고성능 마이크로프로세서에서 순차적 값 예측 실패 복구 방식

전 병 찬*, 박 희 룡**, 이 상 정***

*청운대학교 컴퓨터과학과,**김천대학 컴퓨터정보처리계열,***순천향대학교 정보기술공학부

e-mail jbc66@cwunet.ac.kr, hrpark@kimcheon.ac.kr, sjlee@sch.ac.kr

Sequential Value Misprediction Recovery Mechanism in High Performance Microprocessors

Byung-Chan Jeon , Hee-Ryong Park, Sang-Jeong Lee

Dept. of Computer Science, Chungwoon University

Dept. of Computer Information Process, Kimcheon College

Div. of Information Technology Engineering, Soonchunhyang University

요 약

고성능 슈퍼스칼라 프로세서에서 값 예측 실패 시에 잘못 예측된 값을 사용하여 모험적으로 수행된 명령들만을 순차적으로 취소하고 복구한 후에 재이슈하는 값 예측 실패 복구 메커니즘을 제안한다. 제안된 복구 방식은 값 예측이 틀린 종속명령만을 선택적으로 재이슈하여 불필요한 재이슈를 줄임으로써 값 예측 실패 시에 손실을 줄인다. 또한 기존의 방식들처럼 잘못 예측된 명령에 종속적인 명령들의 한번에 병렬로 검색하지 않고 명령들의 종속체인을 따라 순차적으로 검색함으로써 프로세서의 클럭 사이클에 영향을 미치지 않으면서 하드웨어의 구현의 복잡성을 줄인다.

1. 서론

고성능 슈퍼스칼라 프로세서에서 높은 성능을 도달하기 위해서는 명령어 수준 병렬성(Instruction Level Parallelism, ILP)을 이용하여 다수의 명령을 동시에 이슈하고 처리해야 한다. 따라서, 최근에는 실행되는 명령의 결과 값을 미리 예측하고, 이후 데이터종속 관계가 있는 명령들에게 값을 조기에 공급하고 이들 명령들을 모험적으로 실행하여 성능향상을 꾀하는 값 예측(value prediction)방식에 관하여 활발히 연구가 진행되고 있다. 기존의 많은 논문에서 값 예측으로 성능이 크게 향상됨을 보였다. 그러나 모험적 실행의 성격 상 값 예측이 실패하여 잘못된 값에 근거하여 데이터 종속관계를 끊고 모험적으로 이슈될 때 마다 복구를 한 후 올바른 값을 가지고 다시 이슈해야 문제점이 있다.

본 연구에서는 고성능 슈퍼스칼라 프로세서에서 값 예측 실패 시에 잘못 예측된 값을 사용하여 모험적으로 수행된 명령들만을 순차적으로 취소하고 복구한 후에 재이슈하는 값 예측 실패 복구 메커니즘을 제안한다. 제안된 복구 방식은 값 예측이 틀린 종속명령만을 선택적으로 재이슈하여 불필요한 재이슈를 줄임으로써 값 예측 실패 시에 손실을 줄인다. 또한 기존의 방식들처럼 잘못 예측된 명령에 종속적인 명령들의 한번에 병렬로 검색하지 않고 명령들의 종속체인을 따라 순차적으로 검색함으로써 프로세서의 클럭 사이클에 영향을 미치지 않으면서 하드웨어의 구현의 복잡성을 줄인다.[1)]

2. 값 예측 실패 복구방식

본 연구에서는 SimpleScalar 프로세서 모델[1]에 근간을 둔 비순차 이슈 슈퍼스칼라 프로세서 상에서 값 예측기 및 제안된 값 예측 실패 복구기법을 설계하였다. SimpleScalar 모델은 명령페치(Instruction Fetch, IF),명령 디코드 및 디스페치(Instruction Decode & Dispatch, ID),명령 이슈 및 수행(Instruction Issue & Execution, EX),결과저장(Writeback, WB) 및 완료(commit, CM)의 5개 파이프라인 스테이지로 동작한다. 명령의 값 예측을 이용한 파이프라인 단계는 다음과 같다. IF 스테이지에서 예측기를 참조하여 결과값을 예측하고 ID 스테이지에서는 이 예측값을 사용하는 종속명령들의 소스 오퍼랜드를 가용한 것으로 표시하고 디스페치한다. EX 단계에서는 예측값을 사용한 종속명령들을 모험적으로 이슈하고 수행한다. 그리고 결과 값이 예측된 명령이 수행 완료된 경우 WB 스테이지에서 예측의 성공 여부를 검증하여 예측이 성공한 경우에는 계속 진행하고 예측이 실패한 경우에는 잘못된 예측명령을 사용하여 모험적으로 이슈되어 수행된 종속명령들의 결과를 취소하고 다시 이슈하는 복구 과정을 수행한다. 이러한 복구 기법으로는 명령 재페치(instruction refetch) 방식과 선택적 재이슈(selective instruction reissue) 방식이 있다[5].

