 4. CPU-Hotplug/CPU-DVFS기반의 모바일 디바이스를 위한 CPU 인식 방법
Fig. 4. CPU recognition method for CPU-Hotplug/CPU-DVFS based mobile device

제안하는 시스템은 효과적인 태스크 병렬화를 지원하기 위하여 Online 상태의 CPU정보, Offline 상태의 CPU정보, 실제로 사용가능한 CPU정보들을 모니터링 한다. 이를 위해서 계층적 트리 구조를 유지하는 *CPU topology*의 논리적 구조를 기반으로 하여 *sysfs*의 정보들을 런타임에 해독한다. 그림 4는 *CPU Count Detector*가 *procfs*가 아닌 *sysfs*를 통해서 필요한 점보를 수집/파싱하는 역할을 담당한다.

우리가 제안하는 기법을 통해서, 2개의 CPU가 Online 상태이고, 2개의 CPU가 Offline 상태인 경우에 실제로 사용가능한 CPU의 개수를 4개로 계산한다. 그리고 이에 상응하는 4개의 쓰레드 개수를 만들어 병렬로 수행하도록 관리한다. 이 때, 리눅스 커널은 CPU의 Utilization이 100%에 가까이 도달하는 지의 여부를 체크 후, Offline되어 있는 CPU2개를 즉시 Online상태로 바꾼다. 결과적으로 4개의 CPU가 모두 enable됨으로써 태스크의 병렬 처리속도를 효과적으로 수행할 수 있다.

4.4. 리눅스 커널에서 CPU 카운트 적용 사례

리눅스 커널은 부팅시점에 최대로 사용가능한

CPU의 개수를 지정할 수 있도록 *.config* 파일을 통해서 *CONFIG_NR_CPUS* 변수를 제공한다. 그리고 부팅시점에 매개변수 인터페이스를 통해서 *possible_cpus=N* 형식으로 사용가능한 CPU의 개수를 선언할 수 있다. 그림 4의 *sysfs*를 통한 CPU정보가 어떻게 산출되는지에 과정에 대한 예제가 그림 5이다. 그림 5는 실제로 실행 가능한 CPU 카운트를 검사하기 위해서 2가지의 사례를 보여주고 있다.

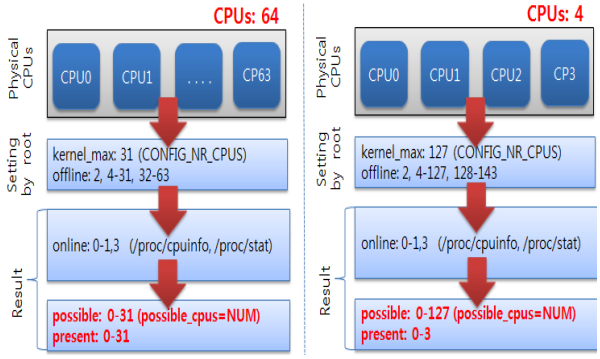


그림 5. 실제 실행 가능한 CPU 카운트 검사 예제
Fig. 5. Example to monitor actual CPU count

표 1. 그림 5의 좌측 예제 실행 결과
Table 1. Execution result of the left example on Figure 5

CPU Type	CPU Count
Online CPU	0-1, 3
Possible CPU	0-31
Actual CPU (*)	0-31

표 2. 그림 5의 우측 예제 실행 결과
Table 2. Execution result of the right example on Figure 5

CPU Type	CPU Count
Online CPU	0-1, 3
Possible CPU	0-127
Actual CPU (*)	0-3

1) **그림 5의 좌측 예제** : 물리적인 CPU가 80개인 시스템이 있다. 이때, 컴파일시점에 *CONFIG_NR_CPUS*의 값을 31로 입력하고, CPU의 2, 4-31, 32-79번을 Offline 하였다면, 표 1의 결과와 같다[10,13]. *CPU Count Detector*는 "실제 CPU"의 값 기준으로 32개의 신규 병렬 쓰레드를 생성한다.

2) **그림 5의 우측 예제** : 물리적인 CPU가 4개인 시스템이 있다고 가정한다. 이때, 컴파일 시점에 *CONFIG_NR_CPUS*의 값을 127로 입력하고, CPU의 2, 4-127, 128-143번을 Offline 하였다면, 표 2의 결과와 같다.

*CPU Count Detector*는 "실제 CPU"의 값을 기준으로 4개의 신규 병렬 쓰레드를 생성하는 방식으로 동작한다.

V. 성능 평가

5.1. 실험 환경

그림 6. 실험 환경
Fig. 6. Experimental environment

우리가 제안하는 아이디어가 실제로 멀티코어 기반의 모바일 디바이스 환경에서 효과적인지 검증을 하기 위하여 쿼드코어 CPU를 내장하고 있는 삼성 *ARM Exynos* 기반의 개발보드에 Android *Jelly Bean version 4.2*을 포팅 후 BioMP를 적용하였다. 그림 7은 성능평가를 위해 포팅 및 BioMP가 적용된 실험환경이다.

