머쉰 행위기술로부터 Retargetable 컴파일러 생성시스템 구축

준회원 이 성 래*, 정회원 황 선 영*

Construction of a Retargetable Compiler Generation System from Machine Behavioral Description

Sung-Rae Lee* *Associate Member*, Sun-Young Hwang* *Regular Member*

요 약

ASIP 디자인에서 디자인되는 프로세서의 성능을 측정할 수 있는 컴파일러가 요구된다. 머쉰에 맞는 컴파일러 설계는 매우 많은 시간을 요구한다. 본 논문은 MDL 기술로부터 C 컴파일러를 생성하는 시스템을 보인다. MDL을 이용한 컴파일러 생성은 user retargetability와 컴파일러와 프로세서 디자인의 일관성을 유지시켜 준다. 그러나 MDL을 이용한 컴파일러 생성 시스템은 컴파일러와 머쉰 간 의미적 차이를 줄여야 한다. 이러한 문제를 해결하기 위해 제안하는 시스템은 트리 패턴마다 행위정보를 가지는 라이브러리에 머쉰 행위기술을 맵핑한다. 맵핑된 인스트럭션과 레지스터 파일 사용정보를 이용해 제안하는 시스템은 컴파일러 후위부 interface function을 생성한다. 생성된 MIPS R3000와 ARM9 컴파일러가 C로 기술된 어플리케이션 프로그램으로 검증되었다.

Key Words : Retargetable compiler, ASIP, MDL, User retargetability, SSA form

ABSTRACT

In ASIP design, compiler is required for performance evaluation of processors being designed. The design of machine specific compiler is time consuming. This paper presents the system which generates C compiler from MDL descriptions. Compiler generation using MDL can support user retargetability and concurrency between compiler design and processor design. However, it must overcome semantics gap between compiler and machine. To handle this problem, the proposed system maps behavioral descriptions to library which contains abstract behavior for each tree pattern. Using mapped instructions and information on register file usage, the proposed system generates back-end interface function of the compiler. Generated compilers, for MIPS R3000, ARM9 cores, have been proved by application programs written in C code.

Ⅰ. 서 론

C나 C++등의 상위수준 언어를 타겟 프로세서 코드로 변환 시키는 컴파일러는 오랫동안 운영체제나 어플리케이션 소프트웨어 설계에 쓰여 왔다. 최근 내장형 시스템의 수요가 늘고 제한된 전력과 비용 내에서 최대의 성능을 낼 수 있는 ASIP (Application Specific Instruction set Processor)의 사용이 급격히 증대되면서 ASIP에 최적화된 컴파일러 설계의 중요성이 증가하고 있다. 이러한 추세는 컴파일러가 단순히 코드를 생성해 주는 것뿐만 아니라 ASIP 설계간 프로세서 인스트럭션 세트의 변경에 따른 성능 측정을 위해 필요한 인스트럭션 변경에 따른 컴파일러를 생성할 수 있는 retargetable

※ 본 논문은 2007년도 「서울시 산학연 협력 사업」의 「나노 IP/SoC 설계 기술 혁신 사업단」의 지원으로 이루어졌습니다.
* 서강대학교 전자공학과 CAD&ES 연구실 (hwang@sogang.ac.kr)
논문번호 : KICS2007-02-052, 접수일자 : 2006년 2월 9일, 최종논문접수일자 : 2006년 2월 10일

compiler를 요구하게 되었다[1][2]. 컴파일러의 설계는 지나치게 많은 비용과 시간을 필요로 하므로 Time-to-Market의 중요성이 부각되고 있는 현 추세에서 컴파일러 설계는 상당한 병목으로 작용하고 있다.

머쉰 기술 언어로부터 컴파일러 생성은 컴파일러 설계시간을 단축시켜 줄 뿐만 아니라 하나의 기술로 컴파일러와 프로세서 설계가 가능하므로 디자인 일관성을 유지 할 수 있으며 컴파일러 프로세서 상호 디자인을 통해 프로세서 최적화를 이룰 수 있다는 점에서 의의가 있다.

본 연구에서는 SMDL (Sogang Machine Description Language)을 이용하여 LCC (Little C Compiler)[3][4]의 후위부 자동 생성기인 lburg[5]의 입력을 생성하는 SRCC (Sogang Retargetable Compiler Compiler)를 구축하였다. 머쉰 기술 언어로부터 컴파일러 생성을 위해서는 머쉰-컴파일러 간 의미적 차이를 줄여야 한다. 제안된 시스템은 머쉰 행위정보를 컴파일러의 내부 트리 패턴에 따른 행위 정보를 가지는 라이브러리와 맵핑을 통해 컴파일러의 인스트럭션 맵핑 룰을 만든다. 또한 다양한 맵핑 방식을 통해 맵핑 영역의 확대와 어플리케이션에 최적화된 인스트럭션 셋의 선택이 가능하게 하였으며 머쉰 기술만으로 컴파일러를 생성하는 시스템을 구축하였다.

