제어 독립적인 명령어의 선택적 복구 메커니즘

윤성룡, 신영호, 조영일

SK텔레텍,에이엠소프트,수원대학교 컴퓨터학과
slyoon@skteletech.co.kr, yhshin@airmedia.co.kr, yicho@mail.suwon.ac.kr

A Selective Recovery Mechanism of Control-Flow Independent Instructions

Sung-Lyong Yoon, Young-Ho Shin, Young-Il Cho
SK Teletech, AM Soft, Dept. of CS, Univ. of Suwon

요 약

최신의 프로세서는 분기명령에 의한 파이프라인 지연을 피하기 위해 분기 예상 기법을 사용하고 있다. 그러나 예측기에서 예상이 잘못된 경우에는 예상한 분기 방향의 명령어들을 무효화시키고 올바른 분기 방향의 명령어들을 다시 반입하여 수행시키므로 서 수행 사이클과 하드웨어 자원을 낭비하게된다.

본 논문에서는 컴파일 시 프로파일링을 통한 정적인 방법과 프로그램상의 제어 흐름을 통해 동적으로 제어 독립적인 명령어를 탐지해서 분기 명령어의 잘못된 예상으로 인해 무효화되는 명령어를 효과적으로 감소시켜 프로세서의 성능을 향상시키는 메커니즘을 제안한다.

SPECint95 벤치마크 프로그램에 대해 기존의 방법과 본 논문에서 제안한 방법 사이의 사이클 당 수행된 명령어 수를 분석한 결과, 4-이슈 프로세서에서 2%~7%, 8-이슈 프로세서에서 4%~15%, 16-이슈 프로세서에서 18%~28%의 성능 향상을 보이고 있다.

1. 서 론

최신의 프로세서는 분기명령에 의한 파이프라인 지연을 피하기 위해 분기 예상 기법을 사용하고 있고, 분기 정확도에서 상당한 개선을 얻고 있다[1,2]. 그러나 완벽한 예상을 얻기에는 몇 가지 문제점들이 있다. 첫 번째는 프로그램에 있는 어떤 분기 명령들은 예상하기가 어렵고, 두 번째는 유한 크기의 예상 테이블은 부정적 간섭(destructive interference)을 발생시킨다. 이러한 문제점들은 분기 예상 실패를 유발시키고, 그 결과 프로세서 성능에 상당한 손실을 초래한다.

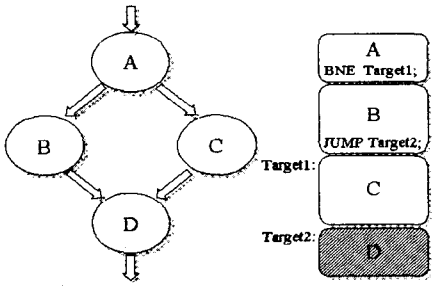
예상이 어려운 분기 명령어는 예상이 틀렸을 경우 분기 명령어 이후에 반입된 명령어들을 모두 무효화시키고 올바른 방향의 명령어들을 재 반입해서 수행시키는 복구 메커니즘을 사용한다. 이는 명령어 윈도우의 효율성을 제한하고, 수행 사이클과 하드웨어 자원의 낭비가 심해져 프로세서 성능 감소를 가져온다[3,4,5].

본 논문에서는 분기 예상이 실패했을 경우 잘못 예상된 분기 방향의 명령어의 수행을 무효화시키고 올바른 분기 방향의 명령어를 수행시키는 복구 과정에서 무효화되는 명령어를 효과적으로 감소시켜 프로세서 성능을 향상시키는 새로운 분기 예상 실패 복구 메커니즘을 제안한다. 제안한 복구 메커니즘은 분기 명령어의 결과에 상관없이 항상 수행되는 명령어 즉, 분기가 수렴한 이후의 명령어 중 재 사용될 수 있는 명령어는 분기 예상 실패 시 무효화하지 않고 이후에 재 사용함으로써 하드웨어 자원의 낭비를 줄이고 반입 및 수행에 필요한 사이클을 감소시켜 프로세서의 성능을 향상시킬 수 있다.

2. 배경 및 관련 연구

분기 명령어 이후에 나타나는 명령어는 분기 명령어의 결과에 따라 수행할 명령어들이 결정된다. [그림 1]의 블록 B와 C의 명령어처럼 분기 명령어의 결과에 따라 수행 여부가 결정되는 명령어를 제어 종속적인 명령어라고 한다. 반대로 블록 D와 같이 분기 결과에 상관없이 항상 수행되는 명령어를 제어 독립적인 명령어라 한다.

