VLIW 프로세서를 위한 Swing Modulo Scheduler 구현

신장섭*, 한상준*, 정현균*, 안민욱**, 윤종희***, 백윤흥*
*서울대학교 전기정보공학부
**삼성전자
***영남대학교 컴퓨터공학과
e-mail : jsshin, sjhan, hgjung@sor.snu.ac.kr, minwook.ahn@samsung.com, youn@yu.ac.kr, ypaek@snu.ac.kr

Implementing Swing Modulo Scheduler for VLIW Processor

Jangseop Shin*, Sangjun Han*, Hyungyun Jung*, Minwook Ahn**, Jonghee M. Youn***, Yunheung Paek*
*Dept. of Electrical and Computer Engineering, Seoul National University
**Samsung Electronics
***Dept. of Computer Engineering, Yeungnam University

요 약

하드웨어가 해저드(hazard) 검출을 지원하지 않는 멀티이슈 VLIW 프로세서의 성능을 높이기 위해서는 컴파일러가 명령어 의존성과 하드웨어 자원의 제약을 지키는 범위 안에서 최대한 명령어수준 병렬성(ILP)을 활용하는 것이 중요하다. 기본 블록(basic block) 스케쥴링은 Branch 등 제어 흐름(control flow)의 경계를 넘어선 스케쥴링을 행하지 않아 그 효과가 제한적이다. 소프트웨어 파이프라이닝(software pipelining)은 루프(loop)의 경계를 허물어 여러 반복(iteration)의 명령어가 동시에 수행되도록 하는 것으로 모듈로 스케쥴링(modulo scheduling)은 그 중에 한 범주의 스케쥴링 기법들을 일컫는다. 본 연구에서는 그 중 한가지인 스윙 모듈로 스케쥴러(swing modulo scheduler)[1]를 구현하여 그 효과를 알아보고자 한다.

1. 서론

현재의 프로세서들은 성능을 높이기 위해 여러 개의 명령어의 연산을 동시에 수행하는데 필요한 하드웨어를 갖추고 있다. 저전력이 중요한 지표로 여겨지는 분야에서는 해저드 검출기능을 포함한 복잡한 하드웨어가 요구되는 수퍼스칼라(superscalar) 프로세서보다는 하드웨어가 간단한 대신 컴파일러가 프로그램의 올바른 수행을 보장해야 하는 VLIW 프로세서가 많이 사용된다. VLIW 프로세서의 성능을 최대로 하기 위해서는 컴파일러의 명령어 스케쥴링 단계에서 명령어 의존성과 하드웨어 자원의 제약을 지키는 범위에서 주어진 명령어 순서를 재조정하여 VLIW 의 이슈 슬롯을 최대한 채워야 한다.

기본 블록 스케쥴링은 명령어 수행 흐름을 바꾸는 명령어가 존재하는 경우 그 이전과 이후의 명령어를 같이 스케쥴 할 수 없다. 기본 블록 내에 병렬성을 이끌어낼 명령어 자체가 적거나, 각 명령어의 수행시간이 길고 명령어간 의존 관계가 복잡해 병렬성을 충분히 이끌어낼 수가 없어 하드웨어 자원 효율이 떨어지는 경우 스케쥴링의 효과가 작을 수 밖에 없다.

이 한계를 넘어 기본 블록들 간의 명령어를 함께 스케쥴링하는 기법 중 루프의 여러 반복의 명령어를 함께 스케쥴링 하는 소프트웨어 파이프라이닝 기법이 있다. 많은 프로그램에서 수행 시간 대부분을 루프에서 소요하므로 이 기법을 활용하면 성능을 크게 향상시킬 수 있다. 본 연구에서는 소프트웨어 파이프라이닝 기법들 중 널리 쓰이는 모듈로 스케쥴링 기법의 한 종류인 스윙 모듈로 스케쥴러를 VLIW 프로세서를 위한 컴파일러에 추가하여 성능 향상을 살펴보았다. 2 장에서 소프트웨어 파이프라이닝의 목표와 모듈로 스케쥴링에 대해 소개하고, 3 장에서 본 연구에서 구현한 스윙 모듈로 스케쥴러에 대해 설명하겠다. 4 장에서는 실험 환경과 결과에 대해 논하고, 5 장에서 결론 및 향후 계획을 서술하겠다.

