# 자바 가상 머신과 달빅 가상 머신의 성능 비교[1)]

김범준○, 정동헌, 문수묵
서울대학교 전기.컴퓨터공학부 가상 머신 및 최적화 연구실
pl@altair.snu.ac.kr, clamp@altair.snu.ac.kr, smoon@snu.ac.kr

# Performance Comparison of between Java Virtual Machine and Dalvik Virtual Machine

Beom-Jun,Kim○ Dong-Heon,Jung,Soo-Mook,Moon
Seoul National University, Department of Electrical Engineering and Computer Science,
Virtual Machine and Optimization Lab

기존의 자바 가상 머신은 스택 기반 가상 머신으로써 바이트코드 상의 최적화가 쉽지 않다. 하지만 달빅 가상 머신은 레지스터 기반 가상 머신으로 바이트코드 상에서 몇 가지의 최적화를 추가적으로 적용할 수 있기 때문에 자바 가상 머신에 비해서 성능면에서 더 좋은 효과를 볼 수 있을 것으로 기대된다. 이 논문에서는 잘 알려진 몇 개의 벤치마크를 통해서 실제로 자바 가상 머신의 인터프리터와 달빅 가상 머신의 인터프리터가 얼마나 성능 차이가 있는지 알아보고, 그에 대하여 분석하였다.

달빅에서는 자바의 수행을 위해서 기존 자바 가상 머신 바이트코드로 이루어진 클래스 파일을 달빅 가상 머신의 바이트코드인 덱스 파일로 바꾸게 되는데 이 과정에서 많은 성능 향상을 가져오게 된다. 결과적으로 달빅에서는 인터프리터 만의 성능을 놓고 봤을 때 많은 향상을 가져오게 되었다.

## 1. 서 론

근래의 구글의 안드로이드 운영체제[4]는 자바를 통해서 어플리케이션의 수행이 가능하도록 한 운영체제이다. 하지만, 안드로이드는 라이선스 등의 문제로 OS를 종속시키지 않기 위해서 기존의 자바 수행 환경인 자바 가상 머신을 사용하지 않고 독자적인 가상 머신인 달빅 가상 머신[5]을 개발하여 자바 수행 환경을 구성하였다.

이 논문에서는 자바 가상 머신에 비해서 달빅 가상 머신은 어떠한 성능 차이가 있으며 그 이유는 무엇인지 몇 가지 벤치마크의 성능 비교와 분석을 통해서 비교를 해보고자 하였다. 단, Sun의 자바 가상 머신에서 구현된 메소드 JIT 컴파일러와 달빅 가상 머신에서 구현된 Trace JIT 컴파일러 간의 비교는 하지 않고, 인터프리터 만을 비교 분석 하였다.

## 2. 본 론

달빅 가상 머신은 자바 바이트코드로 이루어진 클래스 파일을 수행하는 것이 아니라, 클래스 파일을 덱스 파일로 변환 시켜 주는 dx tool을 이용해 달빅 가상 머신의 바이트코드로 변환된 덱스 파일을 수행시킨다. 이 과정에서 dx는 바이트코드 상에서 몇 가지 최적화들을 적용한다. 이 최적화는 클래스 파일을 덱스 파일로 바꾸고 나서 추가적으로 적용하게 된다.

최적화를 적용하기 위해서 각각의 메소드에 있는 명령어들을 SSA Form으로 변환한 다음 최적화를 적용하고 다시 원래 Form으로 되돌리는 방식을 사용한다. 이렇게 SSA form[6]으로 만들고 나서, 크게 Dead Code를 없애는 최적화와 쓰이는 레지스터의 개수를 줄이는 최적화를 적용하게 된다.

이렇게 바이트코드 상의 최적화를 마치고 난 후의 성능을 보게 되면 적용전보다 평균적으로 61%의 성능이 좋아진 것을 확인할 수 있었다. 하지만 여전히 String은 성능 개선이 거의 되지 않았다. 이 이유는 dx 최적화의 루틴을 보면 invoke를 하는 과정에서 invoke시에 쓰일 argument들을 레지스터에 할당하고 invoke 직전 다시 이 레지스터에서 invoke에서 쓰일 레지스터로 move를 시키는 연산이 사라지는 것이 대부분임을 알 수 있는데, String의 경우는 constant pool에서 해당 스트링을 얻어오는 연산이 대부분이기 때문에 dx 최적화가 적용됨으로써 얻어지는 성능 이득이 거의 없었기 때문이라고 볼 수 있다.

1) 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업의 일환으로수행하였음. [KI002119, 고성능 가상머신 규격 및 기술 개발]

좀 더 정확한 성능 분석을 위해서 이 논문에서는 바이트코드 상의 최적화가 제거된, 따라서 순수한 가상 머신의 성능을 비교해 보았다.

가상 머신의 성능 만을 비교해본 결과에도 마찬가지로 달빅 가상머신의 성능이 평균적으로 약 1.4배 좋아진 것을 확인할 수 있었다. 이 이유는 가상 머신 상에서도 최적화를 수행하기 때문이다.

