VLIW형 마이크로컨트롤러를 위한 최적화 컴파일러의 구현 *

홍 승표 ○ 문 수묵

서울대학교 전기공학부

An Optimizing Compiler for VLIW Microcontrollers

Seung-Pyo Hong Soo-Mook Moon

School of Electrical Engineering,
Seoul National University

요 약

90년대 중반이후 고성능의 프로세서들은 성능향상을 위해 명령어 수준의 병렬성을 이용하고 있다. 특히 실행화일의 호환성을 고려할 필요가 없는 마이크로컨트롤러에서는 같은 하드웨어로 더 많은 함수유닛을 가질 수 있는 VLIW 구조가 널리 사용된다. 이러한 VLIW 형의 마이크로컨트롤러에서는 병렬성을 추출하는 역할이 전적으로 소프트웨어에 있으므로 컴파일러가 성능향상에 매우 큰 영향을 미치게 된다. 본 논문에서는 마이크로컨트롤러의 구조와 그룹짓기 조건을 분석하고 선택 스케쥴링과 소프트웨어 파이프라이닝을 이용한 VLIW 형 마이크로컨트롤러용 최적화 컴파일러를 구현하고 그 성능을 측정한다.

1 서 론

90년대 중반이후 영상 압축 표준의 제정 및 정보통신망 보급과 멀티미디어 산업의 발전으로 고성능의 멀티미디어용 마이크로컨트롤러가 그 필요성을 더해가고 있다. 이러한 고성능의 마이크로컨트롤러들은 한 클럭 사이클에 여러 개의 명령어를 동시에 실행시키는 명령어 수준 병렬성(Instruction Level Parallelism, ILP)을 이용하는 추세이다. 명령어 수준 병렬성을 이용하는 방법은 병렬성을 찾는 주체에 따라 수퍼스칼라 방식과 VLIW(Very Long Instruction Word) 방식으로 나뉘게 되는데 수퍼스칼라 방식은 프로세서가 프로그램의 실행시간에 동시 수행가능한 명령어를 찾아주고 VLIW 방식은 소프트웨어가 프로그램의 컴파일 시간에 그러한 명령어를 찾아준다.

마이크로컨트롤러는 그래픽처리나 실시간 컨트롤 등 응용 프로그램이 정해진 특수목적으로 쓰이므로 실행 화일의 호환성에 대한 고려가 일반 마이크로프로세서보다 적다. 따라서 독립적인 명령어 코드를 가질 수 있으므로 수퍼스칼라보다 하드웨어의 부담이 적은, 혹은 같은 하드웨어로 보다 많은 함수유닛을 가질 수 있는 VLIW 방식이 상용화되고 있다. Texas Instrument사의 TMX320C6x[2]나 Chromatic사의 Mpact , MicroUnity사의 MediaProcessor[3]등이 여기에 해당한다.

VLIW 구조에서는 컴파일러가 순차적 코드에서 독립적인 명령어를 추출하게 된다. 따라서 이러한 VLIW 방식의 마이크로컨트롤러의 성능은 하드웨어 뿐만 아니라 병렬성을 추출하는 컴파일러의 성능에 의해서도 크게 좌우된다. 본 논문에서는 현재 VLIW 형 프로세서에서 적용되고 있는 최적화 기법들을 마이크로컨트롤러에 적용하여 보았다.

본 논문은 2장에서 VLIW 형 마이크로컨트롤러의 구조에 대하여 분석하고 3장에서는 이러한 명령어를 찾는 스케쥴링 방법을 설명하며 4장에서 실험환경을, 5장에서 결론 및 향후 과제를 기술한다.

*This study was supported by the academic research fund of Ministry of Education, Republic of Korea, through Inter-University Semiconductor Research Center(ISRC 97-E-2102) in Seoul-National University

2 마이크로컨트롤러의 구조

우리의 최적화 컴파일러는 VLIW 칩의 사이클 타임을 줄이기 위한 버스 구조의 마이크로컨트롤러를 타겟으로 한다. 여기에서 언급하는 버스는 CPU 내부에서 각 함수유닛과 레지스터 파일 등을 연결하는 버스를 의미한다. 어셈블리어 레벨에서 보이는 각 함수 유닛의 입력과 출력이 레지스터인 RISC 구조와는 달리, 버스구조에서 각 함수 유닛의 입력과 출력은 지정된 버스가 된다.

이러한 구조에서의 명령어는 RISC에서의 명령어보다 더 작은 연산을 하므로 미소 명령어(Micor Operation)로 취급할 수 있다. 예를 들어 RISC구조에서 'add r3, r1, r2'와 같은 명령어는 버스구조에서 'b1=read(r1); b2=read(r2); b3=add(b1,b2); r3=write(b3);'(b1,b2,b3는 각각 레지스터 파일의 read port1, read port2, ALU의 출력단에 연결된 버스)처럼 쓰여져 어셈블리어 레벨에서 각 버스와 파이프라이닝 단계가 프로그래머에게 보이게 된다. 또한 오퍼랜드로 이러한 버스를 쓰므로 레지스터보다 한 명령어의 폭이 작아져, 같은 명령어 폭으로 더 많은 함수유닛을 쓸 수 있다는 장점이 있다.[1] 본 연구의 실험에서는 64비트에 8개의 명령어가 동시에 실행 가능한 마이크로컨트롤러를 기반으로 하였다. 반면에 함수유닛의 출력이 지정된 버스로 제한된다는 단점이 있다.