본 논문의 2절에서는 관련 연구를 기술하며 3절에서는 SMDL 시스템을 보인다. 4절에서는 구축된 컴파일러 생성 시스템인 SRCC의 개관과 인스트럭션 선택기 생성기, 그리고 seamless 컴파일러 생성 과정을 보인다. 5절에서는 구축된 시스템의 검증을 위한 시뮬레이션 결과를 보이고, 마지막으로 6절에서는 결론 및 추후 연구 과제를 제시한다.

Ⅱ. 관련 연구

ASIP설계를 지원하기 위한 MDL 기반의 retargetable 컴파일러 생성은 많은 연구가 되어왔다. 이중 RECORD[6], CHESS[7][8], LISA[9]-[11]가 대표적인 MDL을 이용한 컴파일러 생성 시스템이다. RECORD 컴파일러 시스템은 MIMOLA HDL[12]로 생성된 net-list로부터 인스트럭션 정보를 축출한 후 iburg 입력을 만듦으로써 컴파일러를 생성한다. MIMOLA는 매우 낮은 레벨에서 기술이 이루어지고 이로 인해 기술과 수정이 어렵다.

CHESS 코드 생성기는 nML[13]을 입력으로 받아 저장 유닛, 오퍼레이션, 저장 유닛 간 연결 관계 등의 정보를 가지는 ISG (Instruction Set Graph) 생성 후 코드 생성을 한다. nML은 언어자체에서 파이프라인을 지원하지 않아 파이프라인 기능이 있는 프로세서 생성에 어려움이 있다.

LISA는 ACE의 CoSy 컴파일러 시스템[14]의 후위부 생성기 기술인 CGD를 생성함으로써 컴파일러를 생성한다. LISA는 소스코드의 서브젝트 트리와 인스트럭션 간 다양한 맵핑 방식을 사용하며, C/C++로 기술된 행위 정보의 복잡함으로 behavior 기술과 분리된 semantics 섹션을 만들어 컴파일러 생성을 위한 정보를 얻는다. 하지만 behavior 정보가 아닌 semantics정보에 의한 컴파일러 생성은 전체적인 기술 량이 증가하며, 행위 정보로부터 컴파일러 생성 정보를 얻지 못하므로 이상적이지 못하다.

본 논문에서 제안하는 SRCC 시스템은 행위/구조 정보를 모두 가지는 SMDL의 인스트럭션 행위 정보 기술로 LISA와 같은 다양한 맵핑이 가능하도록 하였다.

Ⅲ. SMDL 개관

SMDL 시스템은 SoC 설계 자동화를 위해 구축한 시스템으로, 타겟 프로세서를 자동 생성 해주는 CoreGen과 SMDL로 기술된 프로세서에 최적화된 컴파일러를 생성 해주는 SRCC, 타겟 프로세서의 ISS (Instruction Set Simulator)를 자동 성성해 주는 RISGen (Retargetable Instruction-set Simulator Generator)으로 이루어져 있다.

설계자는 이 시스템에 타겟 프로세서의 SMDL기술과 어플리케이션 프로그램을 입력하여 일련의 피드백을 통해 어플리케이션에 최적화된 Processor Core, 컴파일러, ISS가 자동생성 된다. 그림 1에 SMDL 시스템을 보인다.

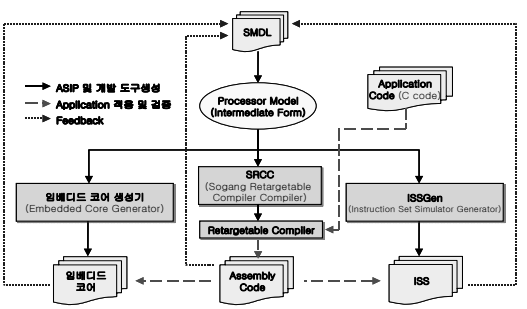


그림 1. SMDL 시스템 개관

그림 2. SMDL의 기술구조

SMDL은 블록 다이어그램 수준의 구조 기술과 C언어 수준의 인스트럭션의 행위 기술을 통해 타겟 프로세서를 기술할 수 있도록 개발된 MDL이다[15].

SMDL은 크게 내부 구조 기술부와 외부 구조 기술부, 인스트럭션 셋 기술부로 나뉜다. 내부 구조 기술부에서는 타겟으로 하는 프로세서의 실행 유닛이나 레지스터, 메모리 등을 선언해 주고 각각의 내부 리소스들이 어느 스테이지에 할당되는지에 대해 기술한다. 외부 구조 기술부에서는 프로세서 이름, 외부 포트, 인터럽트 등을 정의한다. 인스트럭션 셋 기술부에서는 어드레싱 모드 및 각각의 인스트럭션의 특성과 행위정보를 기술하게 된다. SMDL의 기술구조는 그림 2와 같다.