2. 배경

소프트웨어 파이프라이닝에서 몇 프로세서 사이클마다 새로운 반복이 시작되는가를 시작간격(initiation interval, II)이라 한다. 루프의 수행시간은 II 와 루프가 도는 횟수의 곱에 비례하기 때문에 II 를 최소로 할수록 수행시간이 줄어들게 된다. II 를 최소화 하는 것 외에 동시에 겹치는 레지스터 라이브 구간(register live range)의 최대 개수를 의미하는 레지스터 압박(register pressure)을 최소화 하는 것도 중요하다. 소프트웨어 파이프라이닝을 행하면 여러 루프 반복이 겹치면서 레지스터의 라이브 구간이 겹치게 되어 레지스터 할당 시 더 많은 물리적 레지스터를 필요로 할 가능성이 커진다. 만약 물리적 레지스터가 모자라게 되면

일정 수의 레지스터를 메모리에 저장(spill) 시키거나 II 를 늘려서 스케쥴링을 다시 해야 하므로 루프의 수행 시간이 길어지게 된다. 최적의 스케쥴을 찾는 것은 NP-complete 로 알려져 있기 때문에 적절한 경험적 방법을 사용하여 컴파일 시간을 크게 소요하지 않으면서 최적에 가까운 스케쥴을 구하는 것도 소프트웨어 파이프라이닝의 중요한 목표이다.

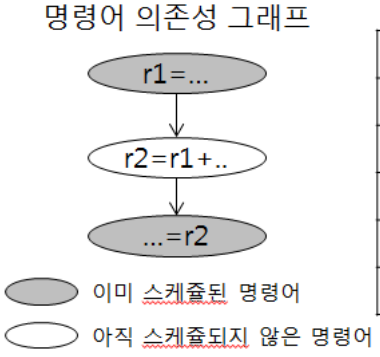
모듈로 스케쥴링은 소프트웨어 파이프라이닝의 한 방법으로 루프의 한 반복에 대한 스케쥴을 구하되 그 스케쥴대로 II 의 사이클마다 루프가 수행될 때, 하드웨어 자원 제약과 명령어 의존성이 지켜질 수 있도록 하는 것이다. 이를 위해 모듈로 스케쥴링에서는 II 사이클 동안의 사이클 별 하드웨어 자원 사용상태를 테이블로 나타내고 여기에 명령어를 하나씩 배치하며 스케쥴링을 행한다.

모듈로 스케쥴링에서 어떤 명령어를 먼저 스케쥴할 것인가와, 각 명령어를 배치할 때 스케쥴링 제약을 만족하는 여러 위치 중 어느 곳을 선택할 것인가에 따라 II 와 레지스터 압박이 달라진다. 한정된 시간에 스케쥴을 구해야 하므로 모든 가능성에 대해 시도해 볼 수가 없기 때문에 위 사항들에 대한 선택 기준과 스케쥴링 재시도를 어느 단계부터 어느 정도 허용할 것인가가 결과에 큰 영향을 미친다.

어떤 명령어를 먼저 스케쥴 할 것인가는 명령어 의존성 그래프를 어떤 방식으로 방문(visit)하며 우선적으로 스케쥴할 명령어를 정할지와 방문 후보가 여럿 있을 경우 무엇을 기준으로 스케쥴 우선순위를 정할지에 대한 선택으로 정해진다[2].

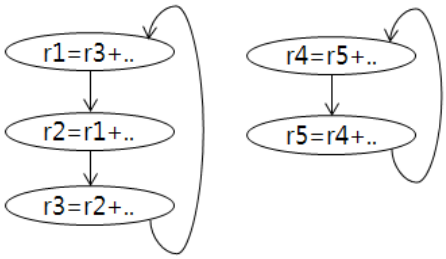
3. 스윙 모듈로 스케쥴링

스윙 모듈로 스케쥴링의 가장 핵심적인 원리는 명령어를 의존관계에 있는 명령어에 최대한 가깝게 배치하여 레지스터 라이브 구간을 최소화 하는 것이다. 어떤 명령어를 스케쥴 할 때 이미 스케쥴 된 명령어들과의 의존관계를 고려해야 하는데, 이미 스케쥴 된 명령어 중 선행자(predecessor)만 있거나 후속자(successor)만 있다면 해당 명령어에 최대한 가깝게 스케쥴하면 되지만, 두 종류가 모두 있다면 어느 한 쪽에 가깝게 스케쥴했을 때 다른 한 쪽과는 멀어져 라이브 구간이 길어지게 된다. 예를 들면 그림 1 에서 두 번째 명령어를 선행자에 가깝게 스케쥴하면 r1 의 라이브 구간은 짧아지지만 r2 의 라이브 구간은 길어



