한국산학기술학회논문지
Vol. 12, No. 1 pp. 486-492, 2011

안드로이드 달빅과 자바 가상머신의 성능비교

이종혁[1], 김형신[1*]
[1]충남대학교 컴퓨터공학과

Performance Comparison of Android Dalvik and Java Virtual Machines

Jonghyuk Lee[1] and Hyungshin Kim[1*]
[1]**Department of Computer Science and Engineering, Chungnam National University**

요 약 본 논문에서는 안드로이드에 탑재되는 가상머신인 달빅(Dalvik) 가상머신의 성능을 벤치마크를 구현하여 측정하고 기존 임베디드 자바 가상머신과의 성능을 비교한다. 달빅 가상머신의 성능 측정을 위하여 가상머신 성능 측정에 널리 사용되는 SPECJVM 벤치마크를 사용하였다. 달빅과의 성능 비교를 위하여 임베디드 자바 가상머신인 SUN Java 임베디드 버전을 비교 대상으로 선정하여 동일한 벤치마크를 사용하여 실행시간을 비교하였다. 성능 측정은 스마트폰 개발 하드웨어인 오드로이드(Odroid)에 가상머신과 벤치마크 프로그램들을 포팅하여 이루어 졌다. 또한, 달빅 가상머신의 최근 버전에서 구현되지 않은 적시(Just-In-Time) 컴파일 기능을 달빅에 추가하여 성능 개선 정도를 분석하였다. 분석 결과 안드로이드 달빅 가상머신은 SUN 가상머신 성능의 15% 수준을 보였으며, 적시 컴파일 기법을 적용한 달빅은 63% 수준에 도달함을 보였다.

Abstract In this paper we analyzed performance of Andriod's Davik virtual machine(VM) using standard benchmark and compared the result with the embedded Java virtual machine. We used a well known benchmark suit named SPECJVM for the measurement. For the fair comparison, Sun Java embedded JVM is ported and the same benchmark is ported on it. The Odriod smartphone hardware platform is used as the target hardware. We have added a Just-In-Time compiler to Dalvik, which is not supported in the recent Android release, and measured performance improvement. The experiment result show that Dalvik achieved 15% and Dalvik with JIT shows 63% of the Sun's JVM performance.

Key Words : Android, Dalvik, Virtual Machine, System Software

1. 서론

오늘날 임베디드 프로세서를 탑재한 스마트 폰은 사용자에게 편리하고 다양한 기능을 제공해 주기 위해서 GPS, 가속도 센서 등과 같은 다양한 센서들을 포함하기도 하며, 무선 인터넷 이외에도 근거리 무선 통신을 위한 블루투스와 같은 장치 등의 다양한 주변 장치들이 포함되기도 한다. 또한, 이러한 스마트 폰에 탑재되는 프로세서의 발달 속도는 점차 가속되어 점점 다양한 프로세서들이 스마트폰에 탑재되고 있다. 따라서 새로운 스마트폰이 시장에 출시되는 주기 역시 점점 짧아지고 있으며, 새롭게 출시되는 스마트폰 하드웨어 아키텍처에 따라 스마트폰에 탑재되는 소프트웨어의 개발이 이루어져야 하므로 소프트웨어의 개발 주기 역시 짧은 시간에 끝내야 하는 어려움이 생기게 되었다. 이러한 문제를 해결하기 위한 방법 중 하나는 가상머신(Virtual Machine)을 이용하는 방법이다. 가상머신의 경우 하드웨어 아키텍처에 의존적인 부분과 하드웨어 아키텍처에 독립적인 부분들이 나누어 설계되므로 하드웨어 아키텍처에 독립적인 부분들의 경우 쉽게 재사용이 가능하다는 장점이 있으며, 이

본 연구는 국토해양부 첨단도시기술개발사업의 지능형국토정보기술혁신사업과제(07 국토정보 C03)중 “실시간 공중자료획득 시스템 개발”과제의 일환으로 수행되었음.
*교신저자 : 김형신(hyungshin@cnu.ac.kr)
접수일 10년 12월 08일 수정일 11년 01월 03일 게재확정일 11년 01월 13일

러한 가상머신 위에서 동작하도록 작성되는 응용 프로그램의 경우 하드웨어에 상관없이 가상머신이 동작하는 모든 하드웨어에서 실행이 가능하다는 장점을 얻게 된다.

