RAPTOR 의 명령어 페치 유닛 설계

이성권, 오형철, 이상원[1], 한우종[2]
고려대학교 정보공학과 병렬연산연구실
[1]고려대학교 전자공학과 ASIC 연구실
[2]한국전자통신연구원 컴퓨터시스템연구부
email : {myjove, hyeong}@hard.korea.ac.kr, [1]swlee@asic.korea.ac.kr, [2]wjhan@computer.etri.re.kr

Design of an Instruction Fetch Unit for RAPTOR, a On-Chip Multiprocessor

Sung-Kwon Lee, Hyeong-Cheol Oh, Sang-Won Lee[1], Woo-Jong Hahn[2]
Parallel Computation Lab., Department of Information Engineering, Korea University
[1]ASIC Lab., Department of Electronic Engineering, Korea University
[2]Computer System Research Department, ETRI
email : {myjove, hyeong}@hard.korea.ac.kr, [1]swlee@asic.korea.ac.kr, [2]wjhan@computer.etri.re.kr

Abstract

This paper introduces an instruction fetch unit which is designed for RAPTOR, an on-chip multiprocessor. In order to reduce control hazards, the proposed fetch unit supports a hybrid branch prediction scheme which consists of a static scheme and the 2bC branch prediction scheme. The fetch unit also utilizes the branch folding technique with two instruction buffers to avoid the branch penalty caused by mispredictions. Instructions are predecoded in the fetch unit to achieve extra performance gain.

1. 서 론

파이프라인 기법은 프로세서의 성능을 높이는데 사용되는 중요한 기술로서 오늘날 대부분의 프로세서들에서 사용되고 있으나, 분기 명령어의 불확실성의 결과로서 생성되는 분기 손실(branch penalty)이 파이프라인의 성능 향상의 주요 장애가 되고 있다[1, 2, 3]. 이러한 문제를 해결하기 위하여 추측 수행 기법이 사용되고 있는데, 이들 기법들은 크게 분기의 다중 경로에 자원을 할당하여 명령어를 처리하는 기법들과 단일 경로에 자원을 할당하여 명령어들을 처리하는 기법들로 분류할 수 있다. 전자의 경우 eager execution 과 disjoint eager execution 을 들 수 있으며, 후자의 경우 정적 분기 예측, 동적 분기 예측, 혼합 분기 예측, 다중 분기 예측 기법 등을 들 수 다[1]. Eager execution 은 모든 분기 경로에 자원을 할당하여 분기의 순서에 따라 경로를 선택하여 수행하며, 예측이 맞는 경로를 선택함으로써 패널티가 없게 되나 자원이 무한정하게 소모되므로 실제적으로 구현하기 힘들며, Disjoint eager execution 은 선택 확률이 높은 곳부터 자원을 할당함으로써 한정된 자원으로 효과적으로 분기 손실을 줄일 수 있다.[1, 4]. 단일 경로에 자원 할당하는 기법들은 프로세서의 성능을 극대화하기 위해 정확한 분기 예측을 추구하는 기법들인데, 최근에는 분기 예측 실패 시에 이를 보상하기 위하여 위의 두 기법을 혼합하는 방법이 추구되고 있다[8]. 본 논문에서는 분기 예측과 더불어 분기 손실을 줄이기 위해 분기 명령어의 두 경로에 버퍼를 할당하는 기법을 제안하고, RAPTOR 로 명명된 단일 칩 다중프로세서[9]의 명령어 페치 유닛으로 설계하였다. 또한 명령어 페치 유닛은 내부의 명령어 버퍼에서 프리디코드를 통해 유효 명령어를 판별함으로써 성능 향상을 기하였다.

2. RAPTOR 의 명령어 페치 유닛

현재 RAPTOR 는 컴파일러에 의해서 지원되는 프로파일 정적 기법[1, 5]과 2bC(2-bit saturating up-down counter)[2, 5, 6, 7]및 캐쉬를 이용한 동적 분기 예측 기법을 혼합 사용하며[3, 8], 분기 경로 자원 할당 기법으로 disjoint eager execution 과는 비슷하나 두 개로 한정된 분

기 경로에만 선택적으로 버퍼를 할당하여 이를 효율적으로 사용하는 이중 버퍼 기법을 사용한다.

