pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.6.231

A Trace-based Precompile Method for Improving the Response Times of Android Applications

Sunggil Hong† · Kanghee Kim††

ABSTRACT

Recently, to improve the user response times of Android applications, several studies have been proposed to combine the idea of Ahead-of-Time compilation into Dalvik virtual machine, which uses Just-in-Time compilation. The studies, however, require modifications of the Dalvik executables of target applications, thus are difficult to be adopted for legacy applications already deployed. This paper proposes a JITwP(JIT with Precompile) technique that precompiles hot traces at application launch time with no modification of the Dalvik executable. It improves the user response times of target applications by providing precompile hints prepared offline. Our experimental results demonstrate a 4% improvement in terms of execution time for the Web browser application.

Keywords : Android, JIT, AOT, Precompile

안드로이드 응용의 응답 시간 향상을 위한 트레이스 기반 프리컴파일 기법

홍 성 길† · 김 강 희††

요 약

최근에 안드로이드 응용의 사용자 응답 시간을 향상시키기 위해 JIT 컴파일 방법을 사용하는 달빅 가상 머신에 AOT 컴파일 아이디어를 적용하는 방법들이 제안되었다. 그러나 기존 방법들은 DEX(Dalvik Executable)라는 달빅 실행 파일의 구조를 변경해야 하기 때문에, 이미 시장에 배포된 안드로이드 응용들을 대상으로 적용하기 어렵다. 본 논문은 DEX 실행 파일의 구조 변경 없이 달빅 가상 머신에서 핫 트레이스를 미리 컴파일하는 JITwP(JIT with Precompile) 기법을 제안한다. JITwP 기법은 달빅 가상 머신에 프리컴파일 힌트를 제공하여 타겟 응용의 응답 시간을 개선한다. 성능 평가 결과, 웹 브라우저 응용에 대해서 약 4%의 실행 시간을 단축하는 결과를 얻었다.

키워드 : Android, JIT, AOT, Precompile

1. 서 론

최근에 안드로이드 스마트폰이 확산됨에 따라, 안드로이드 응용의 응답 시간을 향상시키기 위해 달빅 가상 머신의 실행 방법을 개선하려는 시도들이 있었다[1-4]. 이들 연구는 달빅 가상 머신이 JIT(Just in Time) 컴파일 방법을 사용하기 때문에 JIT 컴파일 방법 자체의 효율을 향상시키거나[1], JIT 컴파일 방법에 AOT(Ahead of Time) 컴파일의 아이디어를 결합하거나[2, 3], 응용 별로 존재하는 JIT 코드 캐시들 간에 코드 공유를 허용하여 코드 캐시 효율을 높이는 기법[4] 등을 제안하고 있다. 그러나 사용자 응답성을 위해서는 응용의 초기 단계 실행 시간을 개선해야 한다는 점에서 JIT에 AOT의 아이디어를 결합하는 하이브리드 기법들에 대한 관심이 높다. 하지만 현재까지 제안된 하이브리드 방법들은 DEX(Dalvik Executable)라는 달빅 실행 파일의 구조를 변경해야 하므로, 타겟 응용의 개발자가 직접 시장에 배포한 패키지들을 직접 AOT 컴파일하여 마켓 서버에 업데이트하는 하는 노력이 요구된다. 이 방법들은 응용마다 프로파일링을 통해서 자주 실행되는 핫 메쏘드들을 찾아내고, 이 메쏘드들을 네이티브 코드로 변환하여 변환된 코드를 DEX 실행 파일에 통합한다. 또한, DEX 파일에 타겟 CPU에 맞추어 컴파일된 네이티브 코드가 포함되기 때문에, DEX 실행 파일의 하드웨어 독립성이 지켜지지 않는다.