이러한 가상머신은 이식성이란 측면에서 장점을 가지고 있으나 내부적으로는 가상머신이라는 환경 상에서 바이트 코드들을 하드웨어에 의존적인 고유의 명령어 세트로 하나씩 변환하는 과정을 거치므로 가상머신 상에서 동작하지 않는 소프트웨어를 실행 시키는 것 보다 매우 느린 성능을 보인다는 단점을 가지고 있다. 이러한 가상머신의 단점은 특히 컴퓨팅 파워가 낮은 환경에서의 가상머신에 매우 치명적으로 작용하며, 따라서 이러한 단점을 극복하기 위한 다양한 연구들이 이루어지고 있다. 이러한 연구들로는 적시 (JIT :Just In Time) 컴파일 방법[1], DAC(Dynamic Adaptive Compilation) 컴파일 방법[2], AOT(Ahead of Time) 컴파일 방법[3]과 같은 소프트웨어 적인 방법에서 접근하여 바이트코드의 변환 시간을 최소화 하는 연구들이 있으며, Jazelle[4]과 같은 하드웨어적으로 제공되는 바이트코드 디코더를 이용하여 가상머신을 최적화한 연구가 제안되었다.

구글(Google)에서 발표한 안드로이드 (Android)[5][14]는 이러한 가상머신을 이용하여 응용 프로그램을 동작시키는 임베디드 운영체제이다. 특히 안드로이드에 탑재되는 가상머신인 달빅(Dalvik)[5]은 응용 프로그램의 작성을 위해서 자바 환경을 제공한다. 하지만 달빅 가상머신의 경우 아직 그 성능에 대한 평가가 제대로 이루어지지 않은 상태이며, 앞서 언급된 가상머신의 성능 향상을 위한 연구들도 많이 이루어지지 않은 상태이다. 특히 달빅 가상머신의 성능 향상을 위해서 적시 컴파일 방법과 같은 연구가 현재 이루어지고 있으나 그 성능 평가를 위한 도구들에 대한 연구는 매우 미미한 실정이다.

본 연구에서는 이러한 달빅 가상머신의 성능 평가를 위한 벤치마크 환경 구성 및 벤치마크를 작성하고, 구성된 벤치마크 환경에서 달빅 가상머신의 성능 분석을 수행한다. 이를 위해여 가상머신의 성능 평가를 위한 벤치마크의 선정 및 벤치마크의 포팅과정을 거쳐 벤치마크를 작성하였으며, 추가적인 성능 분석을 위하여 달빅 가상머신에서 제공되는 Traceview[5] 기능을 사용하여 그 결과를 성능 분석에 활용하였다. 또한 그 결과를 동일한 하드웨어 환경 및 동일한 리눅스 커널 환경에서 동작하는 SUN Java Embedded Edition[6]과 비교 분석 하여 달빅 가상머신의 성능 향상의 가능성에 대한 추가적인 분석을 수행하였다.

본 연구에서 벤치마크 환경의 구성에 사용된 안드로이드 버전은 2.1 Eclare로 리눅스 커널 2.6.29가 사용되었으며, 하드웨어 플랫폼으로는 Samsung S5PC100을 프로세서로 탑재한 오드로이드[7] 플랫폼을 이용하였다. 달빅 가상머신의 성능 평가를 위해서 사용된 벤치마크는 SPECJVM2008[8]을 기본으로 작성되었으며, 그 수행 결과 적시 컴파일 방법이 적용되지 않은 달빅 가상머신의 경우 SUN Java Embedded Edition에 비교하여 대부분의 벤치마크에서 평균 16%로 낮은 성능을 보여 주고 있으며, 적시 컴파일 방법이 적용된 달빅 가상머신의 경우 적시 컴파일 방법이 적용되지 않은 달빅 가상머신에 비해 평균 300%의 성능향상을 보였다.

본 논문의 구성은 다음과 같다. 2장에서 가상머신의 성능 개선에 대한 연구들 및 가상머신의 성능 분석에 대한 연구를 기술한다. 3장은 본 연구의 주제인 달빅 가상머신을 위한 벤치마크의 환경 및 벤치마크 구현에 대해 기술한다. 4장에서는 본 연구에서 구현된 벤치마크를 이용하여 실제 달빅 가상머신에 대한 성능 측정 결과 및 그에 대한 분석을 수행하고, 마지막 5장을 통해 결론을 도출한다.

