컴파일 된 시뮬레이션 기법을 이용한 ASIP 시뮬레이터의 성능향상

김호영, 김탁곤

{hykim@smslab.kaist.ac.kr tkim@ee.kaist.ac.kr}

한국과학기술원 전자전산학과 전기 및 전자공학전공

대전광역시 유성구 구성동 373-1

Performance Improvement of ASIP Simulator Using Compiled Simulation Technique

Ho Young Kim, Tag Gon Kim

Department of EECS, Division of Electrical Engineering, KAIST

요 약

이 논문은 빠른 ASIP(application specific instruction processor) 시뮬레이션을 위한 재적응성을 가진 컴파일드 시뮬레이션 기법에 대해 이야기 한다. 다양한 응용분야에서의 설계 요구사항을 충족시키는 ASIP의 빠른 개발을 위해서, 건전한 설계 방법론 및 고성능의 시뮬레이터가 필요하다. 본 논문에서는 HiXR²라는 ADL(architecture description language)을 이용하여 인스트럭션 수준에서 컴파일드 시뮬레이터를 자동 생성하였다. 컴파일드 시뮬레이션은 시뮬레이션 수행 시 반복되는 인스트럭션 페칭 및 디코딩 부분을 시뮬레이션 런-타임 이전에 미리 수행함으로서 일반적으로 사용되는 인터프리티브 시뮬레이션에 비하여 큰 성능향상을 얻을 수 있다. HiXR²에 기반 한 컴파일드 시뮬레이션은 ARM9 프로세서와 CalmRISC32 프로세서 예제들로 수행하였고, 결과로서 인터프리티브 방식에 비해 150배 이상의 성능향상이 있었다.

Ⅰ. 서 론

ASIP 개발은 군사, 상업적으로 사용되는 응용 프로그램의 핵심이 되는 중요한 기술이다. 뿐만 아니라, 시스템칩(SOC)과 같은 ASIP을 이용한 시스템 집적을 위해서는 이른 설계 단계에서 ASIP을 다른 시스템 콤포넌트들과 함께 시뮬레이션하고 검증하는 일이 필요하다. 따라서 ASIP 시뮬레이터의 수행 속도는 시스템 개발에 큰 영향을 끼친다.[1]

컴파일드 시뮬레이션의 요지는 빈번히 반복되는 비효율적인 부분을 컴파일 타임에 한번만 수행하고 시뮬레이션 런-타임에는 꼭 필요한 부분만을 수행할 수 있도록 하여 성능을 향상 시키는 것이다.[2][3] 그러나, 정해진 프로세서 구조에 국한하

여 시뮬레이터를 개발하는 것은 시스템 집적 시 프로세서의 변화에 따라 각각의 시뮬레이터를 개발해야 할 뿐만 아니라, 에러가 발생하기도 쉽기 때문에 매우 시간 소모적인 일이 된다.

이에 따라 등장한 것이 바로 프로세서 구조 기술 언어인 ADL과 시뮬레이터 자동 합성 기술이다.[4][5] ADL에서 프로세서 구조를 필요한 부분만 추상화 하여 기술한 후, 그 정보를 이용하여 시뮬레이터를 생성한다. 이러한 접근은 ADL 기술을 간단히 수정함으로서 쉽게 프로세서 구조를 변경하고, 그에 따른 시뮬레이터를 에러 없이 자동생성 시킬 수 있다는 장점이 있다. 그러나 하나의 프로세서에 최적화 되지 않은 시뮬레이터 형태이므로 직접 설계한 시뮬레이터에 비해 느려질 수 밖에 없는 단점을 갖는다.

본 논문에서는 HiXR^{2}라는 ADL과 컴파일드 시뮬레이션 기법을 접목하여, 유연성 있게 여러 프로세서를 타겟팅 할 수 있을 뿐만 아니라, 빠른 시뮬레이션 수행속도를 갖는 방법을 보여준다. 챕터 2에서는 컴파일드 시뮬레이션 기법과 인터프리티브 시뮬레이션 기법의 차이를 이야기하고, 챕터 3에서는 제시된 ADL인 HiXR^{2}에 대하여 간략히 설명한다. 챕터 4에서는 ARM9와 CalmRISC32 프로세서에 대한 컴파일드 시뮬레이션 실험 결과를 인터프리티브 방식과 비교하여 속도 향상을 보이고, 챕터 5에서 결론 및 추후과제를 보인다.