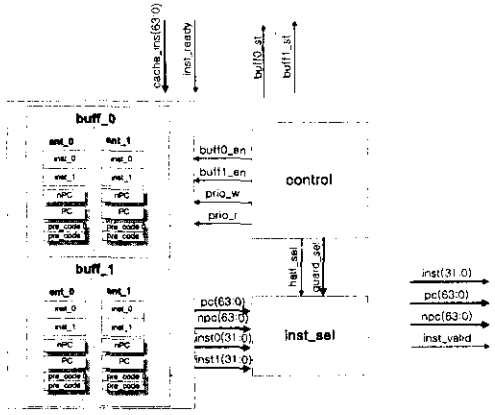

그림 1. RAPTOR 의 명령어 버퍼의 구조

본 논문에서 제안하는 이중 버퍼 기법에서는 분기 명령어에 대해 예측된 경로와 예측되지 않은 경로에 버퍼를 할당하고 이 분기 명령어의 처리 결과에 따라 예측이 성공하면 예측된 경로에 계속해서 버퍼 할당하고, 예측이 틀리면 이미 할당된 예측되지 않은 경로를 따라 수행을 하고 있는 동안 예측된 경로에 있는 버퍼를 다른 경로로 재할당하게 된다. 그림 1 은 제어 블록에서 쓰기 우선권(prio_w)과 읽기 우선권(prio_r)을 두어 두 개의 명령어 버퍼를 선택적으로 할당, 수행하는 것을 보이고 있다.

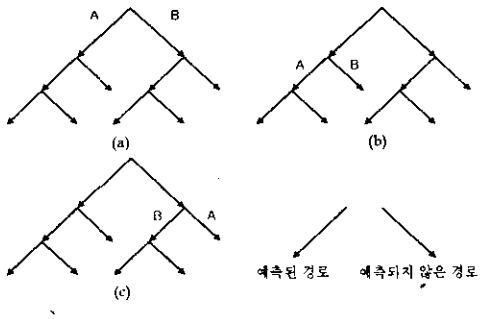

그림 2. RAPTOR 의 분기 경로에 따른 자원 할당

그림 2 는 분기 처리 이전과 이후의 분기 경로에 대해 버퍼 할당이 바뀌는 것을 보여주고 있다. 좌측 하단으로 향하는 방향이 예측된 경로이고, 우측 하단으로 향하는 방향이 예측되지 않은 경로이며, 주어진 자원은 두 개의 버퍼 A 와 B 로 한정되어 있기 때문에 예측 시는 그림 (a) 와 같이 첫 번째 분기 명령어의 경로에 버퍼 A 와 B 가 할당된다. 그림 (b)는 예측이 성공했을 경우, 그리고 그림 (c)는 예측이 실패했을 경우의 버퍼 재할당 순서를 나타내고 있다.

명령어 페치 유닛 내부의 두 개의 명령어 버퍼는 각각 두 개의 엔트리로 되어 있으며, 엔트리의 크기는 64 비트로 두 개의 명령어를 저장하므로 내부 명령어 버스의 크기는 64 비트이다. RAPTOR 의 정수 파이프라인 구조는 그림 3 과 같고, 분기 명령어 판별과 프리페치 요청은 프리디코드(PD) 단계에서 이루어지며 버퍼의 상태와 이전 요청되어 처리되는 명령어의 상태를 보고 요청한다. 그림 3 의 경우 분기 명령어가 없고, 캐쉬 미스가 없는 경우를 설명하였다.

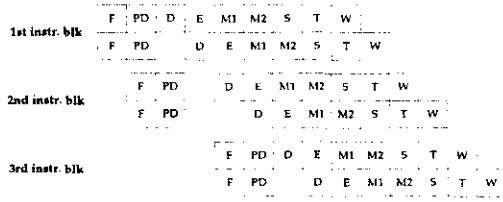

그림 3. Raptor 의 파이프라인 구조

그림 4 는 분기 명령어의 처리에 따른 명령어의 수행 과정을 설명하고 있다. 그림 (a)와 (b)는 선택될 것으로 예측된 조건 분기 명령어가 예측이 성공했을 경우와 실패했을 경우를, 그림 (c)와 (d)는 선택되지 않을 것으로 예측된 조건 분기 명령어가 예측이 성공했을 경우와 실패했을 경우를 보이고 있다. 버퍼의 해당 엔트리의 모든 명령어가 유효하고 캐쉬 미스가 없고 디코더와 실행 유닛에서 매 사이클 명령어를 수행한다고 가정하였을 경우를 보인 것으로 분기 손실이 없음을 알 수 있다. 만약 예측되지 않은 두 번째 해당 버퍼의 엔트리의 두 개의 명령어 중 한 개의 명령어만 유효 명령어인 경우, 한 사이클의 분기 손실이 발생하게 된다. 이는 예측 실패 이후 새로운 명령어를 요청하여 디코드 되기까지 두

사이클이 소요되고, 이러한 분기 손실은 미리 페치해 둔 두 번째 버퍼의 명령어를 수행함으로써 막을 수 있기 때문이다.

