달빅 가상 머신에서의 장기 추적 방식 적시 컴파일[1]

오형석* 문수묵
서울대학교 전기컴퓨터공학부
oracle@altair.snu.ac.kr , smoon@altair.snu.ac.kr

Longer Trace Based Just-in-Time Compiler for Dalvik Virtual Machine

Hyeong-Seok Oh, Soo-Mook Moon
Seoul National University Electrical Engineering

1. 서론

최근 모바일 환경은 스마트폰의 보급을 기점으로 크게 변화하고 있다. 스마트폰은 기존의 일반적인 휴대폰과 비교할 때 웹 접근성이 뛰어나고 사용자는 더욱 자유롭게 원하는 어플리케이션을 플랫폼이 제공하는 어플리케이션 스토어(application store)에서 구매하여 다운로드 받아서 설치할 수 있으며 어플리케이션 개발자는 호환성에 대한 우려 없이 어플리케이션을 개발할 수 있다. 이러한 스마트폰의 플랫폼으로 대표적인 구글의 안드로이드는 오픈소스로 플랫폼 소스코드에 누구나 접근 가능하여 개방향 플랫폼으로 주목을 받고 있다.

스마트 모바일 플랫폼의 어플리케이션 실행을 위해서 중요한 것이 사용자에게 안정적인 실행 환경을 제공하면서 어플리케이션 개발자에게 호환성의 우려 없는 쉬운 개발 환경을 제공하는 것이다. 안드로이드는 어플리케이션 개발 언어로 자바 언어를 사용하되 실행 환경으로 안드로이드를 위해 새롭게 개발된 달빅 가상머신 (Dalvik Virtual Machine)[1]을 사용한다. 사용자에게 쾌적한 실행 환경을 제공하는 것이 중요해 지면서 안드로이드에서는 어플리케이션 실행 속도의 향상을 위해 적시 컴파일(Just-in-Time compile) 기술[2]을 최근에 도입하였다. 하지만 여전히 하드웨어 자원 제약으로 인해 달빅 가상머신의 적시 컴파일러의 성능 향상이 필요하다.

2. 본론

달빅 가상머신은 자바 프로그램을 수행하기 위한 가상머신이다. 달빅 가상머신은 자바 프로그램을 하나의 덱스 파일(Dex file)로 실행파일을 생성하여 덱스 파일을 수행한다. 따라서 개발자는 자바 언어를 통해 동일한 코드를 작성하더라도 달빅 가상머신의 실행 파일은 자바 가상머신의 실행파일과는 다른 구조를 가지고 있으며 가상머신의 동작에도 차이가 있다.

덱스 파일의 실행 코드인 달빅 바이트코드는 레지스터 기반의 명령어로 기본적으로 2바이트의 크기를 가지고 있다. 하지만 레지스터 기반의 명령어는 스택에 대한 입출력(Push/Pop)연산이 줄어들고 이러한 바이트코드의 차이로 인해 달빅 가상머신의 해석기(interpreter)도 레지스터 기반으로 동작한다.[3]

달빅 가상 머신의 컴파일러는 해석기와는 별도의 쓰레드를 통해 수행되기 때문에 컴파일러의 수행이 해석기에 영향을 주지 않도록 되어 있다. 달빅 가상머신의 적시 컴파일러는 구조상 추적 기반의 경로 컴파일 정책[4]을 사용하고 있다. 우선 1) 해석기를 통해 수행하다가(interpret), 2) 반복적으로 사용되는 구간이라고 판단되면 추적을 시작하여 일정 구역을 해석기를 통해 수행 코드를 기록하면서 수행하고(recording), 3) 기록 종료조건을 만족하면 기록 결과를 바탕으로 머신 코드를 생성(compile)하는 단계를 거친다. 달빅 가상머신에서는 분기 명령어 직후나 컴파일된 코드의 마지막 수행 직후의 바이트코드부터 추적 시작 지점으로 하여 해당 지점을 자주 통과할 경우 추적을 시작 한다. 그리고 추적이 끝나면 컴파일 쓰레드의 작업큐(work queue)에 컴파일 대상을 넣어서 컴파일러가 처리하도록 한다. 경로(path) 컴파일은 자주 사용되는 경로를 잘 선택해야 하는데 추적 기반의 컴파일이 이를 용이하게 한다. 그러나 달빅 가상 조건 분기 명령어나 메소드 호출시에 추적을 중단하게 되어있어서 하나의 컴파일 단위가 매우 작은 수의 바이트코드로 이루어져 최적화에 한계가 있다.