기존의 분기 예상 복구 메커니즘에서는 분기 명령어의 예상이 틀렸을 경우 분기 명령어 이후의 반입, 수행되었던 모든 명령어를 무효화시키고 올바른 방향의 명령어를 반입, 수행시킨다. 그러나, 제어 독립성을 이용하면 무효화되는 명령어를 최소화시켜 프로세서의 성능을 향상시킬 수 있다.



[그림 1] 제어 독립적인 명령어의 흐름도

[그림 1]의 블록 A에 있는 분기 명령어를 예상하여 블록 B와 D가 수행되었다고 가정하자. 분기 명령어의 실행이 완료되어 예상이 실패한 것으로 판명되면, 블록 B와 D의 명령어를 무효화시킨 다음, 올바른 분기 방향의 블록 C와 D를 반입하여 수행시켜야 한다. 이 때, 블록 D는 잘못 예상된 경로에서

이미 반입되어 수행되었다. 따라서, 블록 B만이 분기 명령어에 제어 종속적인 명령어이므로 블록 B의 명령어들을 무효화시키고 블록 C의 명령어들만 반입시키면 된다. 그러나, 블록 D의 명령어 중 제어 종속적인 명령과 자료 종속(Data Dependence) 관계를 가지고 있는 명령어들도 재 수행이 되어야 한다. 이와 같이 모든 제어 독립적인 명령어에 대해서는 재 반입을 위한 페널티를 감소시킬 수 있고, 제어 독립적인 명령어들 중에서 제어 종속적인 명령어와 자료 종속 관계가 없는 명령어들은 재 수행에 드는 페널티를 감소시킬 수 있다.

Smith등은 슈퍼스칼라 프로세서에서 제어 독립성에 대한 기본적인 연구를 통해 잠재적인 성능 향상을 제시하였으며, trace 프로세서에서의 제어 독립성을 이용한 기법도 제안하였다[3].

Shen등은 분기 예상 실패 시 복구하기 위한 메커니즘으로 동적 제어 독립성을 이용하였다[4]. 그러나 파이프라인 이슈 폭이 커지고 있는 최신의 슈퍼스칼라에서는 기하급수적인 하드웨어 자원의 증가를 요구하기 때문에 실제적인 구현이 불가능하다[3].

3. 분기 예상 실패 복구 메커니즘

잘못 예상된 분기 명령어에 의해 변경된 프로세서의 상태를 복구할 때 제어 독립성을 이용하기 위해서 무효화될 필요가 없이 다시 사용되는 제어 독립적인 명령어를 탐지해야 하고, 제어 독립적인 명령어 중에서 제어 종속적인 명령어들과 자료 종속 관계를 가지고 있는 명령어를 찾아내야 한다.

3.1 제어 독립적인 명령어의 탐지

본 논문에서 제안한 기법은 프로그램의 구조에서 나타나는 유형을 이용해 제어 독립적인 명령어를 탐지해 낸다. 컴파일 시 각각의 분기 명령어에 대응하는 제어 독립적인 명령어의 주소 정보를 첨가시킨다. 이렇게 정적으로 구해진 정보를 이용해 프로그램의 수행 시 동적으로 분기 명령어에 대응하는 제어 독립적인 명령어를 탐지한다.

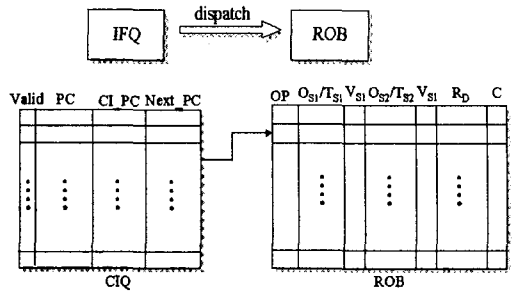
[그림1]은 고급언어에서 if-then-else 구조에 대한 명령어의 유형을 나타낸다. 이 유형의 하위 레벨의 프로그램을 보면 기본 블록 A의 마지막에 조건 분기 명령어가 존재하며, 조건 분기 명령어의 목적 명령어(Target1) 바로 앞의 명령어가 제어 독립적인 명령어로 분기하는 무조건 분기 명령어가 존재한다. 따라서, 양방향으로 분기하는 조건 분기 명령어의 제어 독립적인 명령어는 블록 B의 무조건 분기 명령어의 목적 명령어(Target2)가 된다.

else가 없는 if-then 구조에서의 제어 독립적인 명령어는 블록 A의 조건 분기 명령의 목적 명령어(Target1)가 된다.

반복문과 같은 구조에서는 분기 명령어의 결과에 상관없이 항상 수행되는 명령어, 즉 제어 독립적인 명령어는 분기 명령어의 다음 명령어가 된다.