함수유닛과 버스간의 데이터 패스는 아래 그림과 같다. *

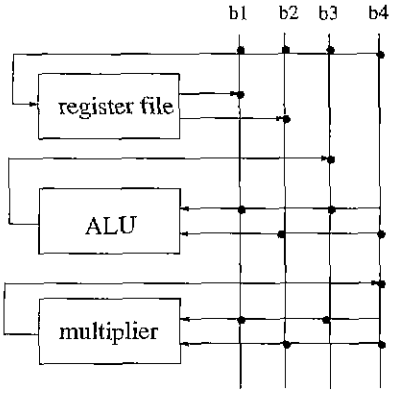

그림 1: 버스구조 마이크로컨트롤러의 데이터 패스

각 함수유닛 즉 ALU, 곱셈기, Shifter/Rotater, Unary 유닛, 메모리 I/O 유닛, 3 포트 레지스터 파일이 하나씩 존재하며 지정된 버스에 연결되어있는 모델을 상정하였다. 이 함수유닛은 지정된 버스에 결과를 출력하므로 동시에 8개의 명령어(4 연산 명령어 + 1 메모리 I/O 명령어 + 3 레지스터 파일 접근 명령어)가 실행이 가능하다.

*편의상 3개의 함수유닛과 4개의 버스만을 표시하였다

3 VLIW 스케쥴링 컴파일러

본 연구에서는 비환형 그래프(Acyclic Graph)에서는 선택 스케쥴링 기법 [4]을 이용하고 환형 그래프(Cyclic Graph)에서는 소프트웨어 파이프라이닝 기법의 일종인 EPS(Enhanced Pipelining Scheduling) 기법으로 이용하여 동시에 실행가능한 명령어를 찾는다. 이 기법들은 원래 RISC 형 프로세서를 기반으로 많이 연구된 기법으로써 전술한 마이크로컨트롤러의 제한에 맞게 변형하여 적용한다.

3.1 비환형 그래프 : 선택 스케쥴링 기법

선택 스케쥴링 기법은 DAG(Directed Acycleic Graph)의 처음부분(Root †)에 빈 노드를 생성하고 이후의 모든 경로에서 데이터 의존성 (Data Dependency)이 없는 명령어들을 모으고 그 중에서 자원제한 (Resource Constraint)에 맞는 명령어들을 선택하여 동시에 실행 가능한 하나의 VLIW 명령어 그룹을 만들어낸다. 이 두 조건에 해당되는 명령어가 함수유닛의 수보다 많을 때 어떤 명령어를 먼저 선택할 것인가는 빈 노드와 그 명령어 사이의 거리와 분기문의 수 그리고 실행빈도‡ 등에 의해 결정된다. 우선 분기문을 통하지 않은 비추측 명령어(Nonspeculative Operation)가 먼저 선택된다. 그 다음 추측 명령어(Speculative Operation)들에서 다른 경로에서 그 명령어의 대상(Target)이 유효(live)하지 않는 명령어 중 빈 노드에 가까운 명령어가 선택된다.

선택 스케쥴링 기법에서는 추측 명령어를 선택할 때, 다른 경로에서 대상이 유효하더라도 재명명(Register Renaming)을 통하여 복사자를 남기고 분기문을 거슬러 올라갈 수 있으나, 버스구조의 마이크로컨트롤러에서는 각 함수유닛의 출력이 할당된 버스에 고정되어 있고 버스를 재명명하지 못하므로 다른 경로에서 대상이 유효할 때는 분기문을 거슬러 올라가지 못한다.

3.2 환형 그래프 : EPS 기법

환형 그래프 혹은 루프에 대해서는 소프트웨어 파이프라이닝 기법의 일종인 EPS 기법으로 스케쥴링한다. 이 기법은 루프의 적절한 간선(edge)을 끊어 환형그래프를 DAG으로 변환하고 이 DAG에서 적절한 방법으로 서로 데이터 의존성이 없는 명령어들을 하나로 묶는 일을 반복한다. 여기에서는 전술한 선택 스케쥴링 기법이 DAG에서 그러한 명령어들을 찾는데 사용되었다. 이러한 작업의 결과로 나중에 스케쥴된 명령어 그룹에는 루프의 후간선(backedge)을 넘어 다음에 반복되어 수행될 명령어들도 포함되게 된다.

이러한 EPS 기법은 소프트웨어 파이프라이닝의 문제를 DAG 스케쥴링 문제로 치환하여 DAG 스케쥴링을 반복함으로써 파이프라인의 시작코드, 커널, 종료코드를 가진 파이프라인된 루프가 자연히 생성된다는 점이 장점이다.[5]

†Topological Sort에서 첫 노드

‡본 연구에서는 실행빈도는 고려하지 않았다.

