IA-64를 위한 향상된 소프트웨어 파이프라인 명령어 스케줄링

이재목, 문수묵
서울대학교 전기컴퓨터 공학부
jaemok@altair.snu.ac.kr

Enhanced Pipeline Scheduling for IA-64

Jae-Mok Lee, Soo-Mook Moon
School of Electrical Engineering and Computer, Seoul National University

초록

인텔의 IA-64 프로세서는 명령어 수준의 병렬수행을 지원하는 EPIC (Explicitly Parallel Instruction Computing) 구조를 채택하고 있으며 컴파일러가 순차적 코드에서 병렬 수행이 가능한 독립적인 명령어들을 스케쥴링 하도록 되어있다. 본 논문에서는 IA-64 스케쥴링을 위해 향상된 파이프라인 스케쥴링 (Enhanced Pipeline Scheduling, EPS) 기법[1]을 적용한 결과를 소개한다. EPS는 루프수준의 병렬화를 위한 소프트웨어 파이프라이닝 (software pipelining)기법으로 전역 스케쥴링 (global scheduling) 기법을 기반으로 하고 있다. 우리는 IA-64 프로세서를 위한 공개소스 컴파일러인 ORC (Open Research Compiler)에 EPS를 구현하고 실제 프로세서인 Itanium에서 실험을 수행하였다. 상용 프로세서와 컴파일러에 구현과 튜닝을 하는 과정에서 얻은 경험을 소개하고 기존의 ORC 컴파일러와 비교하여 얻은 성능 향상을 보고하고 분석한다.

1. 소개

명령어 수준의 병렬성을 향상시키기 위한 명령어 스케줄 기법은 기법이 적용되는 범위에 따라 지역 스케줄과 전역 스케줄로 나누어진다. 전역 스케줄 기법은 다시 루프의 백에지 (back edge)를 포함하지 않는 Directed Acyclic Graph (DAG) 스케줄 기법과 루프의 백에지를 포함하는 소프트웨어 파이프라인 기법으로 나눌 수 있다.

소프트웨어 파이프라인 스케줄 기법은 루프의 각 iteration 간에 존재하는 병렬성까지 활용할 수 있도록 명령어를 배열함으로써 루프의 수행성능을 향상시키는 기법이다. 대표적인 소프트웨어 파이프라인 스케줄 기법으로는 모듈로 스케줄 기법이 있는데, 현재 대 다수의 컴파일러에서 사용하고 있는 기법이다. 하지만 모듈로 스케줄 기법을 적용할 수 있는 루프는 트리 형태의 흐름구조를 갖는 루프로 제한이 되어, 제어흐름도(Control Flow Graph) 상에서 predecessor 가 2개 이상인 베이직 블록을 포함하는 루프에 대해서는 적용이 불가능하다는 단점을 갖고 있다.

하지만 향상된 파이프라인 스케줄 기법[1]은 아무리 복잡한 제어흐름을 갖고 있는 루프라고 해도, 코드 이동을 이용한 명령어 스케줄을 통해 루프의 iteration 간에 존재하는 병렬성을 활용하여 성능을 향상 시킬 수 있다.

향상된 파이프라인 스케줄 기법은, 가상의 tree-VLIW 구조를 위해서 개발된 알고리즘으로 실제 컴퓨터에 구현되어 성능을 평가 받은 경우가 매우 적다. In-order superscalar 구조인 Sun의 UltraSPARC 을 위한 컴파일러에 구현된 것이 전부이다.

이 논문에서는 IA-64 용 컴파일러인 ORC 컴파일러 위에 구현한 향상된 파이프라인 스케줄 기법을 소개하고, 성능을 분석한다.

2. 관련된 연구

IA-64 는 EPIC 이라는 구조를 택함으로써 기존의 superscalar 구조와 달리 명령어 수준의 병렬성을 소프트웨어적인 방식으로만 활용할 수 있도록 하였다. 이에 따라 IA-64 용 컴파일러에서는 명령어 스케줄 기법이 필 수적이다. 이에 따라 여러 가지 전역 명령어 스케줄 기법이 IA-64용 컴파일러에

적용되었는데, Global Instruction Schedule[3] 과 Wavefront Schedule[4] 이 그것이다. 이 두 가지 기법은 모두 DAG 스케줄 기법으로 루프의 iteration을 넘는 병렬성을 찾을 수 없다.

또 다른 접근 방법으로서, Hyperblock 을 이용한 컴파일[5]을 들 수 있는데, 이 방법은, 제어흐름이 복잡한 루프에 대해서 조건 실행 명령(predicated instruction)을 이용하여, 베이직 블록들을 하나의 큰 hyperblock으로 만들어버림으로써 모듈로 스케줄이 가능하게 하는 방법이다. 이 방법은 모듈로 스케줄이 가능한 영역을 확장해주긴 하지만, 여전히 코드크기의 제약이 있으며, 자주 실행되지 않는 명령어들을 조건 실행 명령으로 만듦으로써 의미 없이 자원을 낭비하게 될 가능성이 크다.