SRCC는 SMDL의 인스트럭션 셋 기술부와 내부 구조 기술부의 리소스 타입정보를 이용하여 기술된 프로세서에 최적화된 컴파일러를 생성한다.

Ⅳ. 구축된 컴파일러 생성 시스템

SRCC 시스템은 SMDL의 행위정보로부터 라이브러리 맵핑에 용이한 중간 형태를 만들고, 그 중간 형태와 동일한 라이브러리 중간 형태에 맵핑을 함으로써 lburg의 컴파일러 코드 선택기 기술을 생성한다. SMDL만을 입력으로 하는 seamless 컴파일러 생성을 위해 SRCC는 SMDL의 레지스터 파일 사용기술과 라이브러리 맵핑된 인스트럭션을 이용하여 컴파일러의 후위부 interface function을 생성한다. 그림 3은 SRCC 시스템의 개관이다.

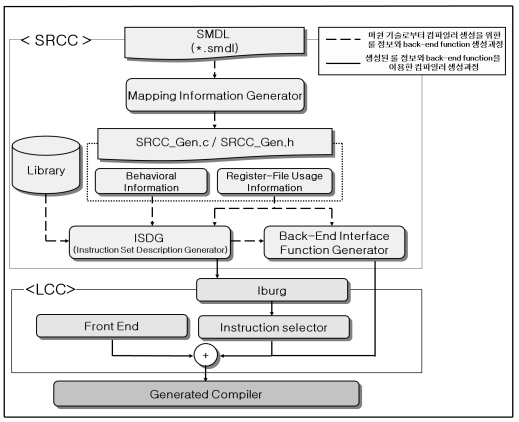


그림 3. SRCC 시스템 개관

4.1 SRCC 맵핑 영역

SRCC는 SISD (Single Instruction Single Data) 모델을 기본으로 하는 파이프라인 아키텍춰 모델을 목표로 한다. 보통의 다이나믹 프로그래밍 기반 트리 코드 선택기[5][16]가 statement level을 넘는 코드 생성을 하지 않고 루프문을 맵핑하기 어려우므로 루프문을 배제하였다.

4.2 SRCC 라이브러리

SRCC 라이브러리는 머쉰 행위 정보와 컴파일러 사이에 발생하는 의미적 차이를 극복하기 위해 사용된다. SRCC 라이브러리는 C와 유사한 문법으로 트리 패턴에 맵핑될 수 있는 머쉰 행위 정보를 기술하고 라이브러리 행위 기술에 SMDL로 기술된 머쉰 행위 정보를 맵핑함으로써 각각의 트리 패턴에 대한 인스트럭션 맵핑 룰을 생성 가능하게 한다. 그림 4는 SRCC 라이브러리의 개략적인 BNF (Backus-Naur Form) 이다.

```
library_list → library
             | library_list library

library → tree_representation = behavior_list

behavior_list → behavior
              | behavior _list | behavior

behavior → [ declaration_list statement_list ]
```

그림 4. 라이브러리 BNF form.

하나의 트리 패턴에는 여러 개의 행위 정보들이 OR룰로 매치가 된다. SRCC는 맵핑 중 하나의 트리 패턴에 대해 여러 개의 행위가 맵핑되었을 시 가장 비용이 적은 행위 정보에 대한 코드를 생성한다. SRCC 라이브러리는 SMDL보다 간소화된 타입으로 하나의 인스트럭션 내지는 여러 인스트럭션 간 행위 정보를 기술 가능하도록 다음과 같은 타입을 가진다.

- T-type : 하나의 인스트럭션 내부 또는 여러 인스트럭션간 데이터 전달을 해주는 타입이다. SMDL

의 non-terminal 타입 , 레지스터 타입 , 템포러리 타입 , 스테이지 레지스터 타입 등이 T-type에 맵핑된다.
- I-type : 인스트럭션의 immediate 필드가 I-type으로 맵핑된다.
- P-type : 프로그램 카운터 타입이 맵핑된다.
- M-type : 메모리 타입이 맵핑된다.

그림 5는 라이브러리 기술의 예이다.

```
reg : BCOMU4(T-type A[32] : reg) =
    [ T-type B[32]; B = ~A; ]  |
    [ T-type B[32]; B = ~(A | 0); ] ;
```

그림 5. 라이브러리 기술 예.

임의의 레지스터 값을 inversion시키는 트리 패턴 기술인 reg : BCOMU4(reg)에 대해 SRCC 라이브러리는 T-type A를 inversion 시키거나 A와 0을 NOR하여 T-type B에 대입하는 행위 기술을 가진다.