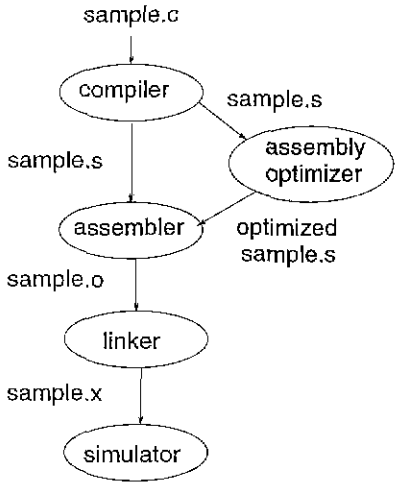

그림 2: 실험환경

4 실험환경

그림 2 는 프로그램의 개발환경의 단계이고 아래 프로그램은 그 단계들에서 나오는 결과들로, C 소스코드((a) sample.c)와 컴파일한 후의 어셈블리코드 ((b) sample s) 그리고 최적화기를 통한 병렬화된 어셈블리 코드 ((c) optimized sample.s)이다. C 소스코드는 배열을 접근하는, 한 응용프로그램의 최내각 루프이다. 이의 스케쥴 결과를 보면 선택 스케쥴과 소프트웨어 파이프라이닝을 통해 미래에 반복될 명령어가 동시에 수행이 가능하도록 추출되어 있으며 이로인해 딜레이 슬롯(Delay Slot)이 효율적으로 채워져, 원래 순차 코드에서는 21 사이클이 걸리던 루프 커널이 9 사이클로 줄어들었음을 확인할 수 있다.

```
for(j=0; j<i+1; j++ )
    buffer[j] = offset + j;
```

(a) sample.c

```
L8:
b1=read(r9);      | DA=wrAd(b3),b5=byte(b2);
b2=read(r11);     | b1=read(1);
b3=add(A,B);      | b2=read(r9);
b1=read(-64);     | b3=add(b1,b2);
b2=read(r9);      | r9=write(b3);
r59=write(b3);    | b1=read(r10);
b3=add(b1,b2);    | b2=read(r9);
b1=read(r7);      | b3=sub(b2,b1);
b3=add(b1,b3);    | ifnot(sgncmp),branch(L8);
b2=read(r59);     | ; // 1st delay slot
                  | ; // 2nd delay slot
```

(b) sample.s

```
b1=read(r9),b2=read(r11);
b3=add(b1,b2),b1=read(-64),b2=read(r9);
r59=write(b3),b3=add(b1,b2),b1=read(r7),b2=read(r59);
L8:
b3=add(b1,b3),b1=read(1);
DA=wrAd(b3),b5=byte(b2),b2=read(r9);
b3=add(b1,b2),b1=read(r10);
r9=write(b3);
b2=read(r9);
b3=sub(b2,b1),b1=read(r9),b2=read(r11);
ifnot(sgncmp),branch(L8);
b3=add(b1,b2),b1=read(-64),b2=read(r9);
                  // 1st delay slot
r59=write(b3),b3=add(b1,b2),b1=read(r7),
b2=read(r59);   // 2nd delay slot
```

(c) optimized sample.s

5 결론 및 향후 연구방향

현재까지 ILP 프로세서를 위한 컴파일러에 사용되는 많은 기법들이 연구되어 왔다. 본 연구에서는 이러한 기법들이 마이크로컨트롤러에 유효한지 알아보기 위해 그 중 선택 스케쥴링 기법과 EPS 기법을 적용하여 성능을 측정하였다. 마이크로컨트롤러의 하드웨어 특성상, 이러한 기법들의 제한점이 있음에도 불구하고 최내각 루프에 대하여 괄목할 만한 성능향상이 이루어짐을 확인하였다. 앞으로는 함수유닛이 확장될 경우에 위한 스케쥴링 기법과 자원 사용이 불규칙한 함수유닛에 대한 스케쥴링 기법이 보완이 되어야 할 것이다

참고 문헌

[1] Arcobel Graphics. "The Imagine Documentation & User Manual"

[2] Robert J. Gove. "The Multimedia Video Processor(MVP) : An Architecture for Advanced DSP Applications" *Proceedings of ICSPAT 94, pp.854-859.*

[3] Microprocessor Report "MicroUnity Lifts Veil on MediaProcessor" pp.11-18, volume 9, no 14.

[4] S.-M. Moon and K. Ebcioğlu. "Parallelizing Nonnumerical Code with Selective Scheduling and Software Pipelining" *to appear in ACM Transactions on Programming Languages and Systems.*

[5] K. Ebcioğlu. "VLIW Compilation Techiniques in a Superscalar Environment" *Proceedings of the SIGPLAN 1994 conference on Programming Language Design and Implementation.*

[6] HoeMok Chung, HanSaem Yun and Soo-Mook Moon. "ILP Scheduling for UltraSPARC Processor" *Proceedings of The 24th KISS FallConference.*