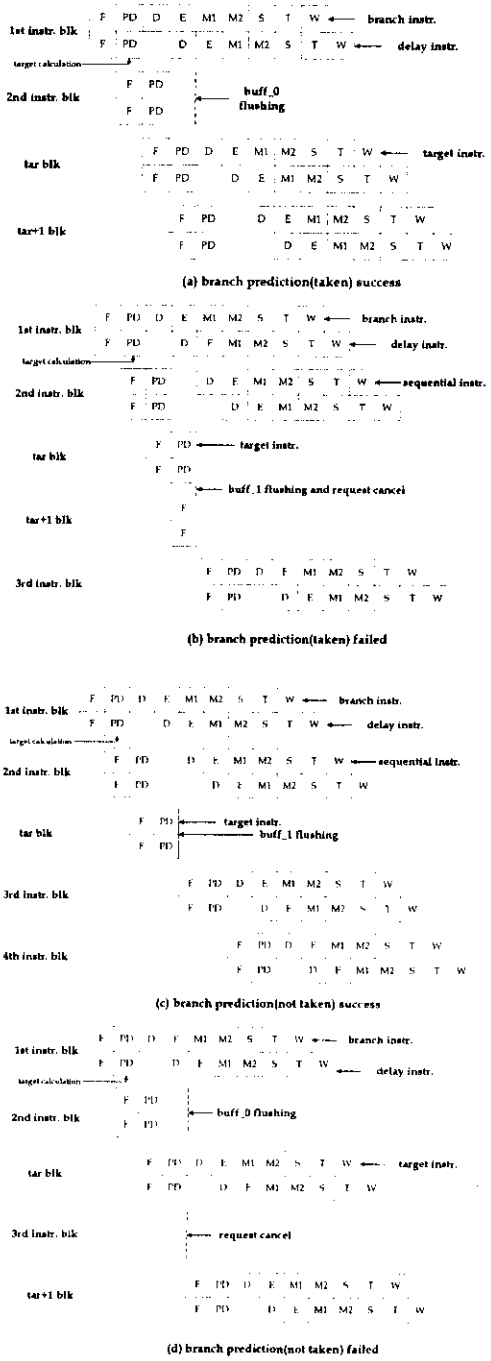

그림 4. 파이프라인에서의 분기 명령어 처리 과정

RAPTOR는 정적 스케쥴링 방식인 지연분기 방식[1, 6]을 지원하므로 유효 명령어 판별과 명령어 프라페치에 있어서 이를 고려하였다. 그림 5는 프리디코드를 통한 유효 명령어 판별의 예로써 조건 분기 명령어가 선택될 것으로 예측된 경우를 보이고 있다.

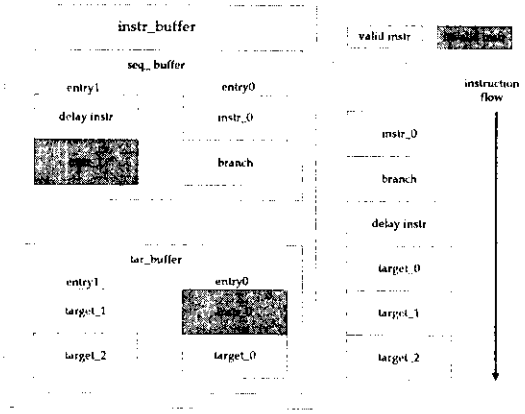

그림 5. 프리디코드를 통한 유효 명령어 판별의 예

3. 설계 및 검증

RAPTOR의 분기 예측은 초기에는 컴파일러에 의해서 지원되는 프로파일 정적 기법을 사용하기 때문에 명령어 코드 내의 예측 비트(predict bit)를 보고 예측하며, 분기 예측 이후 처리된 분기 명령어는 명령어 캐쉬의 분기 인덱스 영역에 처리 결과 상태가 저장되고 이후 이 정보는 동적 예측에 사용된다. 분기 명령어마다 해당된 상태 비트는 분기 명령어 처리 후 재조정되며, 분기 명령어의 상태 관리는 ST(Strongly Taken), LT(Likely Taken), LNT(Likely Not Taken), SNT(Strongly Not Taken)의 네 가지 상태로 분리되어 분기 상태 처리 블록에서 2bC 예측 기법을 사용하여 상태 변화를 관리하게 된다.