※ 본 연구는 문화체육관광부 및 한국저작권위원회 저작권 기술 개발사업(2011-mobile_app-9500)의 지원과 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2010-0023522).
† 준 회 원 : 숭실대학교 정보통신공학과 석사과정
†† 비 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2013년 3월 26일
수 정 일 : 1차 2013년 5월 3일
심사완료 : 2013년 5월 6일
* Corresponding Author : Kanghee Kim(khkim@ssu.ac.kr)

본 논문은 DEX 실행 파일의 구조 변경 없이 달빅 가상 머신에서 핫 트레이스를 미리 컴파일하는 JITwP(JIT with Precompile) 기법을 제안한다. JITwP는 다음과 같이 3단계로 구현된다. 1단계는 프로파일링 단계로서 타겟 응용을 실행하여 JIT 컴파일의 대상인 트레이스 정보를 얻는다. 2단계는 프리컴파일 타겟 블록들을 선정하는 단계로서 트레이스 정보를 이용하여 트레이스마다 프리컴파일의 이득, 즉 utility 값을 계산한다. 이 utility 값이 큰 트레이스들이 프리컴파일 타겟 블록으로 선정되며 utility 값이 큰 순서대로 프리컴파일 힌트 파일에 저장한다. 3단계는 프리컴파일 실행 단계로서 달빅 가상 머신이 프리컴파일 힌트 파일에서 응용별로 선정된 프리컴파일 타겟 블록의 식별 정보를 읽어서 응용 시작 시 프리컴파일을 수행한다. 그 결과, 해당 응용은 프리컴파일이 없었다면 요구되었을 불필요한 인터프리팅 시간을 단축하게 된다. JITwP 기법은 DEX 파일의 구조를 변경하지 않기 때문에, 시장에 배포된 응용들을 업그레이드하지 않고 달빅 가상 머신을 업그레이드함으로써 특정 CPU에 종속되지 않고 독립적으로 적용할 수 있다는 장점이 있다. 이 때 프리컴파일 힌트 파일은 타겟 응용의 개발자가 아니더라도 누구나 생성할 수 있다. 본 논문은 JITwP 기법을 안드로이드 4.04 버전에 구현하여 브라우저 응용에 대해서 성능을 평가하였다. 실험 결과, 브라우저 응용에 대해서 약 4%의 실행 시간을 단축하는 결과를 얻었다.

본 논문은 다음과 같이 구성된다. 2절에서 관련 연구를 설명하고, 3절에서 JITwP 기법의 설계 및 구현 내용에 관해서 기술한다. 4절에서 성능 평가 결과를 제시하며, 5절에서 결론을 맺는다.

2. 관련 연구

최근에 JIT 기반의 달빅 가상 머신에 AOT 아이디어를 결합하는 하이브리드 기법들은 [2]와 [3]이 대표적이다. 이 기법들은 모바일 환경의 메모리 제약을 고려하여 전체 바이트코드가 아닌 일부 선정된 바이트코드에 대해서 AOT를 적용하며 나머지 코드에 대해서는 JIT 방식을 유지한다. [2]의 경우 프로파일링을 통해서 실행 횟수가 어떤 임계치를 초과하는 메쏘드들을 찾아내고, 이 메쏘드들을 오프라인에서 정적 컴파일하여 얻어진 네이티브 코드를 DEX 파일에 첨부한다. 그러면 이렇게 구조가 변경된 DEX 파일을 실행하기 위해 수정된 달빅 가상 머신에서 응용 시작 시 해당 파일을 적재하면, 프리컴파일된 메쏘드들은 적재 즉시 네이티브 코드로서 실행될 수 있다. 한편, [3]의 경우에는 정적 프로파일링을 통해서 핫 메쏘드들을 찾아내고 이 메쏘드들을 네이티브 코드로 변환하여 JNI(Java Native Interface) 라이브러리로 구성한 뒤, 이 라이브러리를 DEX 파일이 적재하여 실행하는 구조를 제안한다. Traceview[5]라는 도구를 이용하여 동적 실행 정보를 수집하고, 개발자가 정의한 메쏘드들만을 대상으로 실행빈도가 주어진 임계치보다 큰 메쏘드만 AOT 컴파일의 타겟으로 선정한다. 이렇게 찾아낸 핫 메쏘드들은 호스트 PC에서 COINS 컴파일러를 이용하여 C 코드로 변환되고, 변환된 C 코드를 다시 arm-linux-gcc를 이용하여 네이티브 코드로 변환한다. 그러나 앞서 언급했듯이 상기 연구들은 DEX 파일의 구조를 변경하기 때문에 이미 시장에 배포된 응용들에게 적용하기가 어려우며, 결과적으로 만들어진 DEX 파일이 타겟 CPU에 종속되는 문제점이 존재한다.