3. 향상된 파이프라인 스케줄

보통의 DAG 스케줄은 스케줄을 하는 동안 고정된 DAG 를 사용한다. 반면에 향상된 파이프라인 스케줄은 스케줄의 범위가 계속 변한다. 즉 스케줄이 끝난 명령어들도 스케줄 후보에 포함시켜서 스케줄을 진행한다.

그림1 은 SPEC CPU2000 integer 벤치마크 중, 164.gzip 의 코드 중 일부를 나타낸다. 베이직 블록 1번이 루프 헤더이고, 한 싸이클씩 스케줄을 해나가는 과정이다. 베이직 블록 3의 Scheduling here 라고 써진 부분이 현재 스케줄을 하려고 하는 싸이클이고, 그 위에 굵은 글씨로 된 부분은 모두 이미 스케줄이 끝난 싸이클이다.

보통의 DAG 스케줄은 Scheduling here 라고 써진 부분 아래의 베이직 블록 3,4번에 있는 명령어들만은 code motion하여 스케줄 하지만, 향상된 파이프라인 스케줄은 3번과 4번 베이직 블록을 지나서 루프의 백-에지를 통과하여 베이직 블록 1번과 2번의 이미 스케줄 된 명령어들까지 code-motion 하여 스케줄 한다.

이 예제에서는 베이직 블록 1번안의 sxt4 GTN748 = GTN724, adds GTN717 = #-1, GTN717 이 두 개의 명령이 베이직 블록 3으로 스케줄이 되면서 이 루프의 수행에 걸리는 싸이클의 수를 하나 줄일 수 있다.

향상된 파이프라인 스케줄은 위에 설명했듯이 스케줄을 하는 동안 스케줄 된 결과가 적용된 DAG에 대해서 반복적으로 DAG 스케줄을 적용하는 방식으로 이루어진다. 이때 사용하는 DAG 스케줄 기법은 선택적 스케줄[2] 로서 rename, forward substitution, 등의 방식으로 non-true dependence 를 극복할

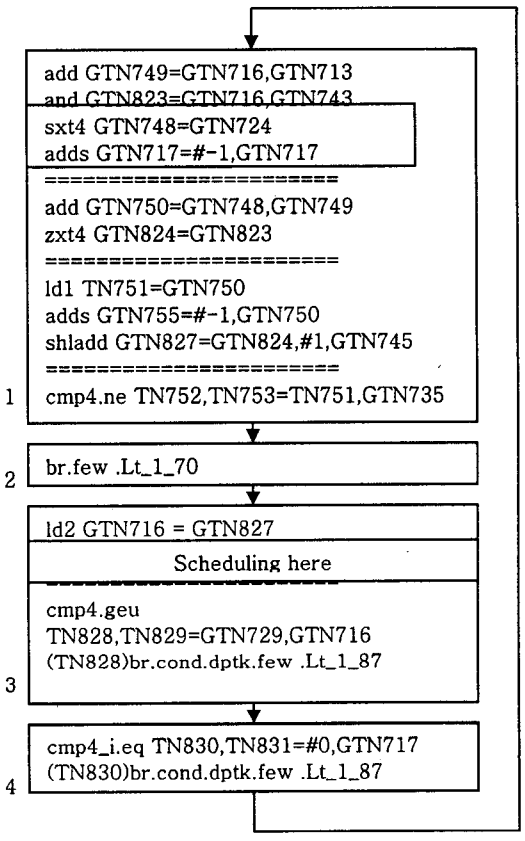

그림 1

수 있으며, Global Instruction Schedule과 달리 미리 계산된 의존성 그래프를 사용하지 않고, 매번 스케줄 후보를 계산하기 때문에 가변인 DAG에서의 스케줄에 유용하다.

4. IA-64 에서의 향상된 파이프라인 스케줄 구현

IA-64는 향상된 파이프라인 스케줄이 처음 발표되었을 때에 사용한 가상의 VLIW 구조와도 다른 점이 많을 뿐 아니라, 최초로 구현된 Sun Microsystems 의 Ultra SPARC과도 많은 차이를 갖고 있는 아키텍쳐이다.

