레지스터 사용해제 지연을 통한 바이너리 변환 성능향상

최 민*

*충북대학교 정보통신공학부

Faster Binary Translation by Delayed Deallocation of Temporary Registers

Min Choi*

*Chungbuk National University

E-mail : mchoi@cbnu.ac.kr

요 약

최근 각종 스마트 디바이스의 활용이 급속히 활용이 늘어나고 있다. 본 연구는 바이너리 변환(Binary Translation) 기술에 있어 레지스터 할당 관점에서의 성능향상 기법을 소개한다. 사용이 끝난 임시 레지스터의 재사용을 위해서 자원의 사용해제를 지연한다. 따라서, 다음 명령어가 직전에 사용된 임시 레지스터의 재사용을 통해, 자원을 해제하고 재할당하는 오버헤드 없이 바이너리 변환의 성능향상을 꾀할 수 있도록 한다.

ABSTRACT

In this paper, we introduce a technique for delayed deallocation of temporary register allocation. We achieve faster binary translation that is used in the context of register allocation. By delaying deallocation of the temporary register containing the value, it is preserved until the next instruction fetched. If subsequent instruction does not require the value again, binary translator deallocate and release the temporary register at the first stage of the next instruction.

키워드

Faster binary translation, Delaying register deallocation

Ⅰ. 서 론

현대 스마트폰에서 안드로이드폰, 아이폰, 윈도우폰 애플리케이션은 ARM 명령어 세트(instruction set)에 기반한 프로세서에서 동작하는 경우가 대부분이다. ARM 아키텍쳐는 물론 저전력성, 고성능, 고집적 패키징 등 여러가지 면에서 장점을 갖고 있어 현대 임베디드 시스템, SoC 분야에서 매우 널리 활용되고 있다. 현재 ARM 기반 프로세서가 대부분의 스마트폰 시장을 점유하고 있는데 반하여, 향후 x86 기반 프로세서가 탑재된 저렴한 스마트폰이 출시될 수 있는 가능성을 제시한다. 이러한 환경에서는 안드로이드 운영체제가 ARM 계열이 아닌 x86 계열 모바일 프로세서에서 운용하는 경우를 가정해야 하는데, 이때, 애플리케이션 호환성 및 성능 문제가 발생하므로 이를 해결하기 위한 바이너리 변환 기법이 제공되어야 한다. 일반적인 안드로이드 애플리케이션은 대부분의 자바 바이트코드(JAVA bytecode)로 compile되어 있으며 JAVA 기반의 Dalvik-VM에서 동작하므로, x86기반 안드로이드 운영체제에서도 문제없이 수행 가능하다, 그러나, 현재 앱스토어에 등록되어 있는 일부 애플리케이션들은 JAVA Bytecode 외에 추가적으로 NDK를 통하여 컴파일된 코드를 포함하고 있는데, 이 부분이 ARM architecture를 target으로 한다.

Ⅱ. 본 론

임시 레지스터 사용 후 해제 지연을 통한 바이너리 변환 성능개선을 제안하였다. 본 논문에서는 레지스터 할당/해제 (register allocation/deallocation) 관점에서 바이너리 변환의 성능을 개선하는 것이다. 즉, 본 연구에서 구현한 바이너리 변환기는 레지스터를 사용한 후

이 레지스터를 사용을 마친 것으로 flag를 업데이트 하지 않으며, 결과적으로 지연된 자원 해제(delayed deallocation) 기법을 동작시킨다. 여기서 임시 레지스터 자원을 활용한다는 뜻은 바이너리 변환 과정에서 논리 레지스터(logical register)/아키텍쳐 레지스터(architectural register)를 프로그램 내부에서 다루는 가상의 임시 레지스터(temporary register)로 대응시키고 실행하는 데, 이에 대한 자원의 사용을 뜻한다. 이러한 임시 레지스터가 사용을 마치고 정상적으로 사용이 해제되어야만, 다음번 명령어를 위해서 그 임시 레지스터가 사용될 수 있다.

그런데, 본 논문에서는 이러한 임시 레지스터가 여전히 의미있는 값을 가지고 있을 가능성에 초점을 두고, 임시 레지스터의 사용해제를 지연시키는 기법을 적용하였다. 이러한 판단의 근거는 연속되는 명령어 간에는 서로 데이터 의존관계(data dependency)가 있을 확률이 크기 때문이다. 기본블럭(basic block)내에 평균적으로 얼마나 많은 명령어 의존관계(instruction dependency)가 많이 존재한다. 이와 같이, 연속된 명령어를 실행하는 데 있어, 앞 명령에서 생성한 데이터 값을 뒷 명령어에 짧은 시간 내에 다시 그 데이터를 필요로하는 상황이 빈번하게 발생할 수 있다. 따라서, 본 연구에서는 이러한 특성을 활용하여, 바이너리 변환에 있어, 앞서 생성한 명령어에 대한 임시 레지스터 자원을 다음 명령어의 실행초기까지 지연시킨다.

