IR Simulator 를 위한 Data structure 와 Interface 연구

최영규, 안민욱, 윤종희, 권용인, 윤종희, 백윤흥
서울대학교 전기컴퓨터공학부
e-mail : ykchoi,mwahn,jhyoun,yikwon, jhyoon@optimizer.snu.ac.kr, ypaek@snu.ac.kr

Development of Data structure and Interface for IR Simulator

Young-Kyu Choi, Min-Wook Ahn, Jong-Hee Youn, Yong-In Kwon, Jong-Hee Yoon, Yun-Heung Paek
Dept. of Electrical Engineering, SNU

요 약

Intermediate Representation(IR)은 컴파일러가 Target code 를 생성하기까지의 중간 단계에서 프로그램을 표현하는 수단이다. 본 논문에서는 컴파일러의 IR 을 직접 Simulation 해서 각종 profiling information 을 수집하여 프로그램의 성능 향상에 도움을 줄 수 있는 IR Simulator 를 제안하고자 한다. 더 나아가 IR Simulator 를 위한 Data structure 와 Interface 는 Target Simulation 에도 재사용이 가능하도록 고안되었기 때문에 재겨냥성 컴파일러의 Target Simulator 를 자동으로 생성하는 연구를 진행하고 있다.

1. 서론

컴파일러는 프로그램을 IR 로 바꾼 후 IR 에 대해 다양한 분석과 최적화를 시도하여 최종 Target code 를 생성한다. Application Specific Instruction set Processor(ASIP) 설계 방법론에 활용도가 높은 재겨냥성 컴파일러가 사용하는 IR 의 경우 분석과 변경의 용이성 이외에 프로그램의 성능 향상을 위한 정보를 제공하는 역할을 할 수 있다.

이런 정보들에는 프로그램의 hot spot, instruction pattern 등이 포함되고 이를 통해 compiler friendly ASIP design 이 가능해진다. 하지만 이런 정보들을 얻기 위해서는 최종 target code 를 생성하고 이를 실행하여 profiling 해야만 한다.

위의 과정을 개선하고자 본 논문에서는 가상 머신(Virtual Machine)에 기반한 IR 과 IR Simulator 를 제안하고 그 구조를 설명한다. 그리고 더 나아가 제안된 IR Simulator 의 Data structure 와 Interface 가 Target Simulator 에도 재사용이 가능하도록 고안되어 재겨냥성 컴파일러에서 Target Simulator 를 생성하는 데에도 사용될 수 있음을 논의하도록 한다.

본 논문의 2 장에서는 가상 머신에 기반을 둔 IR 에 대해서 설명하고, 3 장에서는 IR Simulator 의 생성과 구조에 대해서 설명하도록 하겠다. 그리고 마지막 4 장에서 결론을 맺도록 하겠다.

2. 가상 머신 IR

전체적인 방법론은 Source level code 에서 GCC 를 이용하여 가상 머신을 target 으로 하여 IR 을 생성하고, 이 IR 을 실제 머신으로 targetting 하는 것이다. 이 장에서는 가상 머신과 가상 머신 IR 에 대해 설명하도록 한다.

가상 머신은 32-bit 머신으로 충분히 많은 general purpose register 와 byte addressable memory 를 가지도록 한다. 또한 가상 머신은 base offset 과 absolute addressing mode 를 허용하며, stack pointer 와 frame pointer register 를 두도록 한다.

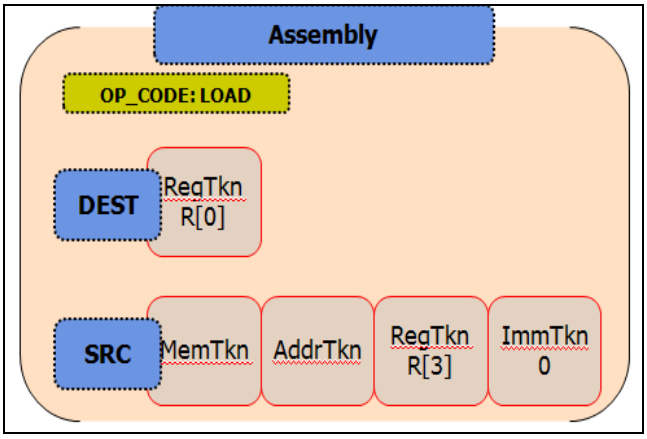
Insn	Virtual Machine code
LOAD	(set (SI:R[0]) (SI:0(R[3])))
STORE	(set (SI:0(R[4])) (SI:R[0]))
ADD	(set (SI:R[3]) (ss_plus:SI (SI: R[3]) (SI: R[0])))
ASHIFT	(set (SI:R[0]) (ashift:SI (SI: R[12]) (const: SI 3)))

(그림 1) Instruction format 예제

가상 머신의 Instruction Set Architecture(ISA)는 Architecture independent minimal set 으로 구성되어 있다. Instruction set format 은 그림 1 에서 볼 수 있듯이 3-address code 로 (set : DEST SRC)의 형태로 구성된다. DEST 에는 register, memory 등이 올 수 있으며, SRC 에는 register, memory, operation 등이 오게 된다. Instruction 에는 기본적인 move, load, store 와 arithmetic, logical operation 을 가지고 있으며, branch, call operation 등의 control flow operation 도 포함되어 있다. Data mode 의 경우 QI, HI, SI, DI mode 가 있으며 각각 8, 16, 32, 64 bits 를 나타낸다.