4.3 SMDL의 레지스터 파일 사용정보 기술

레지스터 파일 사용정보 기술은 SRCC의 seamless 컴파일러 생성을 위해 SMDL에 추가된 기술로 각각의 레지스터 파일을 구성하는 레지스터마다 컴파일러의 활용 정보를 포함하게 하여 인스트럭션 맵핑과 seamless 컴파일러 생성이 가능하게 한다. 표 1은 SMDL의 레지스터 사용 기술을 보인다.

표 1. 레지스터 사용 기술의 종류와 설명.

사 용 정 보	설 명
constant((+|-)?[0-9]+)	임의의 값으로 고정된 레지스터
for_assembler	어셈블러가 사용하는 레지스터
argument	argument 레지스터
temporary	function 호출 간 저장되지 않는 레지스터
saved	function 호출 간 저장되는 레지스터
function_call	function 호출시 호출되는 function의 어드레스를 가지는 레지스터
function_return_value	function의 반환값을 가지는 레지스터
stack_pointer	스택 포인터
frame_pointer	프레임 포인터
return_address	function이 끝날 시 돌아갈 어드레스를 가지는 레지스터

SMDL을 기술 시 레지스터 파일의 terminal 기술에서 위의 사용 정보중 하나를 선택하여 기술한다. 그림 6은 레지스터 사용 정보가 추가된 SMDL 기술의 일부이다.

```
R0 = address (RF[0]) {
  binformat = 5b'0; mnemonic = "$zero";
  usage = constant(0);
}
R1 = address (RF[1]) {
  binformat = 5b'1; mnemonic = "$at";
  usage = for_assembler;
}
R2 = address (RF[2]) {
  binformat = 5b'2; mnemonic = "$a0";
  usage = argument;
}
                    ⋮
R31 = address (RF[31]) {
  binformat = 5b'31; mnemonic = "$ra";
  usage = return_address;
}
 RS =
R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 |
R12 | R13 | R14 | R15 | R16 | R17 | R18 | R19 | R20 | R21 |
R22 | R23 | R24 | R25 | R26 | R27 | R28 | R29 | R30 | R31;
```

그림 6. 레지스터 사용정보 기술 예

각각의 terminal 마다 레지스터 사용 정보가 기술 되고 RS는 R0부터 R31까지 terminal 정보를 가지는 non-terminal로 기술된다.

4.4 중간형태 생성과정

SMDL과 SRCC 라이브러리의 행위정보는 동일한 중간 형태로 변환된다. 중간 형태는 high-level SSA form[17]으로, SSA form[18]은 리네이밍과 Φ-function을 통해 모든 변수가 단 한번 값이 대입되는 형태이다. Alpern et al.[17]은 SSA form에 컨트롤 정보를 추가시켜 high-level SSA form을 제안하였다. High-level SSA form은 프로그램 동일성 체크에 이용 되었다[17][19]. SRCC는 high-level SSA form을 이용한 프로그램 동일성 체크를 활용하여 SMDL을 라이브러리에 맵핑한다. SSA form 생성을 위해 CFG (Control Flow Graph) 로부터 도미넌스 프런티어 맵핑을 하였으며, 맵핑된 도미넌스 프런티어를 이용해 Φ-function을 삽입하고 리네이밍하였다. High-level SSA form은 SSA form과 CDG (Control Dependence Graph)를 사용하여 구현하였으며, high-level SSA form의 맵핑 가능 영역을 넓히기 위해 copy propagation을 수행하였다.

4.5 맵핑을 위한 C 코드 생성 과정

SMDL의 행위 기술은 SRCC와 맵핑을 위해 SRCC 라이브러리와 동일한 중간 표현 형태로 변형된 후 맵핑 정보를 가지는 C 코드를 생성한다. 각각의 SMDL로 기술된 인스트럭션들의 중간 형태 정보는 하나의 C로 기술된 function으로 만들어지며, 레지스터 사용 기술 및 인스트럭션 표현 정보는 배열로 변환되어 표현된다. 인스트럭션 중간 형태를 C 코드로 변환할 시 교환 법칙이 성립하는 오퍼레이션을 가지는 SSA 노드에 대해서는 노드의 좌우 자식노드의 위치를 바꾼 정보까지 고려한 정보를 C 코드로 생성함으로써 맵핑 가능 영역을 넓힌다.

4.6 SMDL과 라이브러리 맵핑 과정

SMDL과 라이브러리의 맵핑은 high-level SSA form간 맵핑이다. 루프를 배제한 맵핑 영역의 제한으로 SMDL과 라이브러리 간 맵핑은 DAG (Direct Acyclic Graph) 맵핑이고 DAG를 duplication 함으로써 트리 맵핑이 되도록 한다.

