

제 19강. 4.11 명령어를 통한 병렬성, 4.12 실례: Intel Core i7 6700과 ARM Cortex-A53, 4.15 오류 및 함정, 4.16 결론

4.11 명령어를 통한 병렬성

- 파이프라이닝 : 명령어들 사이의 병렬성 이용
- 명령어 수준 병렬성 (instruction-level parallelism, ILP)

명령어 수준 병렬성 양을 증가시키는 2가지 방법

1. 파이프라인의 깊이를 증가시켜 더 많은 명령어들을 중첩시키기
세탁기의 세탁, 헹굼, 탈수 세 단계를 각각 다른 기계로 나누기
단계들이 같은 길이를 갖도록
병렬성의 양은 늘어나고 → 클럭 사이클이 더 짧아져 → 성능이 더 좋아질 가능성이 있음
2. 다중 내보내기(multiple issue) 컴퓨터 내부의 구성 요소들을 여러 벌 갖도록 하여 매 파이프라인 단계에서 다수의 명령어를 내보낼 수 있도록 함
한 클럭 사이클에 여러 명령어가 시작되는 기법
세탁소에서 세탁기 3대, 건조기 3대를 두는 것을 의미
단점 : 일거리를 다음 파이프라인으로 넘기기 위해 해야 되는 일이 늘어남

ILP(명령어 수준 병렬성)이 증가하면

명령어 실행 속도가 클럭 속도보다 빨라질 수 있음

CPI (명령어당 클럭사이클 수) < 1



IPC (클럭사이클 당 명령어 수, Instruction Per Clock cycle)

다중 내보내기 구현 2가지 방법

- 정적 다중 내보내기 static multiple issue : 많은 결정들이 실행 **전에 컴파일러**에 의해 이루어지는 다중 내보내기 프로세서 구현 방법
- 동적 다중 내보내기 dynamic multiple issue : 많은 결정들이 실행 **중에 프로세서**에 의해 이루어지는 다중 내보내기 프로세서 구현 방법

다중 내보내기 파이프라인이 다루어야 할 2가지 문제

1. 명령어로 issue slot 채우기

주어진 클럭 사이클에 얼마나 많은 명령어를 내보내고 또 어떤 명령어들을 내보낼지
컴파일러가 다중 내보내기에 유리하게 명령어의 순서를 미리 바꾸어서 내보내기 속도를 개선할 수 있게 도와주는 경우가 많음


2. 데이터 해저드와 제어 해저드의 처리

- 정적 내보내기 프로세서 : 컴파일러가 정적으로 처리
- 동적 내보내기 프로세서 : 실행시간에 작동하는 하드웨어 기술을 이용하여 해저드를 없애려고 노력

→ 정적, 동적 다중 내보내기 둘 다 한 방법이 다른 방법에 차용되는 경우가 많기 때문에 어느 방법도 완전히 순수하다고는 할 수 없음

추정 speculation

1.2절의 7대 아이디어 중 하나인 **예측**을 기반으로 하는 기술

 컴파일러나 프로세서가 명령어의 특성에 대해 추측할 수 있게 하여 이 명령어에 종속적일 수 있는 다른 명령어들의 실행을 시작할 수 있게 함

ex. 분기 명령어의 결과를 추정한다면 분기 명령어 뒤의 명령어들이 일찍 실행될 수 있음

ex. 적재 명령어 바로 앞의 저장 명령어가 같은 주소를 참조하지 않는다고 추정하여 적재 명령어가 저장 명령어보다 먼저 실행될 수 있게 하는 것

- 추정은 잘못될 수 있기 때문에
 - 올바른지 확인할 방법
 - 되돌리는 방법 취소하는 방법

을 포함해야 한다.

추정이 잘못 되었을 경우

- 컴파일러
 - 추정의 정확성을 체크할 명령어들을 추가

- 추정이 잘못되었을 때 사용할 오류 수정 루틴 제공
- 프로세서
 - 추정 결과가 더 이상 추정이 아니라는 것을 알 때까지 추정 결과를 버퍼링
 - 추정이 옳다면 : 버퍼의 내용을 레지스터나 메모리에 씀
 - 추정이 틀렸다면 : 버퍼 내용을 쓸어버리고 올바른 명령어 순서를 다시 실행

정적 다중 내보내기

한 클럭 사이클에 같이 나갈 수 있는 명령어 조합에 제한이 있기 때문에 내보내기 패킷을 미리 정의된 필드에 여러 연산자가 있는 단일 명령어로 보는 것이 유용

VLIW(Very Long Instruction Word)