4.1 speculative load

IA-64 에서는 기존의 아키텍쳐들과 달리 load 명령어에 있어서 control speculation과 data speculation이 모두 가능하게 되어있다. 즉 load 명령어를 분기 명령을 지나서 code motion

하는 control speculation과 memory disambiguation을 하지 않고도 load - store 명령 사이의 의존관계를 극복할 수 있도록 data speculation 을 지원하고 있다. 대신 이러한 speculation 이 정확하지 않을 경우가 있기 때문에 원래 load 명령어가 있던 곳에 검사 및 회복 명령어들을 추가해 주어야 한다. 이에 따라 speculation 되는 load 명령어에 대해서 스케줄링 과정에서 원래의 위치와 스케줄 된 위치를 기록하면서, 스케줄이 종료된 뒤에 회복코드를 생성하는 작업을 해줘야 한다.

그래서 향상된 파이프라인 스케줄 과정에서 load 명령을 code motion 할 때에는 조건분기 명령이나 store 명령을 넘어서 이동을 하는지를 검사하며, 그럴 경우 원래의 위치와 이동된 위치를 저장한다. 그리고 스케줄 된 load 명령어에 대해서 다시 이동을 할 때에는 스케줄 된 명령어가 이미 speculative load 명령어라고 체크되어있을 경우 이미 저장된 자료구조를 검색해서 이동된 위치만을 갱신시켜주다. 이런 과정을 거쳐서 스케줄이 종료된 다음에는 저장된 자료구조를 바탕으로 회복코드를 만들어주고, load 명령어를 speculative load 명령어로 변환해준다.

4.3 New heuristic

기존의 향상된 파이프라인 스케줄과 선택적 스케줄을 구현하는데 있어서 가장 중요하게 사용된 heuristic은 Degree Of Speculativeness(DOS) 로 분기명령을 몇 개를 지나서 code motion이 되는지가 중요한 heuristic으로 사용되었다. 이 방법은 실행 가능성이 높은 명령어를 스케줄 함으로써 자원을 효과적으로 쓸 수 있다는 장점이 있지만, 실제로 결정적인 경로에서 걸리는 시간을 줄인다는 스케줄 본래의 목적과는 차이가 있는 heuristic이다. 그래서 우리는 보통의 DAG 스케줄과 같이 의존관계 그래프에서 의존관계를 가진 명령어들의 개수를 DOS와 함께 사용한다. 대신 보통의 DAG 스케줄과는 달리 DAG가 계속 변하기 때문에 의존관계 그래프도 계속 변한다. 명령어를 한번 스케줄 할 때마다 그 명령어와 관련된 의존관계를 갱신하면서 그래프를 수정하는 방식으로 의존관계 그래프를 유지한다.

5. 실험결과

그림 2는 900Mhz Itanium2 위에서 실험한 결과로, base 는 ORC-2.1 컴파일러에서 O3레벨 최적화를 적용한 것이고 EPS는 ORC-2.1[6] 위에 EPS를 적용하여 O3 레벨 최적화를 한 것이다. 최대 8% 의 성능향상 효과가 있었으며, 평균 2%의 성능 향상이 있었다.

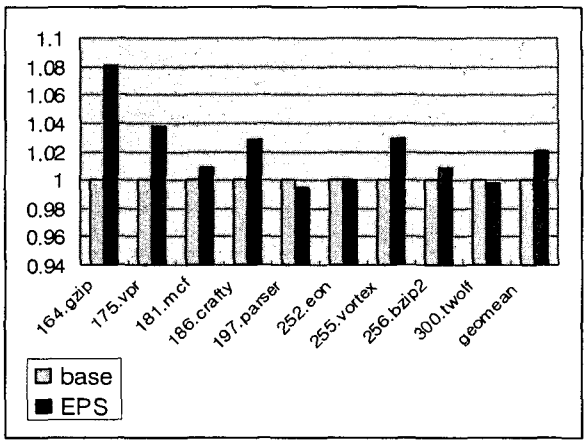

그림 2

6. 참고문헌

[1] Kemal Ebcioğlu. A compilation technique for software pipelining of loops with conditional jumps. In *Proceedings of workshop on Microprogramming*, pages 69-79, 1987

[2] Soo-Mook Moon, Kemal Ebcioğlu. Parallelizing nonnumerical code with selective scheduling and software pipelining. In *ACM Transactions on Programming Languages and Systems*, vol. 19, issue 6, pages 853-898, Nov. 1997

[3] David Bernstein, Michael Rodeh. Global instruction scheduling for superscalar machines. In *Proceedings of conference on Programming language design and implementation*, pages 241-255, 1991

[4] Jay Bharadwaj, Kishore Menezes, Chris McKinsey . Wavefront scheduling: path based data representation and scheduling of subgraphs. In *Proceedings* of international symposium on Microarchitecture, pages 262-271, 1999

[5] Scott A. Mahlke, et.al. Effective compiler support for predicated execution using the hyperblock. In *Proceedings of international symposium on Microarchitecture*, pages 45-54, 1992

[6] http://ipf-orc.sourceforge.net/