2. 관련 연구

2.1 달빅 가상머신

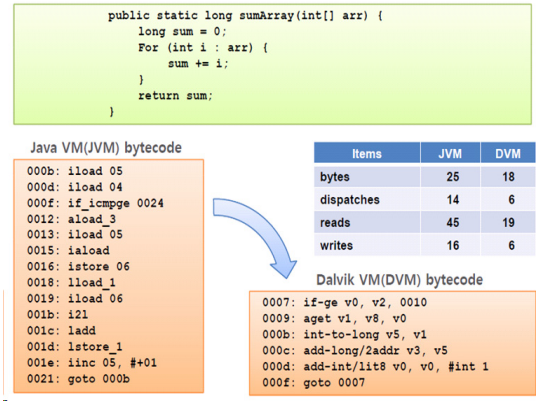
안드로이드는 구글에서 개발하여 2007년 발표한 개방형 모바일 플랫폼으로 단순히 모바일 단말기를 위한 운영체제만을 포함하는 것이 아니라 운영체제, 미들웨어, 기본 응용 프로그램 등의 소프트웨어 스택과 응용 프로그램의 작성 및 작성된 응용 프로그램의 검증을 위한 SDK를 포함하는 플랫폼이다. 안드로이드는 리눅스 커널을 기본으로 하여 리눅스 커널 위에 C/C++로 작성된 다양한 라이브러리들을 두고 그 위에 자바로 작성되는 어플리케이션 프레임워크와 어플리케이션들이 동작하는 구조로 구성되며, 특히 자바로 작성되는 코드들의 실행을 위해서 달빅 가상머신을 이용한다. 안드로이드에 탑재되는 달빅 가상머신은 C/C++로 작성되는 라이브러리들과 자바로 작성되는 응용 프로그램들과의 인터페이스를 담당하고 있다. 구글에서는 오픈소스로 제공될 안드로이드에서 SUN JVM을 사용할 경우에 발생할 수 있는 라이센스와 관련된 문제를 피하기 위해 새로운 가상머신을 필요로 하게 되었으며, 그에 따라 달빅 가상머신을 아래와 같은 규격의 임베디드 환경에서 동작이 가능하도록 설계 및 개발하였다[9].

- Slow CPU (250 500 MHz)
- RAM Usage : 20M~24M (system library : 10M)

- Little RAM : Available RAM : 20M
- Bus speed : 100MHz
- Data Cache : 16~32K
- No swap space, Battery power

위에서 언급된 동작 환경을 고려하기 위해서 달빅 가상머신은 스택기반의 가상머신인 SUN JVM과는 다르게 레지스터 기반의 가상머신으로 구현 되었으며, 메모리의 최적화를 위해여 달빅 가상머신 고유의 바이트코드를 정의하고 해당 바이트코드로 작성된 .dex포맷을 응용 프로그램의 파일 형식으로 이용한다. 또한 다양한 응용 프로그램의 실행 환경을 제공해주기 위해서 다수의 가상머신 인스턴스를 생성하고 관리하는 기능도 포함하고 있다. 이렇게 다양한 점을 고려하여 구현된 달빅 가상머신과 SUN JVM의 차이점은 아래의 그림 1에서 확인 할 수 있다.

그림 1의 Java VM(JVM) 바이트 코드와 달빅 VM(DVM) 바이트 코드는 각각 상단의 자바 코드를 컴파일 하여 바이트 코드로 변환한 결과이다. 변환된 두 종류의 바이트 코드들을 비교하여 보면 변환된 전체 바이트 코드의 크기가 JVM의 경우 25 Bytes 이며, DVM의 경우 18 Bytes로 달빅 가상머신의 경우 그 코드 크기가 JVM보다 작다. 따라서 달빅 가상머신의 경우 동일한 응용 프로그램의 실행 시에 차지하는 메모리 및 저장 공간을 더 적게 필요로 한다. 또, 코드 크기가 작은 만큼 바이트 코드 실행을 위한 코드 인출의 횟수도 줄어들게 되어, 달빅은 JVM보다 메모리에 접근하기 위한 읽기 및 쓰기 명령의 횟수가 훨씬 적어진다.



[그림 1] 바이트 코드의 비교[10]

구글은 달빅 가상머신을 이용하여 SUN과의 라이센스 문제를 회피하면서 가상머신의 장점을 안드로이드에 포함하여 운영체제로서의 장점을 확보할 수 있게 되었다. 하지만 달빅 가상머신의 경우 아직 그 실제 성능에 대한 분석이 많이 이루어지지 않고 있다. 특히 위에서 살펴본 바와 같이 SUN JVM과 비교하여 바이트 코드의 효율성 등에 대해서 강조되고 있으나 두 가상머신의 실제 성능의 차이나 비교 등에 대해서는 연구된 바가 없는 실정이며, 달빅 가상머신의 성능을 측정하기 위한 벤치마크에 대한 연구도 아직 이루어지지 않고 있다.