5.2. 실험 시나리오

그림 7과 그림 8은 CPU 사용률이 높은 피보나치 수열을 실행하였을 때, 제안하는 시스템이 기존대비 실행속도가 얼마나 개선되는 지를 측정한 결과를 보여주고 있다. (1) 수식은 피보나치수열의 구현 공식이다. (2) 수식은 수열의 합산 시 Seed가 되는 F_0, F_1의 상수 값이다.

$$F_n = F_{n-1} + F_{n-2}, \quad (1)$$

$$F_0 = 0, F_1 = 1. \quad (2)$$

1씩 증가시키면서 최대 40회 까지 피보나치수열을 반복 수행하였을 때, 기존의 시스템 (before) 과 제안한 BioMP (after)의 처리속도를 비교 실험하였다[14].

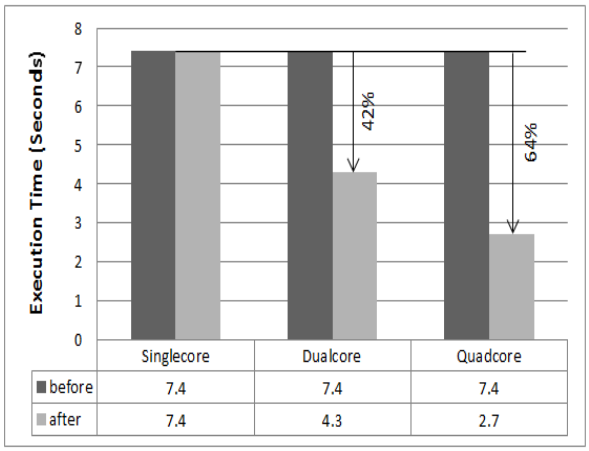


그림 7. BioMP의 실험 결과
Fig. 7. Experimental result of BioMP

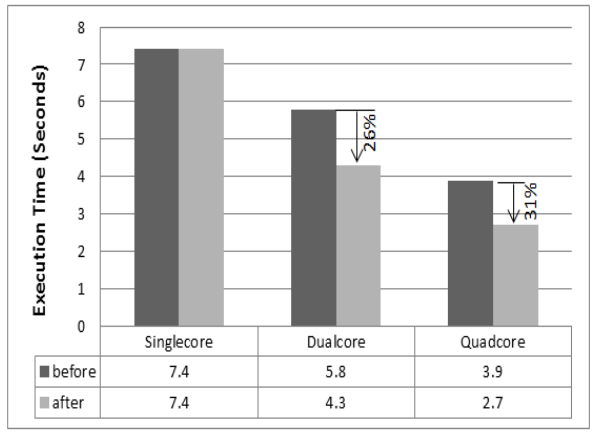


그림 8. 기존 시스템의 멀티쓰레드화 후 실험결과
Fig. 8. Experimental result after multi-threading of the existing system

5.3. 실험 결과

그림 7은 제안하는 시스템이 기존대비 얼마나 개선될 수 있는지 실험한 결과이다.

기존의 시스템은 싱글코어, 듀얼코어, 쿼드코어에서 동일한 수행시간 (7.4초)을 보여줌으로써 멀티코어에 대응하지 못하고 있음을 볼 수 있다. BioMP를 적용한 시스템 (after)은 싱글코어에서 기존의 시스템과 동일하게 7.4초의 수행시간이 소요되었다. 듀얼코어 환경에서 제안하는 시스템은 4.3초의 수행시간 (싱글코어 대비 42%의 수행시간 단축)이 소요되었다. 쿼드코어환경에서는 우리가 제안하는 BioMP를 적용하였을 경우, 2.7초의 수행시간이 소요됨에 따라 싱글코어 대비 64%의 수행시간을 단축할 수 있음을 확인하였다.

그림 7의 실험 결과를 보면, 멀티코어 CPU의 개수가 배수로 증가하였을 때, 성능이 멀티코어 CPU의 개수에 비례하여 개선되지 않음을 볼 수 있다. 그 결과의 주요 원인은 실제로 태스크의 수행이 종료될 때까지 항상 100%의 CPU 리소스를 사용하지 않는 문제, 스케줄링 우선순위가 더 높은 서비스 데몬의 동작, Disk I/O 수행으로 인한 CPU 사용률의 손상 등에 있었다[3,14,15].