여러 opcode 필드가 있는 긴 명령어 하나에 독립적인 연산 여러 개를 정의하고 이들을 한꺼번에 내보내는 명령어 구조 집합 종류

예: MIPS 명령어 집합 구조의 정적 다중 내보내기

ALU 명령어와 데이터 전송 명령어를 병렬로 내보내기 위해서는

일반적인 해저드 검출 회로와 지연 회로 이외에

레지스터 파일의 추가 포트 필요

→ 한 클럭 사이클에 ALU 연산을 위해 2 레지스터를 읽어야 되고

저장 명령어를 위해 2 레지스터를 읽어야 되며

ALU 연산, 적재 명령어를 위해 쓰기 포트 2개가 더 필요하다

⇒ 2배 명령어가 실행 중 중첩되어야 하므로 **사용 지연**이 일어남

⇒ 해결을 위해 **컴파일러 스케줄링** 기법이 요구됨

동적 다중 내보내기

수퍼스칼라

프로세서가 실행 중에 명령어를 선택하여 한 클럭 사이클에 2개 이상의 명령어를 실행할 수 있도록 해주는 고급 파이프라이닝 기법

VLIW vs 수퍼스칼라

- VLIW 는 컴파일러가 잘 스케줄해야 프로그램이 제대로 실행

- 수퍼스칼라 는 스케줄링 여부와 상관없이 하드웨어가 코드의 올바른 실행을 보장, 컴파일된 코드는 프로세서의 내보내기율이나 파이프라인의 구조와 상관 없이 항상 올바르게 동작

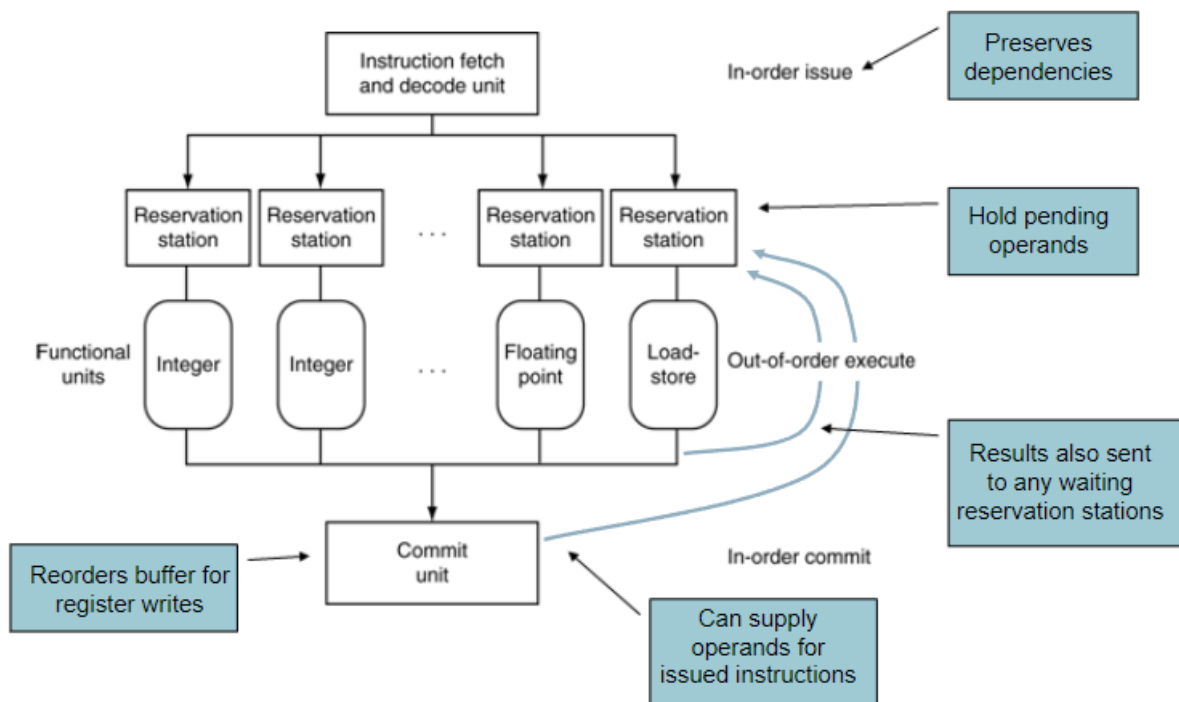
동적 파이프라인 스케줄링

지연을 피할 수 있도록 명령어 실행 순서를 바꾸는 재정렬을 하드웨어가 지원하는 것

동적 파이프라인 스케줄링

프로세서 파이프라인은

- 명령어 인출 및 내보내기 유닛
- 다수의 기능 유닛
- 결과 쓰기 유닛



동적 스케줄링 파이프라인의 3가지 주요 유닛

- 각 기능 유닛은 reservation station(대기 영역)이라 불리는 버퍼를 가지고 있음
이 버퍼는 연산자와 피연산자를 가지고 있음
- 버퍼에 모든 피연산자가 준비되고 실행할 기능 유닛이 준비돼 있으면 연산을 실행
- 연산이 끝나면 그 결과는 결과 쓰기 유닛뿐만 아니라 이 결과를 기다리고 있는 대기 영역에도 보내짐

- 결과 쓰기 유닛은 결과값을 버퍼링하고 있다가 안전할 때 결과값을 레지스터 파일이나 메모리에 씀
 - 결과 쓰기 유닛에 있는 버퍼 : 재정렬 버퍼 (reorder buffer)

비순차 실행 (out-of-order execution)

실행할 수 없는 명령어 때문에 뒤의 명령어들을 기다리지 않게 하는 파이프라인 실행 상황.