위와 같은 방법을 통해 각 분기 명령어에 대한 제어 독립적인 명령어의 정보를 구해서 프로그램이 실제로 수행하는 동안에 동적으로 제어 독립적인 명령어를 탐지할 수 있다.

3.2 제어 독립적인 명령어 중 재 수행할 명령어의 탐지 메커니즘



[그림 2] 제안한 복구 메커니즘의 하드웨어 구조

[그림2]는 제어 독립적인 명령어중 제어 종속적인 명령어와 자료종속관계를 갖는 명령어만 선택적으로 재 수행시키는 구조를 보여준다.

제어 독립적인 명령어의 전체 주소는 CIQ(Control Independent Queue)에 입력된다. CIQ의 엔트리에서 PC는 분기 명령어의 주소를 저장하며 CI_PC 필드에는 해당 분기 명령어의 제어 독립적인 명령어의 주소를 입력한다. 분기 명령어의 수행이 완료되어 분기 명령어의 예상이 틀렸다고 판단되었을 경우 Valid 비트를 셋트시켜 예상이 틀린 분기 명령어임을 명시하고, 이후 예상이 틀린 분기 명령어의 복구 시 제어 종속적인 명령어만을 무효화시킨다.

분기 명령어의 예상이 틀렸을 경우에는 분기 명령어 이후의 명령어를 ROB에서 무효화시키고 올바른 분기 방향의 명령어를 반입해서 수행해야 한다. 이 때, 분기 명령어 이후의 모든 명령어를 무효화시키지 않고 제어 명령어 큐에 저장되어 있는 정보를 이용해 해당 분기 명령어에서부터 제어 독립적인 명령어 바로 이전의 제어 종속적인 명령어만을 무효화시킨다.

[그림2]의 ROB는 기존의 ROB와 동일한 구조에 C(Completion) 비트만을 추가한 형태이다. C 비트는 명령어의 수행이 완료될 때 세트시켜 해당 명령어의 수행이 완료되었는지의 상태를 나타낸다. 제거되는 제어 종속적인 명령어가 생성한 결과 값을 이후의 제어 독립적인 명령어가 입력 값으로 사용하고 있다면, 즉 자료 종속 관계가 존재한다면 해당 제어 독립적인 명령어는 잘못된 입력 값을 가지고 수행된 명령어가 된다. 이와 같은 제어 독립적인 명령어의 C 비트는 리셋시킨다. 해당 제어 독립적인 명령어가 이미 수행이 완료되었다면 세트되어 있는 C비트를 리셋시킴으로써 이후에 제어 독립적인 명령어의 수행이 시작할 때 최근의 올바른 입력 값을 가지고 다시 수행하게 하는 역할을 한다.

이와 같이 분기 명령어의 예상이 실패했을 경우, 이후에 다시 사용될 수 있는 명령어를 ROB에서 제거하지 않고 유지함으로써 제거되는 명령어를 감소시켜 제어 독립적인 명령어의

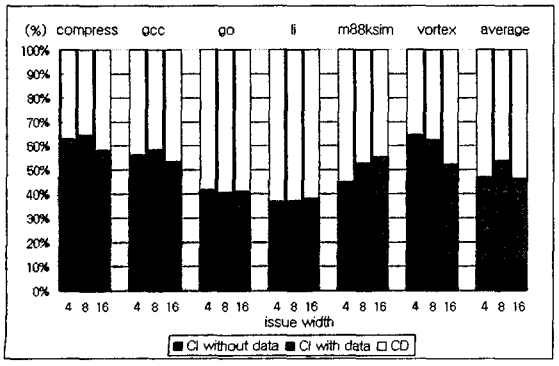
재 수행 시간을 줄일 수 있고 하드웨어 자원의 낭비를 줄임으로써 프로세서의 성능을 향상시킬 수 있다.