SRCC는 T-type을 통해 하나의 인스트럭션 내부 또는 여러 인스트럭션간 데이터 전송을 표현할 수 있으므로 하나의 라이브러리 행위 정보에 여러 가지 인스트럭션이 맵핑될 수 있다. SRCC는 인스트럭션의 개수를 비용으로 계산하여 다이나믹 프로그래밍에 기반 한 가장 비용이 적은 인스트럭션을 맵핑한다. SRCC는 LISA와 같은 one-to-one, one-to-many, many-to-one 맵핑 방식을 행위기술 맵핑을 통해 수행한다. One-to-one 맵핑은 단순히 하나의 서브젝트 트리에 하나의 인스트럭션이 맵핑되는 방식이며, One-to-many 맵핑은 하나의 서브젝트 트리에 여러 개의 인스트럭션을 맵핑하는 방식이다. One-to-many 맵핑은 특정 서브젝트 트리에 one-to-one 맵핑되는 인스트럭션이 정의되지 않아도 다른 정의된 인스트럭션들의 조합이 서브젝트 트리에 맵핑 가능하게 함으로써 많이 사용되지 않는 특정 트리 패턴만을 위한 인스트럭션 사용을 줄일 수 있다. Many-to-one 맵핑은 ASIP 설계에 있어 매우 중요한 맵핑 방식으로 여러 서브젝트 트리에 하나의 인스트럭션을 맵핑하는 방식이다. Many-to-one 맵핑은 여러 서브젝트 트리를 하나의 인스트럭션으로 맵핑함으로써 수행속도를 높이며, 이는 특정 어플리케이션을 위한 연산이 많은 ASIP의 특성상 Many-to-one 맵핑은 수행성능 향상 요인이 될수 있다.

One-to-one, many-to-one 맵핑은 라이브러리 행위기술에 하나의 인스트럭션이 맵핑 되지만 서브젝트 트리 패턴기술에 들어가는 트리 오퍼레이션 개수에 따라 one-to-one과 many-to-one 맵핑이 될 수 있다. 그림 7은 one-to-one과 many-to-one 맵핑에 따른 개략적인 라이브러리 기술이다.

one-to-one	stmt : ADDI4(T-type A[32] : reg , T-type B[32] : reg) = *behavior_list*
many-to-one	stmt : ADDI4(T-type A[32]: reg, MULI4(T-type B[32]: reg, T-type C[32]: reg)) = *behavior_list*

그림 7. one-to-one, many-to-one 맵핑을 위한 라이브러리 기술 예

T-type을 이용한 라이브러리 기술은 여러 인스트럭션간 데이터 전송 행위 기술까지 가능하게 함으로써 트리 패턴과 인스트럭션간 one-to-many 맵핑이 가능하게 한다. 그림 8은 SMDL의 SLT와 BNE 인스트럭션이 LTI4라는 트리 패턴의 라이브러리 행위 기술에 one-to-many 맵핑되는 예를 보인다.

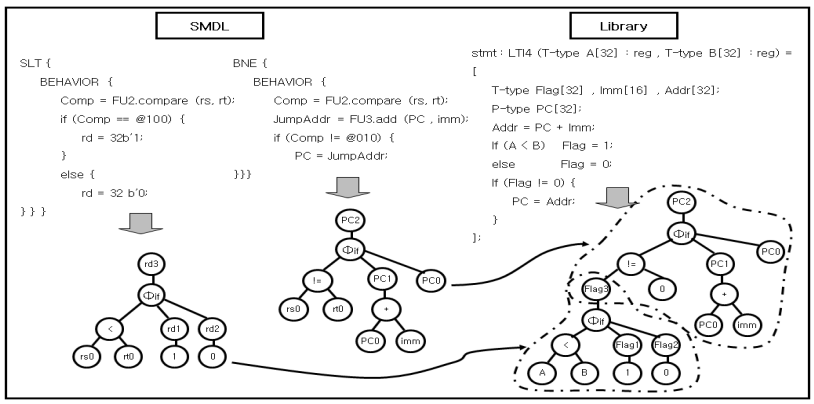


그림 8. SMDL-라이브러리 맵핑 예

4.7 레지스터 / 상수 할당

One-to-many 맵핑 시 인스트럭션 간 데이터 전송을 non-terminal 타입의 리소스가 담당할시 레지스터 할당이 필요하다. 이런 경우 SRCC는 인스트럭션 간 데이터 전송을 위한 레지스터를 사용 정보가 'for_assembler'인 레지스터로 할당한다. 라이브러리의 행위 정보가 임의의 상수를 사용하고 맵핑되는 SMDL 행위 기술이 immediate 데이터 타입을 사용할 시 immediate 데이터에 해당 상수를 할당한다. 임의의 상수에 맵핑되는 SMDL non-terminal 타입의 terminal 기술 중 맵핑되는 상수로 고정된 terminal이 있을시 해당 terminal을 맵핑시킨다. 예를 들어, '$at' terminal 레지스터의 사용 정보를 'for_asembler'로 설정하고 '$zero'라는 terminal 레지스터의 사용 정보를 'constant(0)'라 기술할 시, 그림 8의 맵핑 결과는 레지스터 할당과 상수 정보에 대한 맵핑을 통해 그림 9의 출력을 낸다.

```
stmt : LTI4 (reg , reg)
       "SLT $at, %0, %1\nBNE $at, $zero, %a\n" 2
```

그림 9. 생성된 인스트럭션 선택기 기술의 예

'$at'가 할당되어 인스트럭션 간 데이터 전송을 수행하고 '$zero'가 상수 0에 맵핑된다.