상기 하이브리드 방법들과는 다르게 JIT 컴파일 방법 자체를 개선하는 연구로는 [1, 6]이 있다. [1]은 트레이스 기반 JIT 방식(TJIT)에 메쏘드 기반 JIT 방식(MJIT)을 결합한 것으로, TJIT 기반의 달빅 가상 머신에 어떤 임계치 이상 호출된 메쏘드들을 메쏘드 단위로 컴파일하는 MJIT 기능을 추가한 것이다. MJIT로 컴파일된 메쏘드 안에는 TJIT의 컴파일 대상이 되는 코드들이 포함되는 경우가 많으므로 결과적으로 TJIT를 위한 프로파일링 오버헤드가 줄어드는 효과를 얻게 된다. 또한, MJIT는 메쏘드 단위로 컴파일하기 때문에 다양한 최적화 기법을 적용하여 품질 좋은 코드를 생성할 기회가 많다. 한편 [6]은 달빅 가상 머신이 아닌, 일반적인 자바 가상 머신을 위해 제안된 방법으로서 여러 단계의 임계치를 두어서 동일한 코드 블록에 대해서 처음에는 저품질 JIT 컴파일을 수행하고, 나중에 호출 횟수가 높아지면 고품질 JIT 컴파일을 다시 수행하는 방식이다. 고품질 JIT 컴파일을 위해서 저품질로 컴파일된 트레이스들을 연결하는 그래프를 구성하여 코드 최적화의 적용 범위를 확대한다. 이렇게 얻어진 고품질 네이티브 코드는 결과적으로 저품질 코드를 대체하여 사용된다. 그러나 JIT 컴파일 방법을 개선하는 연구들은 프로파일링 및 컴파일링 오버헤드를 더 요구하기 때문에, 응용에 따라서는 성능 이득이 상쇄되는 경우가 많다.

달빅 가상 머신이 응용마다 별도 코드 캐시를 갖기 때문에 이를 통합하는 전역적인 캐시를 제안하는 연구[4]도 있다. 달빅은 프로세스 단위 가상 머신이므로 응용들은 각기 다른 주소 공간을 사용한다. 이 때문에 자주 사용되는 클래스 라이브러리들은 응용들이 각기 JIT 컴파일해야 한다. 따라서 [4]는 각각의 달빅 가상 머신들이 컴파일한 코드들을 공유하는 전역적인 캐시를 사용하여, 한 응용이 이미 컴파일한 코드를 다른 응용이 재컴파일하는 일을 피한다. 그러나 이 방법은 응용 간에 공유되지 않는 응용 고유의 바이트코드 입장에서는 효과가 없으며, 응용들이 공유하는 시스템 라이브러리들에만 효과가 있다. 또한, 보안상 전역적인 캐시를 보호할 목적으로 별도 프로세스로 구현하기 때문에, 응용 입장에서는 전역 캐시에 접근할 때마다 프로세스 간 통신 오버헤드가 추가된다는 단점이 존재한다.

본 논문은 기존 DEX 파일의 구조를 변경하는 일 없이, 오프라인 코드 선정을 통해서 응용 별로 적합한 프리컴파일을 수행하는 방법을 제안한다. 이 방법은 일반적인 자바 가상 머신에서 JIT 컴파일 오버헤드를 줄이기 위해 오프라인에서 주석을 추가하는 방식[7]과 유사하다. 제안하는 방법은 안드로이드 단말기에서 달빅 가상 머신의 수정만을 요구하기 때문에, 실제 단말기에 적용이 용이하다.