Ⅱ. 컴파일드 시뮬레이션

컴파일드 시뮬레이션의 목적은 시뮬레이션 시간을 줄이는 것이다. 인터프리티브 방식의 인스트럭션 수준의 프로세서 시뮬레이션은 하드웨어에서 일어나는 행위를 그대로 모사한다. 따라서 프로세서 시뮬레이터는 응용 프로그램의 인스트럭션을 읽어오고(fetching), 해당 인스트럭션의 행위를 디코딩 한 후, 디코딩 결과에 따라 수행하는 일을 반복하게 된다. 이러한 일련의 과정은 실제 프로세서에서는 전기적 신호에 따라서 순식간에 이루어지지만, 시뮬레이션에서는 많은 시간을 요하게 된다. 컴파일드 시뮬레이션은 시뮬레이션 수행 시에 일어나는 일들 중 일부를 시뮬레이션 런-타임 이전에 미리 수행하여, 시뮬레이션 런-타임 수행시간을 줄이는 테크닉이다.

일반적으로 ASIP 응용 프로그램들은 코드 안에 많은 루프를 가지고 있다. 프로그램 코드 자체는 길지 않지만, 많은 루프로 인하여, 인스트럭션들을 반복적으로 수행한다. 또한, 런-타임에 달라지는 것은 메모리나 레지스터 각각의 저장 소자의 값일 뿐이고, 인스트럭션 워드자체는 변하지 않는다. 따라서 런-타임에 반복되는 인스트럭션을 계속하여 페칭(fetching)과 디코딩 하는 것은 매우 시간 소모적인 일이다. 컴파일드 시뮬레이션에서는 응용 프로그램의 인스트럭션 페칭 및 디코딩을 런-타임 이전에 수행한다.

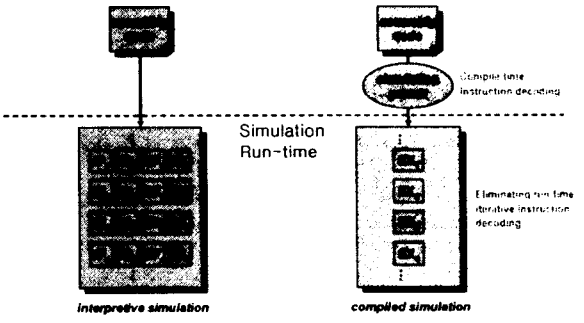

그림 1 인터프리티브 방식과 컴파일드 시뮬레이션 비교

그림 1은 인터프리티브 시뮬레이션과 컴파일드 시뮬레이션을 간략히 비교한 그림이다. 컴파일드 시뮬레이션에서는 시뮬레이션 파서에서 응용 프로그램 어셈블리를 받아들여 시뮬레이션 런-타임 이전에 페칭 및 디코딩을 하고, 실행(execution) 부분만을 런-타임에 수행하도록 만들어 큰 성능 향상을 얻을 수 있다.

컴파일드 시뮬레이터 합성에 대한 전체 프레임웍

은 그림 2와 같다.