2.2 가상머신의 성능에 관한 연구

가상머신의 성능에 대한 연구들은 크게 적시 컴파일 방법, DAC(Dynamic Adaptive Compilation) 방법, AOT(Ahead of Time) 컴파일 방법 등을 다룬 연구들이 있다.

적시 컴파일 방법을 이용하면, 가상머신의 실행 엔진에서 바이트 코드의 디코딩 시 해당 디코딩된 바이트 코드들을 일종의 메모리 캐쉬에 베이직 블록과 같은 단위로 저장하게 된다. 이렇게 저장된 베이직 블록은 다음에 해당 베이직 블록이 다시 사용되어야 하는 경우 해당 베이직 블록에 해당하는 바이트 코드들을 다시 디코딩 하는 과정이 필요 없이 메모리 캐쉬에 저장된 디코딩된 바이트 코드들을 바로 실행이 가능하도록 해준다. 따라서 적시 컴파일 방법은 반복 수행되는 베이직 블록들의 수가 많을수록 디코딩 되어야 하는 바이트 코드의 수는 크게 줄어들게 되므로 가상머신의 성능은 크게 개선되는 결과를 가져오게 된다. 하지만 디코딩된 바이트 코드들을 저장하기 위한 별도의 메모리 캐쉬가 필요하다는 점에서 일반적인 인터프리터 방식 보다 더 많은 메모리 공간을 요구하게 된다는 단점이 존재한다.

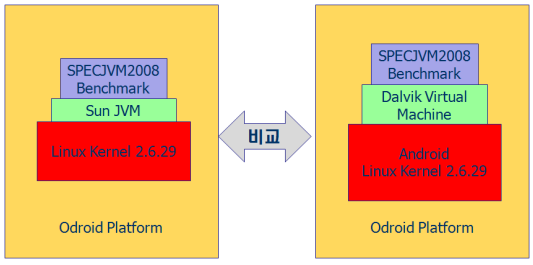
DAC(Dynamic Adaptive Compilation) 방법은 위에서 언급된 적시 컴파일 방법과 인터프리터 방법을 조합해서 사용하는 방법이다. 즉, 자주 실행되는 바이트 코드의 경우에는 적시 컴파일 방법을 사용하여 그 효율성을 높이고, 가끔 실행되는 바이트 코드의 경우에는 인터프리터 방식을 이용하여 적시 컴파일 방법을 사용할 때에 소모될 수 있는 메모리 공간들의 낭비를 최소화 하는 방법이다. 하지만 DAC 방법을 적용하기 위해서는 해당 응용 프로그램에서 자주 실행되는 부분과 자주 실행되지 않는 부분의 구분을 위한 프로파일링과 같은 별도의 처리 과정이 필요하다는 단점이 존재한다.

AOT(Ahead of Time) 컴파일 방법의 경우는 위에서 언급된 다른 방법들에 비해 그 아이디어가 간단하다. 자바 코드로 작성되는 응용 프로그램 중 자주 실행되는 부분들에 대해서는 단순히 바이트 코드를 생성하는 것에 그치지 않고 해당 부분들에 대해서 실제 실행 가능한 코

드, 즉 이미 디코딩된 바이트 코드로 해당 부분들을 미리 채워놓는 것이다. 따라서 자주 실행되는 부분들은 별도의 바이트 코드 디코딩 과정 없이 이미 디코딩된 바이트 코드로 기록된 코드들이 수행되며, 자주 실행되지 않는 코드들은 인터프리터 방식을 이용하여 실행하게 된다. AOT 컴파일 방법의 경우 응용 프로그램의 실행 성능은 빠르게 수행될 수 있으나 이미 응용 프로그램 내에 디코딩된 바이트 코드들이 이미 존재하고 있으므로 가상머신의 장점인 이식성이 떨어지게 된다는 단점이 생기게 된다.

지금까지 가상머신의 성능에 대한 연구들에 대해서 살펴보았다. 해당 연구들 각각에서는 제안된 방법을 적용한 가상머신에 대해서 해당 방법을 적용하지 않은 가상머신과의 차이를 통해서 제안된 방법의 효율성 및 특징을 비교하였다. 대부분의 연구들에서는 이를 위하여가상머신의 성능을 측정하는데 사용하는 벤치마크 툴들을 이용하였다[11]. 이들 연구에서는 자바 자상머신의 특징 및 구조를 기술하며 동시에 인터프리터 방식과 적시 컴파일 방법을 사용하였을 때의 성능 측정을 수행하고 성능 분석을 분기 예측 및 캐쉬의 성능 등과 연관지어 설명한다.