cycle	instruction
0	r1=...
1	
2	
3	
4	...=r2

(그림 1) 라이브 구간 최소화가 어려운 경우



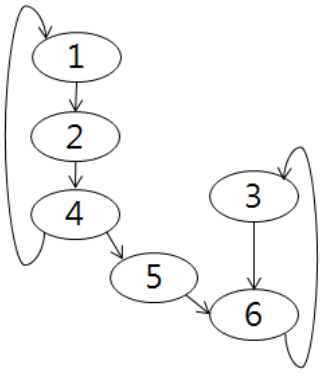
(그림 2) 고리를 형성하는 의존성 그래프

지게 된다. 따라서 스윙 모듈로 스케쥴러에서는 이러한 상황을 최대한 피하여 전체적인 레지스터 라이브 구간을 최소화 한다.

그림 2 에서와 같이 어떤 명령어로부터 다음 루프의 명령어로 의존 관계가 있어 의존성 그래프가 고리(cycle)를 포함하게 되는 경우, 앞서 언급한 상황을 한 번은 마주칠 수밖에 없다. 이 경우 고리에 속한 명령어를 차례대로 스케쥴하지 않는다면 앞서 언급한 상황이 두 번 이상 발생할 수 있기 때문에 스윙 모듈로 스케쥴러에서는 이러한 고리를 기준으로 명령어의 그룹을 만들어 같은 그룹에 속한 명령어들은 한번에 스케쥴한다.

의존성 그래프에 고리가 여러 개인 경우에는 의존성 관계의 사슬의 길이가 길수록 높은 우선순위를 부여한다. 연속된 반복 사이에 형성된 의존 관계로 인해 다음 루프의 반복은 이전 루프의 선행자의 수행이 끝나야 시작될 수가 있다. 만약 그림 2 에서 길이가 더 긴 왼쪽 고리에 속한 명령어가 바로바로 수행되지 못한다면 이는 II 의 증가에 직접적인 영향을 미친다. 반면 오른쪽 고리에 속한 명령어는 바로바로 수행이 되지 못한다고 해도 왼쪽 고리의 명령어가 스케쥴 되고 남는 공간에만 스케쥴 할 수 있다면 II 에 직접적인 영향을 주지는 않는다. 이러한 이유로 사슬의 길이가 긴 고리에 속한 명령어를 우선적으로 스케쥴 하게 된다.

첫 번째 고리를 선택한 다음부터는 고리들 사이에 끼인 명령어가 있는 경우 해당 명령어를 다음 고리보다 앞서 스케쥴하도록 한다. 그림 3 과 같은 의존성 그래프가 있을 때 두 개의 고리를 먼저 스케쥴하고 난 후 5 번 명령어를 스케쥴한다면 선행자와 후속자 모두가 이미 스케쥴 된 상황이 되기 때문에 두 번째 고리보다 우선적으로 5 번 명령어를 스케쥴해야 한다.



(그림 3) 고리 사이에 명령어가 끼인 경우

그룹 간의 순서를 정한 후에는 각 그룹 내의 명령어들 간의 순서를 정한다. 먼저 고리를 없애기 위해 후방 경로(back edge) 하나를 무시하고, 크리티컬 경로(critical path)의 가장 하위 명령어부터 선행자들을 방문해 간다. 크리티컬 경로란 의존성 사슬이 가장 긴 경로를 말한다. 선행자가 여럿 있는 경우에도 역시 크리티컬 경로 상의 명령어를 먼저 선택하고, 그마저 같은 경우에는 미리 계산한 MOB(mobility) 값이 작은 명령어를 먼저 선택한다. MOB 값은 의존성 그래프에서 어떤 명령어를 가장 늦게 스케쥴 할 수 있는 사이클과 가장 빨리 스케쥴 할 수 있는 사이클의 차로, 앞서 명령어 그룹의 순서를 정할 때와 마찬가지로 MOB 값이 클수록 나중에 스케쥴 하더라도 성공할 가능성이 크다. 선행자들을 모두 방문하면 이미 방문한 노드의 후속자들을 방문하고 이 과정을 반복하여 그네와 같은 형태로 그래프를 방문한다. 이를 통해 명령어를 선행자나 후속자에 최대한 가깝게 스케쥴하여 레지스터 라이브 구간을 최소화 할 수 있다.