3. 제안하는 기법

JITwP는 다음과 같이 3단계로 구현된다. 1단계는 프로파일링 단계로서 타겟 응용을 실행하여 JIT 컴파일의 대상인 트레이스 정보를 얻는다. 트레이스 정보를 얻기 위해 타겟 응용에 대해서 다양한 터치 입력을 제공하여 가능한 한 많은 수의 실행 경로들이 활성화되도록 유도하는데[8], 하나의 입력에 반응한 응용의 실행 인스턴스를 작업(job)이라고 부른다. 따라서 Fig. 1과 같이 각 작업에 대해서 달빅 가상 머신이 추적하는 트레이스 BBk마다 실행 횟수(EC), 인터프리팅 모드에서의 실행 시간(IE), JIT 컴파일된 이후의 네이티브 모드에서의 실행 시간(NE)을 얻어내어, 2단계의 입력으로 삼는다. 본 논문의 관찰에 의하면, 일반적으로 하나의 트레이스는 한두 개의 베이직 블록으로 구성된다. 2단계는 프리컴파일 타겟 블록들을 선정하는 단계로서 트레이스 정보를 이용하여 트레이스마다 프리컴파일의 이득, 즉 utility 값을 계산한다. utility 값은 네이티브 코드로 변환했을 때 실행시간 감소가 큰 트레이스들, 그리고 가능한 한 많은 수의 작업들에서 공통으로 사용하는 트레이스들이 큰 값을 갖게 된다. 이렇게 utility 값이 큰 트레이스들이 프리컴파일 타겟 블록으로 선정되며 utility 값이 큰 순서대로 프리컴파일 힌트 파일에 저장한다.

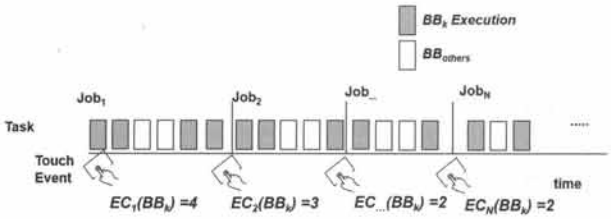

Fig. 1. Job-based target application profiling

3단계는 프리컴파일 실행 단계로서 달빅 가상 머신이 응용 별로 선정된 프리컴파일 타겟 블록의 식별 정보를 프리컴파일 힌트 파일에서 읽어서 응용 시작 시 프리컴파일을 수행한다. 그 결과, 해당 응용은 프리컴파일이 없었다면 요구되었을 불필요한 인터프리팅 시간을 단축하게 된다. 결과적으로 프리컴파일을 적용함으로써 얻게 되는 시간 이득은 다른 오버헤드를 고려하지 않을 때 각 트레이스에 대해서 {(인터프리팅 모드 실행시간 IE - 네이티브 모드 실행시간 NE) × JIT 컴파일 이전에 요구되는 인터프리팅 횟수}와 같다. 따라서 프리컴파일되는 트레이스의 개수가 늘어나면, 타겟 응용의 실행 시간 이득이 커지게 된다. Fig. 2는 JITwP 기법으로 얻을 수 있는 실행 이득을 표현한 것이다.

따라서, JITwP 기법을 구현하기 위해서는 달빅 가상 머신만을 수정하는 것으로 충분하다. 1단계를 위해서는 달빅 가상 머신에 베이직 블록 실행 과정에서 실행 시간을 모니터하는 기능을 추가해야 하며, 3단계를 위해서는 타겟 응용이 구동하기 전에 프리컴파일 타겟 블록들의 정보들을 프리컴파일 힌트 파일에서 읽어서 프리컴파일하는 기능을 구현

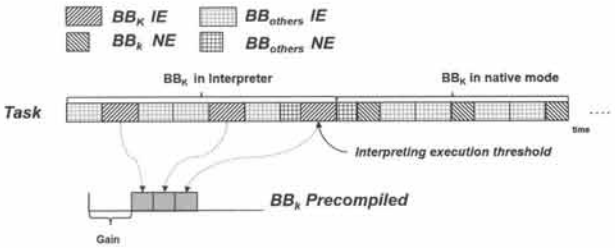

Fig. 2. Execution time gain of JITwP

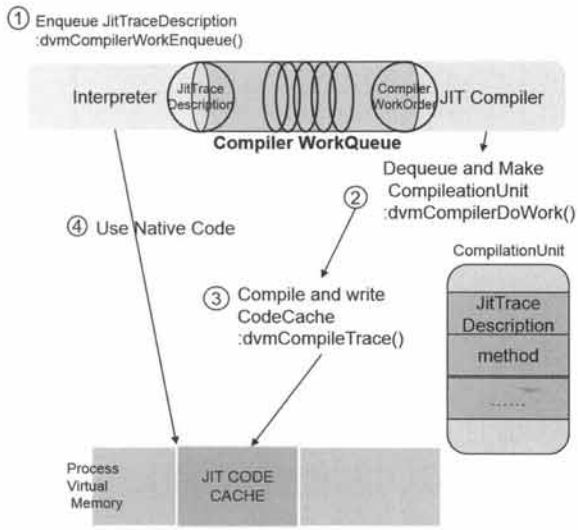

Fig. 3. JIT compilation step

하는 것이 필요하다. 2단계는 오프라인 코드 선정 단계이므로 달빅 가상 머신이 아닌 오프라인 분석 도구를 사용한다.