또한 해당 논문에서는 SPECJVM98이라는 가상머신의 성능 측정을 위한 벤치마크 셋을 이용하여 다양한 부분들에 대해서 가상머신의 성능을 체계적으로 측정하였다.



[그림 2] 성능 측정 방법

3. 달빅 가상머신의 성능 측정

앞서 살펴본 가상머신의 성능에 대한 다양한 연구들에서는 각각 방법들이 제시 되었으며, 해당 방법을 적용한 가상머신에 대한 성능 분석이 이루어 졌다. 따라서 달빅 가상머신에 대해서도 그 성능의 개선에 대한 연구가 이루어진다면 해당 연구가 적용된 달빅 가상머신에 대해서 성능 분석이 반드시 이루어져야 한다. 하지만 현재 달빅 가상머신에 대해 적시 컴파일 방법과 같은 성능 개선에 대한 연구가 진행 중[5]이나 그 성능의 분석에 대한 연구는 매우 미미한 실정이다.

안드로이드의 경우 안드로이드의 성능을 측정하기 위한 벤치마크로 Linpack Benchmark[12]나 BenchmarkPi[13]과 같은 응용 프로그램들이 존재하기는 하나 이러한 응용 프로그램들은 달빅 가상머신의 성능 분석을 위한 것이 아닌 단순한 CPU의 연산 성능과 같은 부분에 그 초점이 맞추어진 벤치마크들이다. 따라서 달빅 가상머신의 성능 분석에 대한 연구는 이루어져야 할 필요가 있으며, 본 장에서는 본 연구에서 수행한 달빅 가상머신의 성능을 분석하기 위한 벤치마크 환경의 구성 및 벤치마크의 작성에 대해서 논하고자 한다.

본 논문에서는 가장 먼저 달빅 가상머신의 성능 측정을 수행하고 그 결과로 얻어진 달빅 가상머신의 성능과 SUN의 자바 가상머신의 성능을 비교를 먼저 수행하고자 한다. 따라서 이를 위해서 동일한 하드웨어 상에 동일한 운영체제를 설치하고 각 운영체제 상에서 각 가상머신을 이용하여 벤치마크를 수행한 결과를 비교하고자 한다. 그림 2는 본 연구에서 사용한 성능 측정 방법을 도식화한 그림이다.

달빅 가상머신의 성능분석을 수행하기 전에 벤치마크 환경의 구성의 선행되어야 한다. 따라서 본 연구에서 구성하여 사용한 하드웨어 플랫폼, 운영체제 및 벤치마크 프로그램 등의 성능비교 환경에 대해서 먼저 소개한다.

[그림 3] 오드로이드

달빅 가상머신의 성능분석을 수행하기 위한 하드웨어 환경을 구성하기 위해 S5PC100기반의 프로세서를 탑재하고 있는 오드로이드 플랫폼을 이용하였다.

또한, 달빅 가상머신의 성능분석을 수행하기 위한 안드로이드 운영체제의 버전은 2.1 이클레어(Eclare)를 이용하였으며, 해당 버전은 리눅스 커널 2.6.29을 사용한다. 또한 해당 안드로이드 버전의 달빅 가상머신의 경우 적시 컴파일 기능이 완전히 동작하지 않으므로, 적시 컴파일 방법이 적용된 달빅 가상머신의 성능 분석을 위해서 해당 안드로이드 소스에 수정을 가하였다.