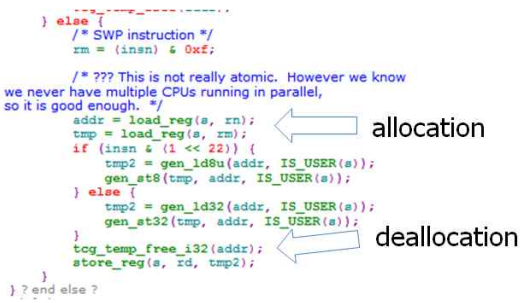

그림 1 임시레지스터 자원할당/해제 예시1

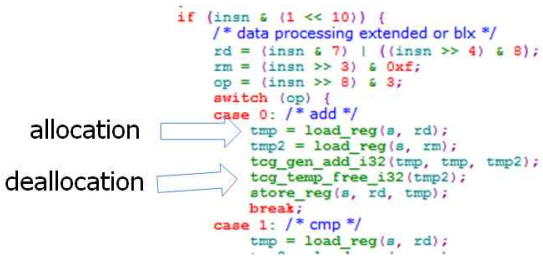

그림 2 임시레지스터 자원할당/해제 예시2

이를 통해, 만약 다음 명령어가 해당 데이터를 필요로 하는 경우, 데이터를 다시 논리 레지스터(logical register) 또는 아키텍쳐 레지트서(architectural register)로부터 읽어올 필요 없이 임시 레지스터로에 저장되어 있는 값을 다시 활용하면 된다. 만약, 다음 명령어가 해당 데이터를 필요로 하지 않는 경우라면, 명령어 인출(fetch)단계가 처리된 직후, 임시 레지스터의 자원할당을 취소하면 된다. 위 그림 1과 2는 앞에서 설명한 바와 같은 레지스터 사용해제 지연을 통한 방법의 효과를 그림으로 도시한 것이다.

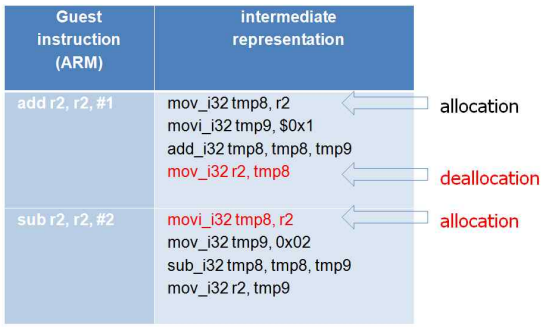

그림 3 임시레지스터 해제 직후 재할당 예시

그림 3은 임시레지스터가 해제된 직후 다시 재할당되는 사례를 보여준다. 이러한 사례는 임시레지스터의 해제와 재할당이 불필요함을 매우 직관적으로 나타낸다. 사실, ARM guest instruction 관점에서 보았을 때에는 이러한 불필요한 자원의 할당/해제가 발생하지 않는다. 그러나, 바이너리 변환 과정에서 매 ARM guest instruction에 대해서 각각 중간표현(intermediate representation)으로 변환을 하다 보니, 이러한 사례가 발생하는 것이다.

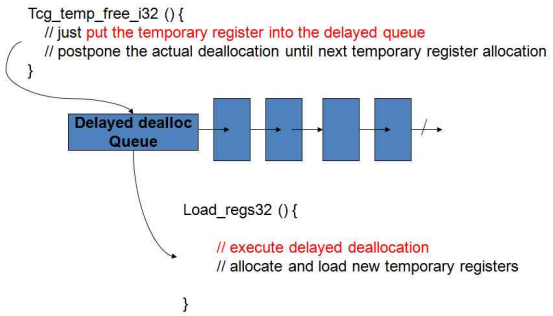

그림 4 레지스터 지연해제를 구현하기 위한 queue 구조

그림 4는 레지스터 지연해제를 구현하기 위한 queue 구조이다. 직전에 사용된 레지스터에 대한 정보를 queue에 넣어두고, 다음 명령어 실행 초반(명령어 인출 단계)에서 사용되지 않은 경우, 이 정보를 바탕으로 임시 레지스터의 자원을 해제한다.

III. 결 론

본 논문에서는 임시 레지스터 사용 후 해제 지

연을 통한 바이너리 변환 성능개선을 제안하였다. 본 논문에서는 레지스터 할당/해제 (register allocation/deallocation) 관점에서 바이너리 변환의 성능을 개선하였다. 즉, 본 연구에서 구현한 바이너리 변환기는 레지스터를 사용한 후 이 레지스터를 사용을 마친 것으로 상태를 업데이트 하지 않으며, 결과적으로 지연된 자원 해제 (delayed deallocation) 기법을 동작시킨다.

참고문헌

[1] "Magic Code: run ARM apps on MIPs devices", http://www.mipsdroid.com/2012/03/08/magic-code-run-arm-apps-on-mips-devices/, March 8, 2012.

[2] S. Bansal and A. Aiken, "Binary Translation Using Peephole Superoptimizers", 8th USENIX OSDI, Mar. 2009.