3.1 프로파일링 단계

프로파일링 단계에서는 다양한 입력을 대상으로 타겟 응용을 실행하여 JIT 컴파일의 대상 트레이스 정보를 얻는다. Fig. 3은 달빅 가상머신의 JIT 컴파일 과정을 나타낸다. 달빅 가상 머신은 정해진 횟수(안드로이드 4.04 버전의 ARM-v7-neon CPU인 경우에는 40회) 이상 인터프리팅된 베이직 블록들의 정보를 컴파일러 큐에 삽입한다(①). 컴파일러는 큐에 삽입된 베이직 블록들을 네이티브 코드로 컴파일한 후 이를 코드 캐시에 보관한다(②,③). 인터프리터는 인터프리팅할 베이직 블록이 이미 네이티브 코드로 변환되어 코드 캐시에 보관되어 있는 경우에 해당 네이티브 코드를 직접 실행한다(④). 본 논문은 과정 ①에서 각 트레이스에 대해서 식별 정보(클래스 이름, 메쏘드 이름, 오프셋), 작업별 실행 횟수(EC), 인터프리터 모드에서의 실행 시간(IE)을 기록한다. 각 베이직 블록의 마지막에는 분기문 혹은 함수 호출과 같은 점프 형태의 바이트 코드가 존재한다. 이러한 바이트 코드가 인터프리팅 모드에서 동작하면 반드시 베이직 블록의 실행 카운트를 업데이트하고 다음 실행할 베이직 블록이 네이티브 코드 캐시에 존재하는지를 확인하여 실행 모드를 결정하는 로직을 수행한다. 따라서 상기 로직에

서 각 베이직 블록의 인터프리터 모드 실행 시간(IE)과 네이티브 모드 실행시간(NE)을 측정하도록 달빅 가상 머신을 수정하였다. 이로써 프리컴파일 타켓 코드 선정을 위한 모든 정보가 확보된다. 프로파일링 단계에서 한 가지 유의할 점은, 타겟 응용 내부의 다양한 실행경로들을 활성화할 수 있도록 다양한 터치 입력을 가지고 프로파일링을 수행해야 한다는 것이다. 각 베이직 블록의 실행 시간(IE 또는 NE)들은 적어도 그 블록이 한번은 실행되어야 측정이 가능하다. 따라서 가능한 한 많은 실행 경로들을 활성화하여 대표적인 핫트레이스들을 확보할 수 있도록 터치 입력 시나리오를 다양화해야 한다. 터치 입력 시나리오의 다양화는 Monkey[11]와 같은 도구를 사용함으로써 가능하다.

3.2 프리컴파일 타겟 코드 선정 단계

프리컴파일 타겟 블록 선정 단계에서는 트레이스 정보를 이용하여 트레이스마다 프리컴파일의 이득, 즉 utility 값을 계산한다. utility의 정의는 Equation (1)과 같다. utility는 주어진 트레이스 BB_k에 대해서 인터프리터 모드 실행 시간을 네이티브 모드 실행 시간으로 나누고, 모든 작업 j에 대한 해당 트레이스의 공통 실행 횟수, 즉 최소 실행 횟수를 곱하는 것으로 정의된다. utility를 이렇게 정의하는 이유는, 첫째 네이티브 코드로 변환했을 때 실행시간 감소가 큰 트레이스들, 둘째 가능한 한 많은 수의 작업들에서 공통으로 사용하는 트레이스들이 실행 시간 측면에서 큰 이득을 가져다 줄 수 있기 때문이다. (예를 들어 Fig. 1에서 트레이스 BB_k의 공통 실행 횟수는 2이다.) 이렇게 utility 값이 큰 트레이스들이 프리컴파일 타겟 블록으로 선정된다.

$$Utility(BB_K) = \frac{IE(BB_K)}{NE(BB_K)} \times \min_j \{EC_j(BB_K)\} \quad (1)$$