달빅 가상머신의 성능 측정을 위한 벤치마크로는

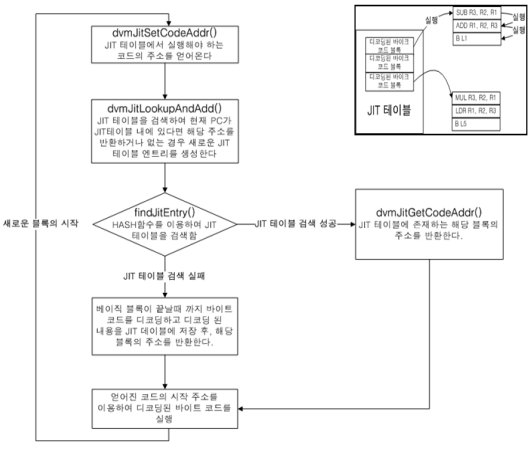
SPCEJVM 2008를 선택하였다. SPECJVM2008 벤치마크는 다양한 연구에서 JVM의 성능을 측정하기 위해 사용되는 대표적인 벤치마크 툴 셋으로 가상머신의 성능 측정을 위한 다양한 벤치마크 서브셋들을 포함하고 있으며, 각 벤치마크 셋들은 각 벤치마크 고유의 특성을 가지고 가상머신의 다양한 부분에서의 성능을 측정할 수 있도록 제공된다. 또한 SPECJVM2008 벤치마크는 단순히 하나의 타겟 플랫폼에만 국한되어 사용되는 것이 아닌 다양한 아키텍처 및 운영체제 상에서 사용가능하므로 가상머신과 관련된 여러 연구에서 인용되어 사용된다. 하지만 현재 달빅 가상머신을 위한 SPECJVM2008 벤치마크의 환경은 제공되지 않는 상태이므로 본 연구에서 SPECJVM2008 벤치마크를 사용하기 위해서는 별도의 작업을 수행하여야 할 필요가 있다. 따라서 본 연구에서는 SPECJVM2008 벤치마크를 달빅 가상머신 상에서 동작가능 하도록 포팅하는 작업을 수행하였으며, SUN JVM과 달빅 가상머신의 성능 차이를 관찰하기 위해서 동일한 하드웨어 플랫폼과 동일한 리눅스 커널 버전에 Java EE(Embedded Edition)을 올려 SPCEJVM2008을 이용하여 성능 측정도 함께 수행하였다.

3.1 벤치마크 작성

앞서 살펴본 SPECJVM2008 벤치마크는 달빅 가상머신 용으로 작성된 벤치마크가 아니므로 달빅 가상머신의 성능 측정에 대한 벤치마크를 이용하기 위해서는 달빅 가상머신에서 사용이 가능하도록 포팅하는 과정이 필요하다. 특히 여러 수정 사항 중 가장 큰 부분은 SPECJVM2008에서 기본으로 사용하고 있는 자바 라이브러리들이 SUN의 자바 라이브러리에 기반하고 있고, 실제 안드로이드 라이브러리의 경우 SUN의 자바 라이브러리와 호환이 되도록 작성이 되었으나 100% 호환이 되지 않으므로 호환이 되는 부분과 되지 않는 부분을 구분하여 수정해야 하는 문제였다. 따라서 이러한 문제를 해결하기 위해서 호환이 되지 않고 안드로이드에서 사용하지 않는 라이브러리들을 사용하는 벤치마크에 대해서는 해당 기능을 사용하지 않도록 수정하였으며, 추후 안드로이드 라이브러리에 해당 라이브러리들이 제공된다면 해당 기능을 사용할 수 있도록 하였다. 또한 안드로이드의 경우 가상머신에서 동작하는 응용 프로그램의 실행을 모니터링 및 디버깅 기능을 제공하기 위해서 Traceview라고 하는 기능을 제공하여 준다. 본 연구에서도 SPECJVM2008 벤치마크를 실행한 후에 각 벤치마크의 성능 분석하기 위해 Traceview기능을 사용할 수 있도록 해당 부분을 추가 구현하였다.

3.2 적시 컴파일 기능 적용

본 연구에서 달빅 가상머신의 성능 측정 및 분석을 위해서 사용하는 안드로이드는 Eclair 2.1버전이다. 해당 버전에 포함되어 있는 달빅 가상머신은 기본적으로 적시 컴파일 방법이 적용되어 있지 않으나 실제 해당 버전의 안드로이드 소스에 포함된 달빅 가상머신의 내부에는 적시 컴파일 방법이 대부분 구현되어 있는 상태이다. 본 연구에서는 적시 컴파일 방법이 적용되지 않은 달빅 가상머신과 적시 컴파일 방법이 적용된 달빅 가상머신 간의 성능 측정도 수행할 예정이므로 이를 위하여 사용할 안드로이드 달빅 가상머신의 적시 컴파일 방법을 적용하도록 이를 수정해야 한다.



[그림 4] 달빅 가상머신의 적시컴파일 구조

이를 위하여 가장 먼저 달빅 가상 머신에 적용된 적시 컴파일 방법의 구조를 분석하였다. 위의 그림 4는 분석된 달빅 가상머신 내부의 적시 컴파일 구조를 나타내고 있다. 대부분의 구조는 앞서 2장에서 살펴본 적시 컴파일 방법에서 사용되는 방식과 동일하나 적시 캐쉬 블록을 관리하기 위해서 해쉬 테이블을 이용하여 블록들을 관리하는 부분에서 차이를 보인다.