4. 실험 결과

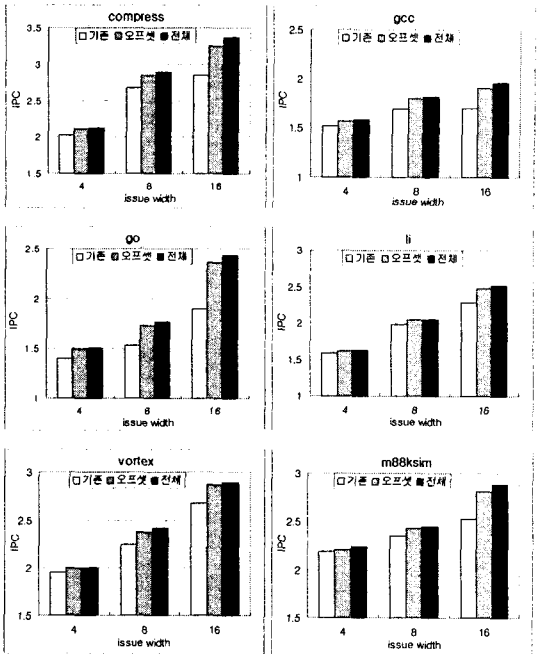
본 논문에서 사용한 시뮬레이터는 SimpleScalar/PISA 3.0 툴 셋을 사용하여 예상 실패의 패널티를 모델링한 실행 구동 시뮬레이터이다. 실험은 4, 8, 16 개의 이슈 폭을 갖는 비 순서적 이슈 및 실행 프로세서를 사용하였으며 ROB 대신 레지스터 재명명과 명령어 재순서화를 동시에 제공하는 RUU (Register Update Unit)를 사용하였다. 실험을 위해서 6개의 SPEC95int 벤치마크 프로그램을 사용하였고, 각 벤치마크 프로그램은 -O3의 최적화 옵션을 가진 GNU GCC(ver 2.6.3)를 사용해 컴파일했으며 각 벤치마크 프로그램의 수행된 명령어 수를 최대 100M까지 제한적으로 수행하였다.

[그림3]의 그래프는 분기 예상 실패로 인해 무효화되는 명령어를 보여준다. 'CD'는 무효화되는 명령어 중에서 제어 종속적인 명령어의 비율을 나타내며, CI는 제어 독립적인 명령어의 비율을 나타낸다. 제어 독립적인 명령어 중에서 'CI with data'는 제어 종속적인 명령어와 자료 종속관계를 갖는 비율을 나타내고, 'CI without data'는 제어 종속적인 명령어와 자료 종속관계가 없는 명령어의 비율을 나타낸다. 6개의 벤치마크 프로그램의 실험 결과, 8-이슈 폭을 갖는 프로세서에서 평균 53%의 명령어의 재 반입 사이클을 줄일 수 있고, 평균 28%에 해당하는 명령어는 재 수행하지 않고 이전 결과 값을 그대로 사용함으로써 재 수행 사이클을 감소시킬 수 있다.

[그림4]는 6개의 SPECint95 벤치마크 프로그램을 사용해 4, 8, 16 개의 이슈 폭을 갖는 프로세서에서 사이클 당 수행된 명령어 수를 측정함으로써 제어 독립을 이용한 복구 메커니즘의 성능 향상을 보여주고 있다. 모든 벤치마크 프로그램에서 기존의 방법에 비해 6-비트 오프셋을 사용한 경우와 전체 주소를 사용한 경우에서 성능이 향상되고 있다. 4-이슈 폭의 프로세서는 2%~7%, 8-이슈 폭의 프로세서는 4%~15%, 그리고 16-이슈 폭의 프로세서에는 8%~28%의 상대적인 성능 향상을 보이고 있다.



[그림 3] 제어 독립적인 명령어의 비율



[그림 4] 복구 메커니즘의 성능 향상

5. 결 론

본 논문에서는 분기 예상 실패시 분기 명령어 이후의 명령어들을 복구하는데 필요한 페널티를 감소시키기 위해 제어 독립적인 명령어를 재 사용하는 새로운 복구 메커니즘을 제안하였다. 제어 독립성을 사용하기 위해서 컴파일 시 프로파일링을 통한 정적인 방법과 프로그램상의 제어 흐름을 통해 동적으로 제어 독립적인 명령어를 탐지하는 방법을 이용하여 분기 명령어의 잘못된 예상으로 인해 무효화되는 명령어를 효과적으로 감소시켜 프로세서의 성능을 향상시켰다.

참 고 문 헌

[1] C. C. Lee, I. C. Chen and T. Mudge, "The Bi-Mode Branch Predictor", in Proc. of 30th MICRO, Dec. 1997.

[2] E. S., M. Alsup and Y. Patt, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference", in Proc. of 24th ISCA, May 1997.

[3] E. Rotenberg and J. E. Smith, "Control Independence in Trace Processors", Journal of Instruction-Level Parallelism, May 2000.

[4] Y. Chou, J. Fung and J. Shen, "Reducing branch misprediction penalties via dynamic control independence detection", Intl. Conf. on Supercomputing, June 1999.

[5] V. Y. Cher and T. N. Vijaykumar, " Skipper: A Microarchitecture for Exploiting Control-flow Independence", 34th MICRO, pp. 4-15, Dec. 2001