그림 8은 기존의 시스템과의 정당한 비교 실험을 위하여 기존 시스템의 애플리케이션을 멀티 쓰레드화한 후의 제안 시스템과의 성능을 비교 실험한 결과이다. 싱글코어환경에서 기존의 시스템과 제안시스템은 동일하게 7.4초의 수행시간이 소요되었다. 듀얼코어 환경에서 제안하는 시스템은 4.3초의 수행시간 (멀티쓰레드화된 기존 시스템 대비 26%의 수행시간 단축)이 소요하였다. 쿼드코어환경에서는 우리가 제안하는 BioMP를 적용하였을 경우, 2.7초의 수행시간이 소요됨에 따라 멀티쓰레드화된 기존 시스템 대비 31%의 수행시간을 단축할 수 있음을 확인하였다. 모바일 디바이스는 배터리 소모를 최소화하기 위하여 CPU가 사용되지 않는 시점에 일시적으로 Offline 시키는 작업을 계속적으로 반복 수행한다. 실험결과에서 우리는 그림 8의 개선된 실행 속도 값이 우리가 제안하는 모바일 디바이스의 CPU 카운트 검색 메커니즘에 의해 기존 모바일 시스템의 멀티코어 인지 수행능력을 개선할 수 있음을 확인하였다.

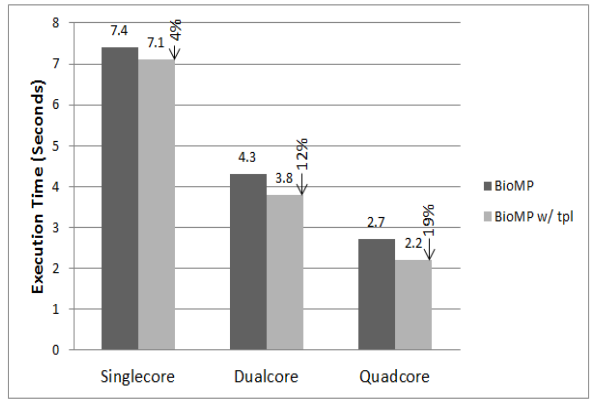


그림 9. 컴파일시점에 반복문 병렬화의 실험 결과
Fig. 9. Experimental result with parallelism of loop statement at compile time

그림 9는 BioMP가 적용된 시스템에서 컴파일 시점에 소스코드 내의 반복문을 병렬화하기 위한 컴파일러 옵션인 *-ftree-parallelize-loops=2* (BioMP w/ tpl)으로 실행하였을 때 얼마나 애플리케이션의 수행속도를 개선할 수 있는지 실험하였다. 실험 결과, *-ftree-parallelize-loops=2* 를

적용하였을 때 듀얼코어에서 12%, 쿼드코어에서 19%의 실행속도 개선 효과를 보았다. 또한 우리는 CPU의 코어가 1개인 싱글코어에서도 반복문의 병렬화 수행을 통해 4%의 성능 개선을 확인하였다. 싱글코어에서 4%의 성능 개선의 원인을 분석한 결과, 실제로 태스크가 수행하는 동안에 항상 100% CPU를 사용하지 않을 경우에 반복문의 쓰레드화가 효과적임을 알았다[16].

우리의 실험 결과는 멀티코어의 CPU들을 100% 모두 사용할 수 있는 멀티코어향 킬러 애플리케이션의 시나리오를 발굴이 또 하나의 숙제임을 보여주고 있다[3,17]. 또한, BioMP와 같은 기술들을 적용하여 애플리케이션 처리속도를 개선하는 것이 멀티코어 환경에서의 중요함을 확인하였다.

Ⅵ. 결 론

우리는 안드로이드 자바 애플리케이션에 포함되어 동작하는 컴포넌트방식의 태스크 병렬화 시스템인 BioMP를 소개하였다. 제안된 시스템은 안드로이드 플랫폼의 어떠한 추가적인 수정도 필요로 하지 않는다. 기존에 개발된 대형 프로젝트의 소스코드를 멀티코어 환경을 위해 다시 개발하는 것은 상당한 개발비용과 개발 시간이 필요하기 때문에 BioMP는 멀티코어환경에 최적화된 자바 애플리케이션을 빠르게 재생산할 수 있도록 돕는다.

제안 시스템은 기존 안드로이드 플랫폼의 수정 없이 실행이 가능하도록 함으로써, BioMP가 적용된 안드로이드 자바 애플리케이션을 플랫폼의 특성의 영향을 받지 않고 독립적으로 병렬화 수행이 가능하다. 구현한 결과물은 구글의 공식 안드로이드 오픈소스 프로젝트[6]에 통합되었기 때문에 누구나 자유롭게 무료로 다운로드하여 사용 가능하다.

References

[1] Y.-H. Chang, C.-B. Kuan, C.-Y. Lin, and T.-F. Su, “Support of software framework for embedded multi-core systems with Android environments,” in *Proc. 9th IEEE Symp. Embedded Syst. Real-Time Multimedia*, pp. 2-8, Oct. 2011.