4.8 컴파일러 후위부 interface function 생성 과정

컴파일러 후위부 interface function 생성을 위해 SRCC는 레지스터 사용 기술과 인스트럭션 맵핑 정보를 사용한다. 상수로 고정된 레지스터 사용 정보로부터 특정 상수에 대한 레지스터를 할당하며, 'argument'로 사용되는 레지스터 사용정보로 function 호출 시 argument의 전달방식을 결정한다. 'temporary'와 'saved'로 사용되는 레지스터 사용 정보로 컴파일러가 할당할 레지스터 필드를 결정하며, 'function_call' / 'function_return_value'로 사용 되는 레지스터 사용 정보로 function 호출시 function 어드레스 정보와 반환 값을 저장하는 레지스터를 할당한다.

컴파일러 후위부 interface function은 activation 레코드의 생성과 소멸, 특정 트리 패턴에 대한 인스트럭션 등에 대한 정보를 가진다. SRCC는 인스트럭션 선택기 생성을 위해 만들어진 인스트럭션 맵핑 정보로부터 컴파일러 후위부 interface function이 지정해야 할 타겟 프로세서의 인스트럭션에 대한 정보를 얻어, 후위부 interface function이 타겟 머쉰에 맞는 인스트럭션을 지정할 수 있게 한다.

Ⅴ. 실험 결과

구현된 시스템의 정확성과 유용성을 검증하기 위해 SMDL로 기술한 MIPS R3000[20], ARM9[21] 프로세서를 위한 컴파일러를 생성하였다. 본 실험에서는 임베디드 설계에 가장 많이 쓰이는 프로세서인 MIPS R3000과 ARM9 프로세서의 컴파일러를 생성 하였으나 다른 대부분의 프로세서에 대해 SMDL로 기술하고 컴파일러를 생성 할 수 있다. 표 2는 컴파일러 트리 패턴에 대해 SMDL로 기술한 인스트럭션 맵핑의 결과이다.

약 24~31%의 트리 패턴이 one-to-many 맵핑되었다. 이는 생성된 두 컴파일러가 one-to-many 맵핑을 고려하지 않을시 모든 트리패턴에 인스트럭션을 맵핑할 수 없음을 의미한다. 약 6~8%의 트리패턴이 many-to-one 맵핑을 통해 코드사이즈의 감소와 실행성능 향상을 얻었다.

표 2. 인스트럭션 맵핑 결과.

	MIPS R3000	ARM9
one-to-one	65 (69.1%)	49 (59.8%)
one-to-many	23 (24.5%)	26 (31.7%)
many-to-one	6 (6.4%)	7 (8.5%)

생성된 컴파일러가 정확히 동작하는지와 성능의 측정을 위해 C로 기술된 JPEG 인코더를 본 연구실에서 생성한 컴파일러로 컴파일하고 시뮬레이션 하였으며, 매뉴얼 포팅한 LCC 컴파일러와 성능을 비교하였다. 표 3은 메뉴얼 포팅한 LCC 컴파일러가 생성한 코드와 생성된 컴파일러가 생성한 인스트럭션 수를 보인다.

표 3. 컴파일된 JPEG 인코더의 인스트럭션 수

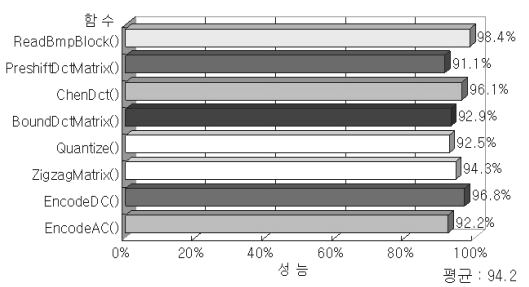
	manual LCC 컴파일러	생성된 컴파일러	비 고
MIPS R3000	2,533	2,773	9.4% 증가
ARM9	2,651	3,244	22.3% 증가

메뉴얼 포팅한 LCC 컴파일러에 비교해 MIPS R3000 컴파일러는 9.4%, ARM9 컴파일러는 22.3%의 코드사이즈 증가를 보인다. 이는 생성된 컴파일러가 레지스터에 임의의 상수나 어드레스를 저장할