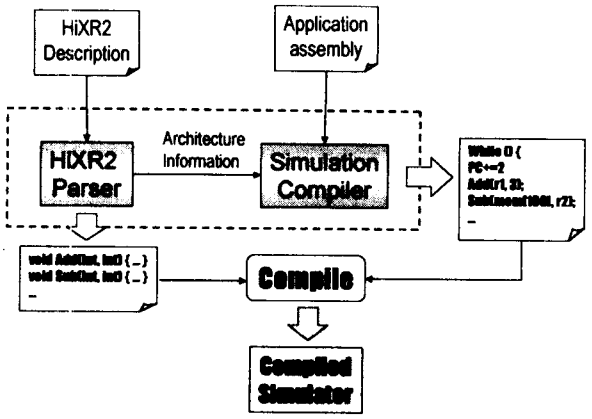

그림 2 컴파일드 시뮬레이션 전체 프레임웍

HiXR^2 파서는 인스트럭션 정보를 갖는 HiXR^2 기술을 받아들여 각 인스트럭션의 정보를 의미 있는 C 코드로 변환한다. 이와 동시에 시뮬레이션 컴파일러는 수행할 응용 프로그램의 어셈블리와 HiXR^2 파서로부터 인스트럭션 구조 정보를 입력으로 받아서 인스트럭션 및 오퍼런드들의 페칭 및 디코딩을 수행한다. 페칭 및 디코딩 결과는 호스트 컴퓨터(시뮬레이션을 수행하는 컴퓨터)에서 돌릴 수 있는 형태의 C 코드로 변환되고, HiXR^{2}에서 생성된 C 코드와 함께 컴파일 하여 실행 가능한 컴파일드 시뮬레이터를 생성한다.

Ⅲ. HiXR^2 언어

HiXR^{2}는 프로세서의 컴파일러와 명령어 수준의 시뮬레이터를 자동 생성하기 위하여 고안된 언어이다. 프로세서의 동작과 인스트럭션의 의미를 표현하기 위하여 HiXR^{2}는 Storage, Addressing mode, Instruction의 세 가지 섹션으로 나누어 기술한다.

Storage

Storage는 타겟팅 하고자 하는 프로세서의 메모리, 레지스터 파일, 상태 레지스터 등의 저장 소자들의 개수 및 표현법 등을 기술한다.

Addressing mode

인스트럭션은 연산기호(mnemonic)와 오퍼런드(operand)로 이루어진다. Addressing mode는 이 오퍼런드들의 의미 및 표현법을 규정한다. 이 정보를 이용하여 인스트럭션 디코딩 시에 원하는 레지스터나 메모리 값을 페칭할 수 있다. 그림 3은 어드레싱 모드 중 하나인 LSL_IMM 어드레싱 모드를 기술한 것이다. "R1 ; LSL #10"과 같은 오퍼런드는 레지스터 R1의 값을 왼쪽 논리적 이동을 10만큼 한 후, 그 값을 넘겨준다는 의미가 된다.

```
AddrMode LSL_IMM : arm9x_AddrMode
{
        operands
        {
                2
                IMMEDIATE imm
                GR Rm
        }

        syntax {
                Rm ";""LSL " imm
        }

        action {
                return lshift(Rm,imm);
        }
}
```

그림 3 LSL_IMM addressing mode 기술

Instruction

Instruction은 연산기호 및 오퍼런드들의 의미, 그리고 오퍼런드들의 어드레싱 모드를 규정한다. 그림 4는 "ADD" 인스트럭션의 기술을 보여준다. 이 기술에 따르면, ADD 인스트럭션은 다음과 같은 형태의 어셈블리로 나타난다.

ADD R1, R2, R3, LSL #10

이 인스트럭션의 의미는 'R3의 값을 10만큼 왼쪽 논리이동을 한 수와 R2의 값을 더하여 R1에 넣는다.'가 된다.

```
Instruction add : ALU
{
        mnemonic "ADD"

        operands
        {
                3
                REG Rd : DST
                REG Rs : SRC
                LSL_IMM Oprnd2 : SRC
        }

        action
        {
                Rd = add(Rs, Oprnd2);
        }
}
```