스윙 모듈로 스케쥴러에서 짧은 시간에 비교적 좋은 스케쥴을 구할 수 있는 이유는 스케쥴링에 유리한 명령어 스케쥴 순서를 처음 한 번만 구하고 스케쥴에 실패할 경우 II 를 늘려 스케쥴링만 다시 행하기 때문이다.

4. 실험 환경 및 결과

스윙 모듈로 스케쥴러는 VLIW 프로세서의 LLVM 기반 컴파일러에 모듈을 하나 추가하는 방식으로 구현하였다. 모듈로 스케쥴러가 동작하는 단계는 PHI 명령어와 같은 유사코드(pseudo-code)가 제거되고 레지스터 할당이 이루어지기 전이다. 이는 모듈로 스케쥴링 수행 후에는 코드 수정이 어렵기 때문에 유사코드가 제거된 후에 실제 머신코드만을 가지고 스케쥴링이 이루어지도록 하고, 레지스터 할당 전에 가상 레지스터를 사용하여 레지스터간의 역의존성(anti-dependency)을 최대한 제거하여 스케쥴 결과가 최대한 짧아지도록 하기 위한 것이다. 레지스터 할당에 문제가 없도록 하기 위해서는 스케쥴러 구현 시 동시에 겹치는 레지스터 라이브 구간의 최대치를 확인해야 하는데 이는 향후 구현에 추가할 계획이다.

모듈로 스케쥴러의 대상 루프는 함수호출이 없고 if-else 와 같은 조건문이 없는 가장 안쪽(innermost) 루프이다. 조건문의 경우 모듈로 스케쥴링을 적용하는 기술이 존재하지만 구현된 스케쥴러에는 현재 적용되지 않았다.

<표 1> 실험 결과

Loop	MII	II Before	II After
FFT2	24	25	25
HPF	17	18	17
FIR	29	39	33
histo1	50	239	122
histo2	43	49	43
gaussian1	23	25	23
gaussian3	14	23	15

표 1 은 멀티미디어 어플리케이션의 주요 루프에 대해 기본 블록 스케쥴러인 리스트 스케쥴러와 본 연구에서 구현한 스윙 모듈로 스케쥴러의 스케쥴링 결과에서 II 를 비교한 것이다. 리스트 스케쥴링으로도 MII 에 가까운 값을 구할 수 있었던 루프들에 대해서는 모듈로 스케쥴링으로 성능 이득을 기대할 수가 없었지만 MII 와 리스트 스케쥴링의 결과에 큰 차이는 보이는 루프들에 있어서는 최대 50%의 성능 이득을 볼 수 있었다. 멀티미디어 어플리케이션 특성 상 루프에서 대부분의 시간이 소요되기 때문에 이는 큰 성능 이득이라 할 수 있다.

5. 결론 및 향후 계획

멀티이슈 하드웨어의 자원을 효과적으로 이용하기 위해서는 명령어 스케쥴러의 역할이 중요하다. 소프트웨어 파이프라이닝 기법은 프로그램 수행시간의 대부분을 차지하는 루프의 여러 반복을 겹쳐서 수행시켜 프로그램 전체의 수행을 빠르게 할 수 있다. 본 연구에서는 대표적인 소프트웨어 파이프라이닝 기법인 스윙 모듈러 스케쥴러를 구현하여 VLIW 컴파일러에 적용해 보았고 실험을 통하여 어느 정도 그 성능을 확인하였다. 향후에는 메모리 명령어 의존성 분석 결과를 활용하여 의존성 제약을 완화하거나 레지스터의 역의존성을 제거하여 스케쥴러의 성능을 개선시키고, if-else 등의 조건문이 포함된 루프, while 루프 등 다양한 루프에 적용 가능하도록 하는 등 추가 기능을 구현해볼 계획이다.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2012-0000470), 중소기업청에서 지원하는 2012 년도 산학연공동기술개발사업(No. C0019562), 미래창조과학부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신) [No. 10047212, 1kB 이하 암호문 간의 연산을 지원하는 동형 암호 원천 기술 개발 및 응용 연구], 및 IDEC 의 지원을 받아 수행하였습니다.

참고문헌

[1] J. Llosa, et. al., “Swing Modulo Scheduling: A Lifetime-Sensitive Approach”, IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques, October 1996

[2] Y.N. Srikant, P. Shankar, “The compiler design handbook: optimizations and machine code generation”, CRC Press, 2007