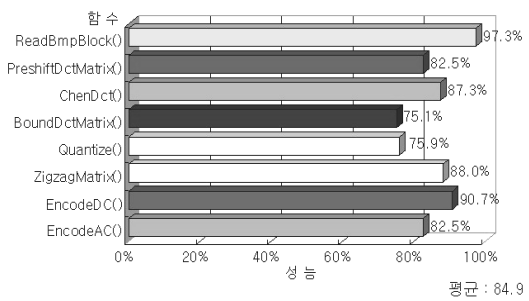
때와 immediate 필드를 가지는 인스트럭션에 대해 이상적인 코드생성을 못하여 메뉴얼 포팅한 LCC 컴파일러에 비해 많은 코드를 생성했기 때문이다. 생성된 ARM9 컴파일러는 activation record의 생성과 소멸에 관계된 연산을 하는 인스트럭션으로 매뉴얼 포팅한 LCC 컴파일러보다 많은 코드 사이즈를 가지는 인스트럭션을 선택한다. 이는 SRCC가 어떠한 프로세서 대해서도 activation record 모델을 만들어야 하므로 범용으로 사용할 수 있는 activation record를 생성 기준으로 했기 때문이다.

생성된 컴파일러가 생성한 JPEG 인코더 코드의 수행시간을 실행 cycle을 기준으로 측정하였다. 그림 10은 메뉴얼 포팅한 LCC 컴파일러의 수행 cycle을 100%로 할 때 JPEG 인코더의 중요 function별 성능 측정치를 보인다.

생성된 MIPS R3000 컴파일러는 메뉴얼 포팅한 LCC 컴파일러 대비 평균 94.2%의 성능을 보이며 ARM9 컴파일러는 평균 84.9%의 성능을 보인다. ARM9 컴파일러의 성능이 낮은 이유는 생성된 ARM9 컴파일러가 immediate 필드를 가지는 인스트럭션의 운용에 부담이 크고, ARM9 프로세서에 대한 activation record 모델이 최적화되지 않아



(a) MIPS R3000 컴파일러



(b) ARM9 컴파일러

그림 10. 생성된 코드의 성능 분석

MIPS R3000 ARM9

그림 11. 생성된 컴파일러에 의해 컴파일된 JPEG 인코더의 수행 결과

function의 호출이 많이 일어날시 성능 저하가 일어나기 때문이다. 생성된 MIPS R3000, ARM9 컴파일러로 JPEG 인코더를 컴파일하여 생성된 코드는 실행시 11.9kB의 bmp파일을 읽어 1.66kB의 JPEG 파일을 생성한다. 그림 11은 생성된 JPEG 파일을 보인다.

생성된 컴파일러에 의해 컴파일된 코드 수행으로 얻어진 JPEG 이미지가 오류 없이 출력되는 것을 확인함으로써 생성된 MIPS R3000과 ARM9 컴파일러의 정확한 동작을 검증하였다.

Ⅵ. 결론 및 추후과제

본 논문은 SMDL로 기술로부터 seamless 컴파일러 생성 시스템인 SRCC의 구축에 대해 기술하였다. SRCC는 SMDL의 행위 정보와 레지스터 사용 기술로부터 컴파일러 생성을 위한 정보를 추출하며 high-level SSA form을 사용한 SMDL-라이브러리 맵핑으로 머쉰과 컴파일러 사이에 생기는 의미적 차이를 최소화하였다. 또한, one-to-many 맵핑으로 적은 인스트럭션으로 많은 트리 패턴 맵핑이 가능하게 하였으며 many-to-one 맵핑으로 실행 성능 향상이 가능하도록 하였다. MIPS R3000과 ARM9 컴파일러의 생성과 JPEG 인코더를 통한 검증으로 생성된 컴파일러의 정확성을 검증하였다. 생성된 컴파일러는 기존 컴파일러와 비교 시 MIPS R3000과 ARM9 컴파일러에 대하여 각 94.2%, 84.9%의 실행 성능을 보였다. 하지만 프로세서, 시뮬레이터 생성 시스템인 SMDL을 이용해 seamless 생성에 기반 한 빠른 컴파일러 생성시간을 가지는 점에서 의의가 있다.

추후 과제는 생성된 컴파일러를 위한 프로파일러, SMDL 기술로부터 프로세서 의존적인 정보 추출을 통한 타겟 프로세서에 최적화되고 메뉴얼 포팅된 컴파일러 수준의 성능을 갖는 컴파일러의 생성이다.

참 고 문 헌

〔1〕 R. Leupers, "Compiler Design Issues for Embedded Processors", IEEE Design & Test of Computers, Vol. 19, No. 4, pp. 51-58, July-Aug. 2002.

〔2〕 M. Jain, M. Balakrishnan, and A. Kumar, "ASIP Design Methodologies : Survey and Issues", in Proc. IEEE/ACM Int. Conf. VLSI Design. (VLSI 2001), pp. 76-81, Jan. 2001.

〔3〕 C. Fraser and D. Hanson, *A Retargetable C Compiler : Design and Implementation*, Benjamin/ Cummings, 1995.

〔4〕 C. Fraser and D. Hanson, "The lcc 4.x Code-Generation Interface", Microsoft Research, 2003.