그림 4 ADD instruction 기술

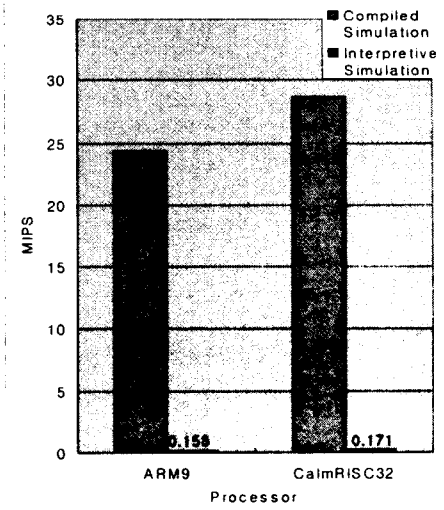

그림 5 시뮬레이터 성능 비교

Ⅳ. 실험 결과

컴파일드 시뮬레이션의 성능 향상을 측정하기 위하여, ARM사의 ARM9 프로세서와 삼성에서 개발된 CalmRISC32 RISC 프로세서를 이용하여 각각 인터프리티브 방식과 컴파일드 방식으로 시뮬레이션 하여 보았다. 시뮬레이션을 수행한 호스트 컴퓨터는 Athlon 2100+, 512MB이고, Windows XP를 기반으로 실험하였다. 사용된 응용 프로그램은 통신과 멀티미디어에 주로 사용되는 MediaBench[6] 중 하나인 ADPCM 프로그램이 사용되었다.

그림 5의 실험결과를 보면 컴파일드 시뮬레이션은 인터프리티브 방식에 비해 150배 이상의 성능 향상이 있음을 알 수 있다.

Ⅴ. 결론 및 추후과제

본 논문에서는 ASIP 시뮬레이션을 효율적으로 하기 위한 컴파일드 시뮬레이터 자동 생성 기법을 나타내었다. HiXR^{2}를 기반으로 간단한 기술을 통하여 프로세서 구조를 바꿀 수 있는 재적응성을 가질 뿐만 아니라, 매우 빠른 시뮬레이션 속도를 갖기 때문에, 프로세서 설계 공간 탐색이나 응용 프로그램의 코드 검증 시에 매우 유용하게 쓰일 수 있다. 본 연구의 정확성 및 성능 검증은 널리 사용되는 ARM9 프로세서와 삼성에서 개발된 RISC 프로세서인 CalmRISC32 프로세서를 이용하여 증명하였고, 컴파일드 방식은 일반적으로 사용되는 인터프리티브 방식에 비하여 150배가 넘는 속도 향상을 얻을 수 있었다.

추후 계획 사항으로는 HiXR^{2}의 표현력을 확장하여, 인스터럭션 수준의 병렬성을 나타내고 그에 따라 VLIW와 Super Scalar 머신의 컴파일드 시뮬레이션을 수행할 수 있도록 하는 것이다. 또한, 현재의 인스트럭션 수준의 시뮬레이션에서 파이프라인 구조 정보까지 컴파일드 시뮬레이션 할 수 있도록 확장하는 일이 필요하다.

참 고 문 헌

[1] M. Hartoog, J. Rowson, et al., "Generation of software tools from porcessor descriptions for hardware/software codesign," *in Proc. of the Design Automation Conference(DAC)*, Jun. 1997.

[2] V. Zivojnovic, S. Tjiang, and H. Meyr, "Compiled simulation of programmable DSP architectures," in *Proc. of IEEE Workshop on VLSI Signal Processing*, pp. 187-196, Oct. 1995

[3] S. Pees, V. Zivojnovic, A. Ropers, and H. Meyr, "Fast Simulation of the TI TMS 320C54x DSP," in *Proc. Int. Conf. on Signal Processing Application and Technology*, pp. 995-999, Sep. 1997.

[4] V. Zivojnovic, S. Pees, and H. Meyr, "LISA - machine description language and generic machine model for HW/SW co-design," in *Proceedings of the IEEE Workshop on VLSI* Signal Processing, Oct. 1996

[5] A. Halambi, P. Grun, et al., "EXPRESSION: A language for architecture expoloration through compiler/simulator retargetability," in *Proc. of the Conference on Design, Automation & Test in Europe,* Mar. 1999

[6] C. Lee, M. Portkonjak, W. H. Mangione-Smith "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", in *Proc. IEEE/ACM International Symposium on*, pp. 330-335, 1997