3.3 프리컴파일 실행 단계

JITwP 실행 엔진은 프리컴파일 타겟으로 선정된 트레이스들을 해당 응용이 시작될 때 바로 프리컴파일 한다. JITwP의 실행 엔진은 dvmJITwP라는 함수로서 달빅 가상 머신 내부에 구현된다. 이 함수는 타겟 응용 생성 시, 달빅 가상 머신에 의해서 컴파일러 스레드가 생성되자마자 프로파일링 단계 없이 프리컴파일 타겟 블록 선정 단계에서 생성한 파일을 읽어서 Fig. 3의 함수 ③을 직접 호출하여 선정된 트레이스들을 프리컴파일한다. 컴파일된 모든 트레이스들은 코드 캐시에 저장되고 처음부터 네이티브 모드로 실행한다. 프리컴파일 기능의 구현을 위해서는 프로파일링 과정에서 문자열로 저장한 클래스 이름, 메소드 이름, 오프셋 정보를 가지고 달빅 가상 머신 내부의 연관된 객체 정보를 초기화하는 작업이 필요하다. dvmFindClass 함수에서 연관된 클래스 객체를 찾으면 클래스 객체 내에서 자식 메소드를 접근하여 메소드 객체를 초기화할 수 있다. 베이직 블록을 복원한 이후에는 lookupAndAdd 함수를 통해서 베이직 블록을 JItEntryTable에 등록한다. JItEntryTable은 JitEntry 구조체 배열로서 베이직 블록의 달빅 프로그램카운터(dPC)와 이에 대응하는 네이티브 코드의 주소를 저장하는 테이블이다. 그러면 최종적으로 등록된 모든 타겟 베이직 블록들을 컴파일러 스레드가 일괄적으로 프리컴파일을 수행하도록 달빅 가상 머신을 수정하였다.

4. 실험 결과

본 논문은 성능 평가를 위해서 ODROID-A 단말기[9]를 실험에 사용하였다. ODROID-A는 1.2GHz로 동작하는 듀얼 코어 S5PV310 CPU와 1GB RAM을 사용하며, 안드로이드 4.04와 리눅스 커널 3.0.15 버전을 사용한다. 실험에서는 순수한 JITwP의 성능 평가를 위해서 2개의 CPU 코어 중에서 하나만을 사용하여 성능 데이터를 얻었다. 2개 코어가 동시에 동작하면 프리컴파일 또는 JIT 컴파일을 위한 컴파일러 스레드가 병렬 수행되기 때문에 컴파일 오버헤드 측정이 어렵다. 타겟 응용으로는 안드로이드 4.04 버전에 기본으로 포함된 웹 브라우저를 사용하였다.

실험 과정은 프로파일링 단계, 프리컴파일 타겟 코드 선정 단계, 실행 단계로 나뉜다. 프로파일링 단계에서 웹 브라우저의 다양한 실행 경로들을 활성화할 수 있도록 텍스트, 이미지, 스크립트 등 다양한 컨텐츠 타입들로 구성된 웹 페이지를 순차적으로 방문하였다. 방문 순서는 www.google.com(홈페이지), www.daum.net(사이트 순회), www.naver.com(사이트 순회), www.cnn.com(사이트 순회) 순이다. 이 과정에서 각 트레이스마다 얻은 식별 정보, 인터프리터 모드 실행 시간(IE), 네이티브 모드 실행 시간(NE), 공통 실행 횟수(CEC)는 Fig. 4와 같다. 프리컴파일 타겟 코드 선정 단계에서 Equation (1)에 따라 계산한 각 트레이스의 utility 값 또한 Fig. 4에서 확인할 수 있다. 마지막으로 웹 브라우저 실행 단계에서는 공정한 성능 평가를 위해서, 프로파일링 단계에서 방문한 사이트들과 다르게 7개의 사이트들(cafe.daum.net, book.naver.com, www.danawa.com, cafe.naver.com, publisher.naver.com, www. daum.net, www.facebook.com)의 홈페이지를 방문하여 웹 브라우저의 실행 시간을 측정하였다.