〔5〕 C. Fraser, R. Henry, and T. Proebsting, "BURG - Fast Optimal Instruction Selection and Tree Parsing", ACM SIGPLAN Notices, Vol. 27, No. 4, pp. 68-76, April 1992.

〔6〕 R. Leupers and P. Marwedel, "Retargetable Generation of Code Selectiors from HDL Processor Models", in Proc. European Design & Test Conf., 1997.

〔7〕 J. V. Praet, D. Lanneer, G. Goossens, W. Geurts, and H. De Man, "A Graph Based Processor Model for Retargetable Code Generation", in Proc. European Design & Test Conf., 1996.

〔8〕 D. Lanneer et al, "CHESS : Retargetable Code Generation for Embedded DSP Processors", in P. Marwedel and G. Goosens, ed., *Code Generation for Embedded Processors*, pp. 85-102, Kluwer Academic Publishers, 1995.

〔9〕 A. Hoffmann et al, "A Novel Methodology for the Design of Application-Specific Instruction Set Processors(ASIPs) Using a Machine Description Language", IEEE Trans. Computer-Aided Design, Vol. 20, No. 11, pp. 1338-1354, Nov. 2001.

〔10〕 J. Ceng, M. Hohenauer, R. Leupers, G. Ascheid, H. Meyr, and G. Braun, "C Compiler Retargeting Based on Instruction Semantics Models", in Proc. Design Automation and Test in Europe, pp. 1150-1155, 2005.

〔11〕 J. Ceng, W. Sheng, M. Hohenauer, R. Leupers, G. Ascheid, H. Meyr, and G. Braun, "Modeling Instruction Semantics in ADL Processor Descriptions for C Compiler Retargeting". Int. Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS), 2004.

〔12〕 S. Bashford, U. Bieker, B. Harking, R. Leupers, P. Marwedel, A. Neumann, and D. Voggenauer, "The MIMOLA Language Version 4.1", Technical report, University of Dortmund, 1994.

〔13〕 A. Fauth, J. Van Praet, and M. Freericks, "Describing Instructions Set Processors using nML", in Proc. European Design & Test Conf., Paris (France), pp. 503-507, Mar. 1995.

〔14〕 ACE - Associated Compiler Experts, The COSY Compiler Development System, http://www.ace.nl

〔15〕 조재범, 유용호, 황선영, "임베디드 프로세서 코어 자동생성 시스템의 구축", 한국통신학회논문지, 30권 6A호, pp. 526-534, 2005년 6월.

〔16〕 H. Emmelmann, F.-W. Schröer, and R. Landwehr, "Beg - A Generator for Efficient Back Ends", in Proc. SIGPLAN Conf. Programming Language Design and Implementation, pp 227-237, 1989.

〔17〕 B. Alpern, M. Wegman, and F. Zadeck, "Detecting Equality of Value in Programs", in Conf. Record of the 15th ACM Symposium on Principles of Program Languages, ACM, New York, pp. 1-11, January 1988.

〔18〕 R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and F. Zadeck, "Efficiently computing static single assignment form and the control dependence graph", ACM Trans. Program. Lang. Syst., Vol. 13, No. 4, pp. 451-490, Oct. 1991.

〔19〕 W. Yang, S. Horwitz, and T. Reps, "Detecting Program Components with Equivalent Behaviors", Tech. Rep. 840. Dept. of Computer Science, Univ. of Wisconsin at Madison, Madison, Apr. 1989.

〔20〕 G. Kane and J. Heinrich, *MIPS RISC Architecture*, Prentice Hall, 1992.

〔21〕 S. Furber, *ARM System-on-chip Architecture*, Addison-Wesley, 2000.

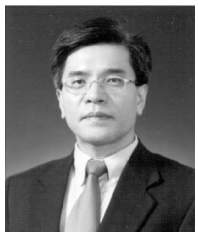
이 성 래 (Sung-Rae Lee) 준회원

2006년 7월 서강대학교 전자공학과 졸업

2006년 8월~현재 서강대학교 전자공학과 석사과정

<관심분야> Embedded 시스템을 위한 고성능 프로세서 및 optimizing compiler 설계

황 선 영 (Sun-Young Hwang) 정회원



1976년 2월 서울대학교 전자공학과 졸업

1978년 2월 한국과학원 전기 및 전자공학과 공학석사 취득

1986년 10월 미국 Stanford 대학교 전자공학 박사학위 취득

1976년~1981년 삼성 반도체(주) 연구원, 팀장

1986년~1989년 Stanford대학 Center for Integrated Systems 연구소 책임 연구원 및 Fairchild Semiconductor Palo Alto Research Center 기술자문

1989년~1992년 삼성전자(주) 반도체 기술자문

2002년 4월~2004년 3월 서강대학교 정보통신대학원장

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD시스템, Com. Architecture 및 DSP System Design 등