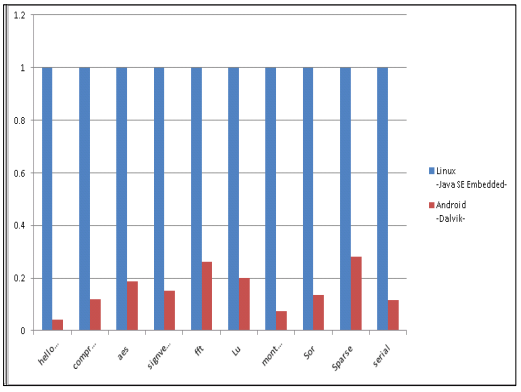
본 연구에서는 이러한 분석된 달빅 가상머신의 적시 컴파일 구조를 이용하여 달빅 가상머신에 적시 컴파일 방법이 적용되도록 가상 머신의 인터프리터 내에서 바이트 코드의 디코딩 시에 Table을 검사한 후 실행하는 기능이 동작하도록 수정하였다.

4. 성능 비교

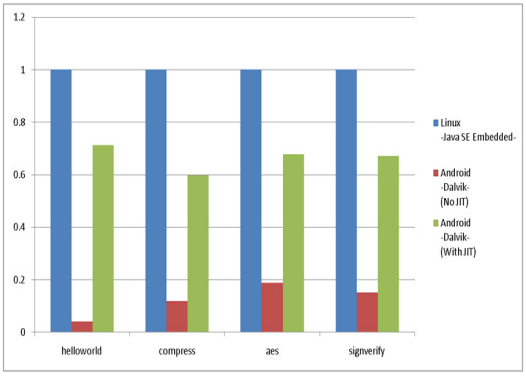
3장에서는 달빅 가상 머신을 위한 벤치마크 환경의 구

성 및 달빅 가상머신을 위한 벤치마크 작성, 달빅 가상머신에 JIT 컴파일 방법 적용 등에 대해서 살펴보았다. 본 장에서는 앞서 수행된 벤치마크의 결과를 설명한다.

그림 5는 달빅 가상머신의 각 벤치마크 프로그램에 대한 실행시간을 JVM 실행시간으로 나누어 비율을 표시한 그래프이다. 달빅 가상머신의 경우 그 성능이 SUN JVM의 성능과 비교할 때 평균 15%정도로 JIT 컴파일 방식이 적용된 SUN JVM과 해당 방법이 적용되지 않은 달빅 가상머신의 큰 성능의 차이를 확인 할 수 있다. 특히 여러 벤치마크 중 hello 벤치마크의 경우 그 차이가 다른 벤치마크보다 큰데 이 이유는 해당 벤치마크의 경우 다른 벤치마크들 보다 어떠한 데이터의 처리하기 보다는 특정 라이브러리의 단순한 호출이 빈번하게 이루어지는 형태이므로 코드의 지역성으로 인하여 JIT 컴파일 방법을 이용할 때와 그 차이가 크게 벌어진 것이라고 볼 수 있다.



[그림 5] 실행속도 비교



[그림 6] 적시 컴파일 적용 달빅의 성능 비교

그림 6은 JIT 컴파일 방법이 적용되지 않은 달빅 가상머신과 JIT 컴파일 방법이 적용된 달빅 가상머신에 대해서 벤치마크를 수행한 결과를 나타낸 그래프이다. 그림 6을 보면 JIT 컴파일 방법이 적용된 달빅 가상머신의 경우, JIT 컴파일 방법이 적용되지 않은 달빅 가상머신 보다 평균 300%정도의 성능 향상을 보이고 있는 것을 확인할 수 있다. 이것은 JIT 컴파일 방법을 적용 할 때의 얻을 수 있는 응용 프로그램 코드의 지역성으로 인항 성능 향상을 결과로 볼 수 있다. 특히 helloworld 벤치마크의 경우 그러한 원인으로 인하여 다른 벤치마크들 보다 적시 컴파일 방법을 적용하였을 때 얻을 수 있는 성능 향상의 폭이 더 큰 것을 확인 할 수 있다.