그림 1 과 같은 형식으로 생성된 가상 머신 code 를

컴파일러가 읽어 들여 아래의 그림 2 와 같은 자료구조를 가진 IR 로 구성하여 지니게 되고, 여기에 여러 가지 기법을 통해 분석하고 최적화를 시도하여 최종 Target code 를 생성하게 된다.



(그림 2) LOAD IR 예제

3. IR Simulator

IR Simulator 는 기본적으로 simulator 내부에 simulation 을 수행하는 code 와 semantic code 를 함께 포함하고 있는 compiled simulation 기법으로 구현하였다. 그림 3 에서처럼 가상 머신에 기반한 IR 을 동일한 semantic 을 가진 C code 로 변환시켜 host machine 상에서 컴파일 및 실행하여 simulation 을 하도록 구현되었다.

```
ENTER_BB("pin_down_B304");

assembly(Vasm, "", OP_LOAD, 7,
          new RegToken(301, _SI_, 1, 0, 0, true), new UsageType(0x1, -1
          new MemToken("", _SI_, "zqidaugfnc", true), new UsageType(0x2
          new AddrToken("zqidaugfnc", OP_EFA_PLUS, "", 2,
                    new RegToken(201, _SI_, 1, 0, 0, true), new UsageTy
                    new ImmToken(-4, 32, 0, 31, false, IMM_INT), new Us

assembly(Vasm, "", OP_LOAD, 7,
          new RegToken(302, _SI_, 1, 0, 0, true), new UsageType(0x1, -1
          new MemToken("", _SI_, "qyyqhofyep", true), new UsageType(0x2
          new AddrToken("qyyqhofyep", OP_EFA_PLUS, "", 2,
                    new RegToken(201, _SI_, 1, 0, 0, true), new UsageTy
                    new ImmToken(-8, 32, 0, 31, false, IMM_INT), new Us

assembly(Vasm, "", OP_MOV, 5,
          new RegToken(303, _SI_, 1, 0, 0, true), new UsageType(0x1, -1
          new ImmToken(1, 32, 0, 31, false, IMM_INT), new UsageType(0x2


EXIT_BB("pin_down_B304");
```

(그림 3) IR Simulator 예제

가상 머신 IR 의 instruction 은 function call 로, memory 와 register 는 각각의 특징을 모사한 자료구조로 치환하여 C code 를 생성한다. Branch 와 function call 의 경우 각각 target label 과 target function pointer 를 IR 에서 추출하여 처리한다. Library call 의 경우는 semi-hosting 기법을 사용하여 library 를 직접 구현하지 않고도 host machine 의 library 에 연결되도록 하여 쉽게 simulation 결과를 확인할 수 있도록 한다.

IR Simulator 를 통해 Instruction count 와 Basic Block frequency 와 같은 profiling 정보를 수집하여 hot spot 을 찾아낼 수 있고, hot spot 에서의 instruction pattern 을 정보를 알아낼 수 있다. 이를 통해 ASIP 에서 응용 프로그램의 성능 향상을 위한 새로운 instruction 추가에 근거자료를 얻을 수 있다.

또한 register, memory 그리고 IR 의 자료구조와 Simulation code 의 interface 부분에서 Target Simulator 생성을 고려하여 고안하였기 때문에 재겨냥성 컴파일러가 Target code 를 생성했을 때 Simulator 도 함께 자동 생성되는 flow 까지 쉽게 확장될 수 있을 것이다.

4. 결론

오늘날 ASIP 기반의 설계 방법론에서 많이 적용되는 재겨냥성 컴파일러를 위한 IR 을 디자인할 때에는 일반적인 IR 이 가져야 할 특징인 구조를 변경하기 쉽고 변경이 용이해야 한다는 것 이외에 프로그램의 성능을 향상 시킬 수 있는 정보를 제공할 수 있어야 한다는 이슈를 고려해야 한다. 이를 위해서 논문 [3] 에서 IR Simulator 를 제안하였다.

본 논문에서는 IR Simulator 의 Data structure 와 Interface 를 새롭게 고안하여 Target Simulator 생성에도 재사용될 수 있도록 구현하였다. 이를 통해 재겨냥성 컴파일러의 Target Simulator 자동생성에 대한 이슈로 확장할 수 있을 것이다.

참고문헌

[1] James E. Smith, Ravi Nair, Virtual Machines Versatile Platforms for Systems and Processes, Morgan Kaufmann Publishers, ISBN 1-55860-910-5

[2] 황승호, 이종열, “컴파일러의 개발” , IDEC

[3] 최영규 et al., “IR 시뮬레이터”, 프로그래밍언어 연구회 논문지, 2008

[4]Oliver Wahlen, Rainer Leupers, application specific compiler/architecture codesign:a case study, LCTES/SCOPES 02

[5] JongYeol Lee, In Cheol Park, Instruction set generation considering the IR of compilers, KAIST

Acknowledge

본 연구는 교육과학기술부/한국과학재단 우수연구센터육성사업(R11-2008-007-01001-0), 지식경제부 출연금으로 ETRI, SoC 산업진흥센터에서 수행한 ITSoC 핵심설계인력양성사업, 서울시 산학연 협력사업(10560), 2009 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.2009-0083190)의 지원을 받아 수행되었습니다.