이러한 최적화는 일단 덱스 최적화로부터 시작된다. 덱스 파일을 처음으로 로딩할 때, 달빅 가상 머신에서는 이 덱스 파일 내의 메소드들에 대한 최적화를 수행하게 된다.

첫 번째로, 바이트코드 Quickening 이다. 몇 가지 바이트코드들은 정해진 인터프리터 루틴이 아닌 더 짧고 빠른 방법으로 수행이 가능하다. 달빅 가상 머신의 명령어의 특성상, 몇 개의 바이트코드는 달빅 constant pool에 있는 필드를 참조하게 된다. 하지만, 이렇게 constant pool을 다시 읽어서 필드를 가져오는 과정에서 현재 이 필드가 resolve 되어 있는지를 항상 확인하고 안되어 있으면 resolve하는 과정은 비효율적이다. 따라서 이런 바이트코드들의 필드를 미리 resolve하고 바이트코드 상의 constant pool 필드 레퍼런스 값을 resolve 되어 있는 필드의 현재 객체로부터의 오프셋 값으로 대체한다.

두 번째로, 자주 사용하는 15개의 메소드 들에 대해서 미리 머신 코드를 만들어 놓고 DexOpt 단계에서 이 메소드들을 호출하는 invoke 바이트코드가 있을 경우 이 머신 코드를 수행하도록 하는 바이트코드로 변경하는 것이다.

이렇게 바뀐 덱스 파일은 파일로 저장이 되기 때문에 해당 덱스 파일에 대해 이전에 DexOpt를 실행한 적이 있으면 다시 수행되지 않는다.

자바 가상 머신의 경우에 첫 번째 바이트코드의 quickening은 달빅 가상 머신과 마찬가지로 기존의 자바 바이트코드 몇 개가 바뀌어서 수행이 된다.[7,8]

하지만 자바의 바이트코드에는 달빅처럼 미리 머신 코드를 만들어 놓고 호출하는 바이트코드가 존재하지 않는다. 반면에 달빅에서는 기존의 Call overhead가 높은 invoke 바이트코드가 기존에 있던 네이티브 코드를 inline해서 수행하는 바이트코드로 변환이 되므로, 이 메소드들이 많이 불리는 프로그램의 경우에는 성능 향상이 그에 따라서 있게 된다. 하지만 실제로 수행 중에 CaffeineMark나 EEMBC 벤치마크에서 이런 메소드들을 호출하는 경우는 0.5% 정도로 그리 많지 않았으므로, 성능 향상에 많은 영향을 미치지는 못했다.

따라서 성능의 차이는 quickening 되어서 빠르게 수행되는 바이트코드의 비율일 것으로 생각하고, 전체적으로 quickening 되는 바이트코드의 비율을 세어본 결과, 달빅 가상 머신의 경우 전체의 약 15%의 바이트코드가 바뀌는 것을 확인할 수 있었다. 이는 자바 가상 머신에서 quickening 되는 숫자보다 약 5% 더 많았으므로, 이에 따라서 성능 차이가 나게 되었다.

## 3. 결 론

달빅 가상 머신의 바이트코드는 자바 가상 머신의 바이트코드로부터 기본적으로 일대일 대응이 되어서 변환이 되지만, 레지스터 기반 명령어의 특성상 이를 최적화할 수 있는 기회가 많이 생기게 된다. 따라서 달빅 가상 머신에서는 많은 수의 바이트코드들을 삭제할 수 있었으며 그에 따라서 높은 성능 이득을 보게 되었다.

이와는 별개로 가상 머신 자체의 성능 -같은 수의 바이트코드를 수행- 을 비교했을 때에도 달빅 가상 머신의 경우가 더 뛰어난 성능을 보였다.

이를 놓고 볼 때, 임베디드 환경에서 달빅 가상 머신은 자바 가상 머신에 비해서 괄목할 만한 성능 향상이 있었으며, 더불어 자바와는 달리 라이선스 문제를 어느 정도 해결하였기 때문에 앞으로 임베디드 환경에서 자바 가상 머신 대신에 달빅 가상 머신을 포팅해서 쓰는 경우가 많을 것이라고 보여진다.

## 참고 문헌

[1] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, The Java Language Specification, Second Edition, June,2000.

[2] Time Lindholm, Frank Yellin, The Java Virtual Machine Specification, Second Edition. April 1999.

[3] Jon Meyer, Troy Downing, JAVA Virtual Machine, First Edition, O'Relly, 1997.

[4] J. DiMarzio, Android: a programmer's guide, 2008

[5] D. Bornstein, Dalvik virtual machine, 2008

[6] S. Hack, D. Grund, G. Goos, Register allocation for programs in SSA-form, 2006.

[7] SungHyun Hong, Jin-Chul Kim, Soo-Mook Moon, Java client ahead-of-time compiler for embedded systems, July 2007.

[8] SangKyu Lee, Early Null Pointer Check Using Prediction in Java Just-In-Time Compilation, 2004.