하지만 적시 컴파일 방법이 적용된 달빅 가상머신의 성능과 SUN JVM과의 성능을 비교하여 보면, 비록 적시 컴파일 방법이 적용된 달빅 가상머신이 적시 컴파일 방법이 적용되지 않은 달빅 가상머신보다 평균 300%의 성능 향상을 보였다고 하더라고 SUN의 JVM과 비교하였을 때는 평균 63%로 아직도 낮은 성능을 보인다는 것을 확인 할 수 있었다.이러한 원인은 SUN JVM과 달빅 가상머신의 Java Class 라이브러리의 차이에도 원인이 있으며, SUN JVM의 경우 적시 컴파일 방법의 사용 시 실제 물리 메모리의 가용한 공간에 따라 캐쉬를 할당하는 크기를 동적으로 조절하는 방법을 사용 하나 달빅 가상머신의 경우 초기 프로세스가 할당 받을 수 있는 메모리를 기본적으로 16MByte로 제한하고 실제 적시 컴파일 시 사용하게 되는 캐쉬의 크기를 크게 잡을 수 없으므로 그에 따른 차이를 보인 것이기도 하다. 따라서 위의 성능 측정 결과로 볼 때 달빅 가상머신에 적시 컴파일 방법이 적용되었다고 하더라도 SUN JVM의 성능에는 부족한 성능을 보이며, 아직 달빅 가상머신의 성능을 개선할 부분이 존재한다는 것을 확인 할 수 있다.

5. 결론

본 연구에서는 안드로이드의 가상머신인 달빅의 성능을 분석하고 이를 임베디드 자바 가상머신과 비교 및 분석을 수행하였다.

본 연구에서는 먼저 아무런 성능 개선을 위한 방법이 적용되지 않은 달빅 가상머신의 성능과 SUN JVM과 성능 비교를 수행하였으며, 그 결과 SUN JVM 성능의 평균 15%에 해당하는 낮은 성능을 보여 주는 것을 확인 하였다. 또한 이러한 달빅 가상머신에 적시 컴파일 방법을 적용하였을 때 적시 컴파일 방법이 적용되지 않은 달빅 가상머신 보다 평균 300%의 성능 향상을 보였다는 것을 확인 할 수 있었으며, 이러한 성능 향상에도 불구하고 SUN JVM의 성능과 적시 컴파일 방법이 적용된 달빅 가

상머신의 성능을 비교 할 때에는 평균 63%로 아직도 SUN JVM의 성능에는 못 미친다는 것을 확인 하였다.

참고문헌

[1] T Suganuma, T Ogasawara, M Takeuchi, "Overview of the IBM Java just-in-time compiler", IBM SYSTEMS JOURNAL, VOL 39, NO 1, 2000.

[2] Matthew Arnold, Stephen Fink, David Grove, Michael Hind, Peter F. Sweeney "Adaptive optimization in the Jalapeño JVM", OOPSLA, 2000.

[3] Joshua Auerbach, David F. Bacon, Bob Blainey, Perry Cheng, "Design and implementation of a comprehensive real-time java virtual machine", EMSOFT''07, September 30 October 3, 2007.

[4] http://www.arm.com/pdfs/JazelleDBX_WhitePaper_2007v1p1.pdf

[5] http://www.android.com/

[6] http://java.sun.com/

[7] http://www.hardkernel.org/

[8] http://www.spec.org/

[9] http://www.dalvikvm.com/

[10] http://www.kandroid.org/

[11] Ramesh Radhakrishnan, N. Vijaykrishnan, Lizy Kurian John, "Java Runtime Systems: Characterization and Architectural Implications", ACM SIGMETRICS, 2001.

[12] http://www.androlib.com/

[13] http://androidbenchmark.com/

[14] 정성화, 노태정, “SELinux 기반 안드로이드 보안시스템 구축에 관한 연구”, 한국산학기술학회논문지, Vol. 11, No. 8, pp.3005-3011, 2010.

이 종 혁(Jonghyuk Lee) **[준회원]**

- 2007년 2월 : 충남대학교 컴퓨터공학과 (학사)
- 2010년 8월 : 충남대학교 컴퓨터공학과 (석사)

<관심분야>
임베디드 시스템, 안드로이드, 운영체제

김 형 신(Hyungshin Kim) **[정회원]**

- 1990년 2월 : KAIST 전산학과 (학사)
- 1990년 12월 : 영국 Univ. of Surrey, 위성통신공학(석사)
- 2003년 2월 : KAIST 전산학(박사)
- 2004년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 부교수

<관심분야>
임베디드 시스템, 안드로이드, 운영체제