본 논문에서 사용된 혼합 분기 예측 기법의 정확도가 정적 예측 기법 75%, 동적 예측 기법이 90% 정도로 연구되어 있으며[1, 3], 단일 버퍼 기법의 경우 분기 예측 실패 시 세 사이클의 분기 손실이 발생한다. 제 2 절에서 설명된 바와 같이, 이중 버퍼 기법의 경우 엔트리의 명령어들이 모두 유효하다고 하였을 때 분기 손실이 발생하지 않으므로 매 사이클 유효 명령어가 수행되며, 캐쉬 미스가 없고 평균적으로 전체 명령어 중 30% 정

도가 분기 명령어라고 가정하였을 때, 이중 버퍼 기법을 사용하였을 경우 단일 버퍼 기법에 비해 얻어지는 성능 향상은 9~22.5% 정도이다.

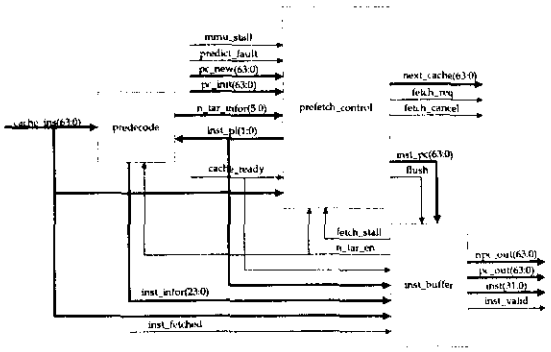

그림 6. RAPTOR 의 명령어 페치 유닛 블록도

제안된 명령어 페치 유닛[그림 6]은 verilog HDL 로 설계되었으며, 현재 Whestone 벤치마크 프로그램을 사용하여 이중 명령어 버퍼 구조를 갖는 경우와 단일 명령어 버퍼를 갖는 경우를 비교하는 실험을 수행중이다.

4. 결 론

RAPTOR 의 명령어 페치 은 정적 분기 예측과 동적 분기 예측을 혼합한 분기 예측을 사용하면서, 두 개의 명령어 버퍼를 사용하여 분기 명령어에 대해 선택된 경로와 그렇지 않은 경로에 해당되는 명령어를 모두 프리 패치하여 예측이 틀렸을 경우 미리 페치해 놓은 다른 경로의 명령어를 수행함으로써 새로운 타겟을 요청하여 디코드 되기까지의 손실을 줄일 수 있다.

본 논문에서 제안한 기법을 토대로 분기 명령어에 대한 자원을 추가함으로써 분기 손실을 완전히 줄일 수 있는데, 버퍼의 수를 늘려 처리할 수 있는 분기 경로의 수를 늘리는 방법과 엔트리의 크기를 확장 시켜 유효 명령어 수를 늘리는 방법이 있다. 전자의 경우 이를 제어하는 하드웨어가 추가적으로 커진다는 단점이 있으며, 후자의 경우 엔트리의 크기가 커짐에 따라 반드시 유효 명령어가 비례하여 많아진다는 보장이 없다는 단점이 있다.

참고문헌

[1] A. K. Uht. V. Singdagi. and S. Somanatha, "Branch Effect Reduction Techniques", *IEEE computers*, May 1997.

[2] T. Y. Yeh and Y. N. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History", *Preceedings of the 20th Annual International Symposium on Computer Architecture*, pp.257-266, May 1993.

[3] M. Evers, P. Y. Chang. and Y. N. Patt, "Using Hybrid branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches", *Proceedings of the 23th Annual International Symposium on Computer Architecture* 1996.

[4] A. Uht and V. Sindagi, "Disjoint Eager Execution: An Optimal Form of Speculative Execution", *Proceedings of the 23th International Symposium on Micorarchitecture, pp.313-325*, 1995.

[5] P. Y. Chang, E. Hao, T. Y. Yeh, and Y. N. Patt, "Branch Classification: A New Mechanism for Improving Branch Predictor Performance", *Preceedings of the 27th Annual International Symposium on Microarchitecture*, Nov 1994.

[6] S. McFarling and J. Hennessy, "Reducing the Cost of Branches", *Proceedings of the 13th International Symposium on Computer Architecture*, 1986.

[7] R. Nair, "Optimal 2-Bit Branch Predictors", *IEEE Transactions on Computers*, May 1995.

[8] P. Y. Chang, E. Hao, and Y. N. Patt, "Alternative Implementation of Hybrid Branch Predictors", *Preceedings of the 28th ACM/IEEE International Symposium on Microarchitecture*, Nov 1995.

[9] K. M. Kavi, "Branch Folding For conditional Branches", *IEEE Computer Society Technical Committee on computer Architecture Technical Committee Newsletter*, pp.4-7, Dec 1997.

[10]이상원 외, "단일 칩 다중프로세서의 설계", 대한 전자공학회 추계 학술대회 발표 논문집, 1998.