[2] H. Yoon, “A study on the performance of Android platform,” *Int. J. Comput. Sci. Eng. (IJCSE)*, vol. 4, no. 4, pp. 532-537, Apr. 2012.

[3] M. Cho, S. J. Hwang, H. J. Lee, M. Kim, and S. W. Kim, “AndroScope for detailed performance study of the Android platform and its applications,” in *Proc. IEEE Int. Conf. Consumer Electron. (IEEE ICCE)*, pp. 408-409, Jan. 2012.

[4] G. Lim, et al, “BioMP: Migrating OpenMP into bionic library for Android platform based on ARM multicore,” *In Proc. KICS Winter Conf. 2013,* Yongpyoung, Korea, Jan. 2013.

[5] O. Cinar, *Pro Android C++ with the NDK (*Bionic API primer), Apress, Dec. 2012.

[6] Google, *Bionic: a Derivation of the BSD Standard C Library*, Retrieved Apr., 30, 2013, from https://android.googlesource.com.

[7] L. Dagum and R. Menon, “OpenMP: an industry standard API for shared-memory programming,” *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46-55, Jan. 1998.

[8] Y.-S. Lu, C.-H. Lee, H.-Y. Weng, and Y.-M. Huang, “Design and implementation of digital TV widget for Android on multi-core platform,” in *Proc. Int. Comput. Symp.*, pp. 576-580, Dec. 2010.

[9] A. Duran, J. M. Perez, E. Ayguadé, R. M. Badia, and J. Labarta, “Extending the OpenMP tasking model to allow dependent tasks,” *Lecture Notes in Computer Science*, vol. 5004, pp 111-122, May 2008.

[10] B. Mohr, A. D. Malony, S. Shende, and F. Wolf, “Design and prototype of a performance tool interface for OpenMP,” *J. Supercomput.*, vol. 23, no. 1, pp. 105-128, Aug. 2002.

[11] E. Ayguadé, N. Compty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, E. Su, P. Unnikrishnan, and G. Zhang, “A proposal for task parallelism in OpenMP,” *Lecture Notes in Computer Science*, vol. 4935, pp. 1-12, Jun. 2008.

[12] A. Duran, J. Corbalán, and E. Ayguadé, “Evaluation of OpenMP task scheduling strategies,” *Lecture Notes in Computer Science*, vol. 5004, pp. 100-110, May 2008.

[13] J. Bircsak, P. Craig, R. Crowell, Z.

Cvetanovic, J. Harris, C. A. Nelson, and C. D. Offner, "Extending OpenMP for NUMA machines," in *Proc. ACM/IEEE Conf. Supercomput. 2000,* no. 48, Nov. 2000.

[14] T. Hubbard, R. Lencevicius, E. Metz, and G. Raghavan, "Performance validation on multicore mobile devices," in *Proc. Lecture Notes in Computer Science*, vol. 4171, pp. 413-421, Oct. 2008.

[15] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *IEEE Comput.*, vol. 41, no. 7, pp. 33-38, Jul. 2008.

[16] I. Foster, "Task parallelism and high-performance languages," *IEEE Parallel Distributed Technol.: Syst. Technol. Archive*, vol. 2, no. 3, pp. 27-36, Sep. 1994.

[17] M. T. Heath and J. A. Etheridge, "Visualizing the performance of parallel programs," *IEEE Software*, vol. 8, no. 5, pp. 29-39, Sep. 1991.

임 근 식 (Geunsik Lim)

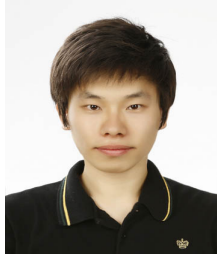
2003년 2월 아주대학교 컴퓨터 공학과 학사

2003년~현재 삼성전자 소프트웨어 센터 책임연구원

2012년~현재 성균관 대학교 정보통신대학 IT융합학과 석사과정

<관심분야> 임베디드 시스템, 모바일 플랫폼, 시스템 최적화 기술, 멀티코어, 스케쥴러

이 세 호 (Seho Lee)



2012년 2월 광운대학교 전자통신공학과 학사

2012년~현재 성균관 대학교 정보통신대학 임베디드 소프트웨어학과 석사과정

<관심분야> 메모리 관리, 저장장치, 클라우드, 가상화, 분산 시스템

엄 영 익 (Young Ik Eom)

1983년 2월 서울대학교 계산통계 학과 학사

1985년 2월 서울대학교 전산과학과 석사

1991년 2월 서울대학교 전산과학과 박사

1993년~현재 성균관대학교 정보통신대학 교수

<관심분야> 분산 컴퓨팅, 시스템 소프트웨어, 운영체제 가상화 기술, 미들웨어, 시큐리티 보안