네트워크 통신 지연 시간이 웹 브라우저의 실행 시간에 영향을 주지 않도록, 상기 7개 사이트의 웹 페이지들을 단

class	method	offset	IE	NE	CEC	Utility
Ljava/nio/charset/ModifiedUtf8	countBytes	51	10	2	28	11.2
Landroid/text/SpannableStringBuilder	getSpans	121	9	2	12	10.8
Ljava/nio/charset/ModifiedUtf8	encode	8	8	2	24	7.2
Landroid/text/StaticLayout	generate	862	10	2	18	6.6
Landroid/webkit/BrowserFrame	firstLayoutDone	0	4	0	16	6
Ljava/nio/charset/ModifiedUtf8	countBytes	24	8	2	21	5.76
Ljava/nio/charset/ModifiedUtf8	countBytes	9	8	2	23	5.76
Ljava/nio/charset/ModifiedUtf8	encode	18	12	3	22	5.76
Landroid/view/ViewGroup	invalidateChild	378	11	1	6	5.72
Landroid/text/BoringLayout	isBoring	44	4	2	20	5.6
Landroid/view/ViewGroup	dispatchTouchEvent	56	20	1	1	5.6
Landroid/text/method/ReplacementTra	getChars	20	5	2	109	5.5
Ljava/math/BigInteger	twosComplement	137	5	1	27	5.4
Ljava/nio/charset/ModifiedUtf8	countBytes	7	6	2	22	5.4
Ljava/lang/String	hashCode	20	5	2	29	5.1

Fig. 4. Example of utility calculation

말기 내부에 미리 다운로드 하였으며, 실험의 자동화를 위하여 각 사이트의 메인 페이지를 순서대로 적재하되, 한 페이지의 렌더링이 완전히 종료해야 다음 페이지를 적재하도록 하였다. 웹 브라우저 실행 시 프리컴파일하는 트레이스들의 개수에 따른 성능 영향을 파악하기 위해서, 트레이스들의 개수는 utility를 기준으로 100개씩 늘려가면서 실험하였다. 각 실험에 대해서 웹 브라우저는 30회 실험하여 실행 시간 평균을 측정하였다.

Fig. 5는 실험 결과를 보여준다. 회색 바는 컴파일러 스레드의 프리컴파일 시간 오버헤드를 포함한 것이고, 흰색 바는 포함하지 않은 것이다. 상기 그림에서 프리컴파일할 트레이스의 수가 100개씩 늘어날수록 실행 시간이 지속적으로 감소함을 확인할 수 있다. 이는 웹 브라우저가 최초 실행할 때부터 네이티브 모드로 실행되는 코드 블록들이 늘어나기 때문이다. 최대 실행시간 이득은 프리컴파일이 전혀 없을 때의 전체 실행 시간 11.6sec와 비교하여 700개의 트레이스를 프리컴파일한 경우로서 670msec, 즉 5.9%이다. 이 값은 컴파일러 스레드가 프리컴파일을 위해 소비한 시간을 포함한 값이며 이를 제외한 순수 실행 시간 이득은 440msec, 즉 3.9%이다. 참고로, 이 프리컴파일 오버헤드는 멀티코어의 경우, 컴파일러 스레드가 타겟 응용의 사용자 스레드들과 병렬 실행되므로 거의 관찰되지 않을 것으로 예상할 수 있다.