순차 결과 쓰기 (in-order commit)

파이프라인 실행의 결과를 명령어 인출과 같은 순서로 프로그래머가 볼 수 있는 상태 소자에 쓰는 결과 쓰기 방식

>> 오늘날 모든 동적 스케줄링 파이프라인은 순차 결과 쓰기를 사용

commit-issue : in-order

execution : out-of-order

왜 동적 스케줄링을 사용할까?

p.389

1. 모든 지연이 예측 가능하지 않다.

메모리 계층구조에서 캐시 실패는 예측 불가능한 지연을 일으킴 → 프로세서로 하여금 지연이 끝나기를 기다리는 동안에도 명령어를 계속 실행할 수 있도록 하여 이런 지연의 일부를 감출 수 있음.

2. 프로세서가 동적 분기 예측을 이용해 분기 결과에 대해 추정한다면 컴파일 시에는 명령어의 정확한 순서를 알 수 없음

이 순서는 분기 예측과 실제 행동 모두에 의존하기 때문

더 많은 ILP를 얻기 위해 동적 스케줄링을 하지 않으면 추정의 이득을 크게 제한하게 됨

3. 파이프라인 지연과 내보내기 폭이 구현마다 달라지면 코드를 가장 잘 컴파일하는 방법도 달라지게 됨

요점 정리

파이프라이닝과 다중 내보내기는 둘 다 최고 명령어 처리량을 증가시키고

명령어 수준 병렬성(ILP)를 활용하는 기술이다.

그러나 종속성이 해결될 때까지 프로세서가 기다려야 하는 경우가 많기 때문에 프로그램의 데이터 종속성, 제어 종속성은 지속적으로 얻을 수 있는 성능의 상한선을 결정.

ILP를 활용하기 위한

소프트웨어 중심 접근 방법은 컴파일러의 능력에 의존해 종속성을 찾아내고

하드웨어 중심 접근 방법은 파이프라인과 내보내기 메커니즘 확장에 의존

4.15 오류 및 함정

오류 : 파이프라인은 쉽다

The devil is in the details

오류 : 파이프라이닝 발상들은 기술과는 상관 없이 구현될 수 있다.

함정 : 명령어 집합 설계를 잘못하면 파이프라이닝에 나쁜 영향을 끼칠 수 있다.

파이프라이닝의 문제 중 많은 것들이 명령어 집합의 복잡성 때문에 일어남

- 너무 다양하게 변하는 명령어 길이와 실행시간은 파이프라인 단계의 균형을 깨뜨려서 명령어 집합 수준의 파이프라인 설계에서 해저드 검출을 매우 어렵게 만듦

마이크로연산과 마이크로 파이프라인 방법을 사용하여 해결 → Intel Core i7에서도 사용

마이크로연산과 실제 명령어 사이의 변환과 대응 관계를 유지하는 오버헤드가 생김

- 복잡한 주소지정 방식은 다른 종류의 문제를 일으킬 수 있음

레지스터 값을 바꾸는 주소지정 방식들은 해저드 검출을 어렵게 함

4.16 결론

- 파이프라이닝은 처리량을 증가시키지만 실행시간 또는 명령어 지연시간에 영향을 끼치지 않음

- 다중 명령어 내보내기는 데이터패스 하드웨어를 추가하여 한 클럭 사이클에 여러 개의 명령어가 시작되도록 하는 기술이지만 그 대가로 실제 지연시간은 증가함
 - 파이프라이닝은 단순한 단일 사이클 데이터패스의 클럭 사이클 시간을 줄이는 것
 - 다중 명령어 내보내기는 CPI 값을 줄이는 기술
 - 둘 다 명령어 수준 병렬성을 이용하려고 하는 기술
 - 하드웨어, 소프트웨어의 스케줄링, 예측을 통한 추정으로 데이터 해저드, 제어 해저드 줄일 수 있음
 - Amdahl's law : 병렬 컴퓨팅에서 멀티 프로세서를 사용할 때 프로그램의 성능향상은 프로그램의 순차적인 부분에 의해 제한된다. 예를 들면, 프로그램의 95%가 병렬화 할 수 있다면 이론적인 최대 성능 향상은 아무리 많은 프로세서를 사용하더라도 최대 20배로 제한된다.
- ⇒ 병렬 프로세서를 통한 처리 성능의 발전 → 메모리 계층구조의 병목