장기 추적 방식 적시 컴파일은 이러한 단점을 극복하고자 한번의 추적시에 좀 더 많은 달빅 바이트코드에 대한 추적을 수행하는 것이다. 이를 위해 본 논문에서는 조건 분기 명령어를 만나더라도

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

추적을 계속하도록 하였다. 그리고 기록한 바이트코드의 크기가 일정 크기를 넘어서거나 메소드 호출시에 추적을 중단하도록 하였다. 이러한 추적을 수행할 경우 추적한 경로 내에 조건 분기 명령이 존재할 수 있게 된다. 기존에는 추적한 경로 외부로의 분기문이 경로의 끝에만 존재하지만, 조건 분기 명령을 포함하면 추적 경로가 아닌 다른 경로로 분기를 수행할 수 있는 코드(side exit)가 내부에 존재하게 되고, 이를 처리할 수 있도록 적시 컴파일러의 머신 코드 생성기를 수정하여 머신 코드 수행시에 정확한 수행이 이루어지도록 해야 한다.

3. 실험

본 논문의 실험은 ARMv7 기반의 Cortex-A8 1GHz CPU를 사용하는 board상에서 이루어졌다. 달빅 가상머신은 안드로이드 (Android) 2.2 Froyo을 바탕으로 구현을 추가하였다. 실험을 위한 벤치마크로 자바의 EEMBC benchmark를 달빅 가상머신과 함께 제공되는 변환기를 활용하여 덱스 파일로 변환하여 사용하였다. EEMBC 벤치마크 항목중 crypto는 변환 오류로 인해 사용하지 않았다. 성능 실험은 5번 연속실행시의 첫번째, 다섯번째, 5번 총 수행시간을 비교하였고, 그 이외의 실험은 5번 연속수행시 전체 수행동안의 결과를 바탕으로 하였다. 한번에 추적할 수 있는 명령어의 최대 길이는 64개로 제한하였다.

그림 1. 장기 추적시 수행시간 변화

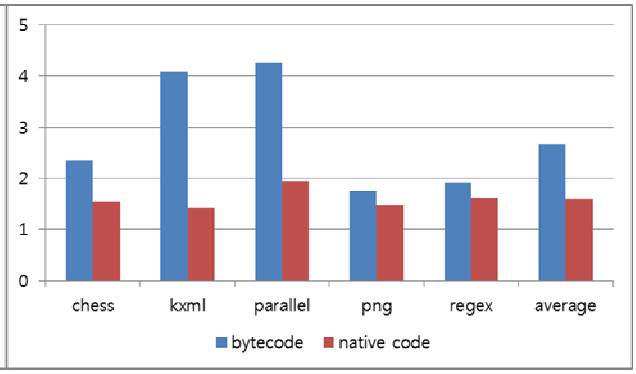

그림2. 컴파일된 바이트코드 및 기계어 코드

대부분의 항목에서 첫번째 수행시에는 장기 추적시에 수행 시간이 길어졌는데, 이는 추적과 컴파일에 필요한 오버헤드가 크기 때문으로 예상된다. 하지만 수행 시간이 지날수록 성능 격차가 줄어드는 것을 확인할 수 있다. 실제 컴파일 시간의 비중을 측정한 결과 장기 추적시에 컴파일 시간이 평균적으로 전체의 1.4%에서 2.5%로 증가하였다.

그리고 다음은 전체 컴파일된 바이트코드의 크기와 기계어 코드의 크기를 나타낸다. 바이트코드는 2.7배, 기계어 코드는 1.6배정도 커진 것을 확인할 수 있다. 이는 컴파일을 수행한 추적 경로의 개수는 거의 변하지 않았는데 단위 컴파일당 추적한 명령어의 개수의 증가로 인한 결과임을 실험을 통해 확인할 수 있다.

4. 결론

조건 분기 명령에서도 추적을 계속 하는 것을 통해 단위 추적당 명령어의 길이를 크게 한 결과 전체적으로 생성된 기계어 코드도 커지게 되었다. 한편으로 컴파일 단위가 커지면서 컴파일에 필요한 시간도 늘어나게 되었으나 성능에 미치는 영향이 크지는 않았다. 현재 구현에서는 컴파일 단위가 커진것에 따른 추가 최적화를 구현하지 않았으나 추가 최적화를 구현할 경우 좀 더 나은 성능을 얻을 수 있을 것이다. 또한 현재 구현에서는 메소드 호출시에 추적을 중단하였으나 메소드 호출 후에도 추적을 지속하는 것을 통해 좀 더 큰 컴파일 단위를 얻어낼 수 있을 것이다. 다만 지나치게 커질 경우 컴파일 시간에 영향을 미칠 수 있으며 특히 메모리 사용량이 증가할 수 있으므로 적절한 크기에서 추적을 중단할 수 있도록 해야 할 것이다.

참고문헌

[1] http://sites.google.com/site/io/dalvik-vm-internals
[2] J. Aycock. "A brief history of Just-in-Time", ACM Computing Surveys, 35(2), Jun 2003.
[3] Yunhe Shi, Kevin Casey, M. Anton Ertl, David Gregg. Virtual machine showdown: Stack versus registers, TACO, 2008.
[4] Andreas Gal, Christian W. Probst, Michael Franz. HotpathVM. An effective JIT compiler for resource-constrained devices.