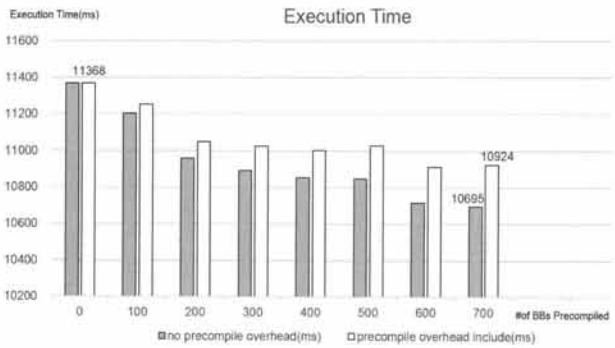

Fig. 5. Execution times of the web browser

5. 결 론

본 논문은 트레이스 기반 프리컴파일 기법으로서 JITwP(JIT with Precompile)을 제안하고 성능을 평가하였다. JITwP 기법의 장점은 기존에 시장에 배포된 응용들의 업그레이드 없이 달빅 가상 머신만을 업그레이드함으로써 적용이 가능하다는 것이다. JITwP는 JIT 컴파일시 런타임에 발견되는 핫 트레이스들의 정보를 수집하고 오프라인에서 성능 이득이 큰 타겟 블록들을 미리 분석하여 응용 시작 시점에 핫 트레이스에 해당하는 트레이스들을 프리컴파일한다. 이를 통해 JIT 컴파일시 발생하는 프로파일링 오버헤드 없이 응용의 사용자 응답 시간을 개선할 수 있다. 본 논문은 JITwP를 안드로이드 4.04에서 구현하고 웹 브라우저 응용에 대해서 성능 평가하였다. 성능 평가 결과 700개의 트레이스를 프리컴파일 시 3.9%의 최대 실행 시간 이득을 얻었다.

향후에 다양한 응용들에 대해서 성능 평가를 확대하고 프리컴파일 성능 이득을 극대화할 수 있는 프리컴파일 타겟 코드 선정 방법에 대해서 추가 연구를 진행할 계획이다. 모든 응용들이 공통적으로 사용하는 시스템 클래스 라이브러리의 경우, 프리컴파일 타겟 코드 블록들을 공유하는 방법 또한 연구할 계획이다.

참 고 문 헌

[1] Guillermo A. Perez, Chung-Min Kao, Yeh-Ching Chung, Wei-Chung Hsu, "A hybrid just-in-time compiler for android: comparing jit types and the result of cooperation," in Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems, Finland, 2012, pp.41-50.

[2] Yeong-Kyu Lim, Parambil, S., Cheong-Ghil Kim, See-Hyung Lee, "A selective ahead-of-Time compiler on android device," in Proceedings of the Information Science and Applications (ICISA), 2012 International Conference, Brazil, 2012, pp.1-6.

[3] Chih-Sheng Wang, Perez, G.A., Yeh-Ching Chung, Wei-Chung Hsu, Wei-Kuan Shih, Hong-Rong Hsu, "A method-based ahead-of-time compiler for android applications," in Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems, Taiwan, 2011, pp.15-24.

[4] Yao-Chih Huang, Yu-Sheng Chen, Wuu Yang, Shann, J.J.-J., "File-based sharing for dynamically compiled code on dalvik virtual machine," in Proceedings of the Computer Symposium (ICS), 2010 International ,Taiwan, 2010, pp.489-494.

[5] Google Traceview profiling tool. [Internet] http://developer.android.com/

[6] Hiroshi Inoue, Hiroshige Hayashizaki, Peng Wu, Toshio Nakatani, "Adaptive multi-level compilation in a trace-based java jit compiler", in Proceedings of the OOPSLA '12 in Proceedings of the ACM international conference on Object oriented programming systems languages and applications, US, 2012, pp.179-194.

[7] Chandra Krintz, Brad Calder "Using annotations to reduce dynamic optimization time", in Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation, U.S, 2001, pp.156-167.

[8] Duhee Lee, Chang-Gun Lee, Kanghee Kim, Eun Yong Ha, "Real-time program execution on nand flash memory for portable media players", in Proceedings of the Real-Time Systems Symposium, Spain, 2008, pp244-255.

[9] Hardkernel Inc. ODROID-A. [Internet] http://www. hard kernel. com

[10] C. A. Lattner(2002). LLVM: An infrastructure for multi-stage optimization. Technical report. [Internet] http://llvm.org/pubs/2002-12-LattnerMSThesis. html

[11] Monkey [Internet] http://developer.android.com/tools/help/monkey.html

홍 성 길

e-mail : tbeyond@naver.com

2012년 숭실대학교 정보통신전자공학부 (학사)

2012년~현 재 숭실대학교 정보통신공학과 석사과정

관심분야 : 실시간 임베디드 시스템

김 강 희

e-mail : khkim@ssu.ac.kr

1996년 서울대학교 컴퓨터공학과(학사)

1998년 서울대학교 전기컴퓨터공학부 (석사)

2004년 서울대학교 전기컴퓨터공학부 (박사)

2004년 삼성전자 무선사업부 책임연구원

2010년~현 재 숭실대학교 정보통신전자공학부 교수

관심분야 : 실시간 임베디드 시스템, 운영체제, 모바일 플랫폼, 모션제어