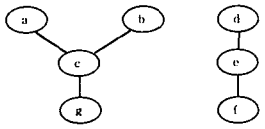
그림 1의 (a)는 SimpleScalar PISA 명령어 세트로 표시된 임의의 코드이며 (b)는 데이터 종속관계를 나타낸다. (c)는 4개의 명령을 각각 페치,이슈 및 수행할 수 있는 머신에서 열의 명령에 대해 행의 클럭 사이클에서 각 파이프라인 단계의 진행 상황을 보여주는 그림이다. IF*에서는 페치 스테이지에서 값 예측을하며 WB*에서는 예측 결과의 검증을 한다. EX$, WB$는 모험적으로 수행된 상태를 표시한다. 그림 1에서 명령 a,b,d가 레지스터 r3,r5,r7의 결과 값을 예측하고 이 중 명령 a만 예측이 틀리고 나머지 b,d는 올바른 예측을 한 경우를 가정하자. 또한 메모리 로드명령(lw)은 2 클럭이 소요된다고 가정한다. 클럭 3에서 명령 c는 명령 a,b의 예측 값을 이용하여 모험적으로 수행되고 명령 e도 d의 예측 값을 이용하여 모험적으로 이슈된다. 클럭 5에서 명령 a가 수행이 완료되어 예측이 틀린 것으로 검증된다. 이때 명령 재페치 방

본 연구는 한국과학재단 목적기초연구(2000-1-30300-008-3)지원으로 수행되었음.

식은 a의 이후 모든 명령 b,c,d,e,f,g에 대해 수행을 취소하고 다시 페치하여 이슈한다. 명령 재이슈 방식은 명령 a에 종속적인 c와 g 명령 만을 취소하고 재이슈하며 나머지 올바른 명령들은 취소하지 않고 계속 진행한다.

```
a. lw      r3,4(r2)      ; r3 = M[r2+4]
b. sll     r5,r4,1       ; r5 = r4 << 1
c. and     r6,r3,r5      ; r6 = r3 & r5
d. lw      r7,8(r2)      ; r7 = M[r2+8]
e. andi    r8,r7,8       ; r8 = r7 & 8
f. addiu   r10,r8,16     ; r10 = r8 + 16
g. beq     r6,r0,next    ; if(r6==0)goto next
```

(a)



(b)

		명령 a	b	c	d	e	f	g
클	1	IF*	IF*	IF	IF*			
럭	2	ID	ID	ID	ID	IF	IF	IF
	3	EX	EX	EX$	EX	ID	ID	ID
사	4	EX	WB*	WB$	EX	EX$		
이	5	WB*			WB*	WB		EX$
클	6	CM	CM				EX$	WB$

(c)

그림 1. 값 예측 코드 예: (a) 프로그램 코드 (b) 데이터 종속 관계 그래프 (c) 파이프라인 수행

3. 순차적 값 예측 실패 복구 기법

값 예측과 선택적인 재이슈 지원을 위해 명령 윈도우인 RUU의 엔트리에 부가적인 필드가 추가된다. 그림 2는 값 예측과 순차적이고 선택적인 재이슈를 지원하는 RUU 엔트리 구조를 보여주는 그림이고 진하게 표시된 부분이 값 예측과 선택적 복구를 위해 추가된 필드이다. 그림 2에서 각 소스 오퍼랜드에서 ready 비트는 소스 오퍼랜드의 가용 여부를 표시하며 이전 명령의 수행으로 소스 오퍼랜드가 이미 수행되었거나 또는 값 예측된 경우 TRUE로 세트된다. spec(speculation) 비트는 소스 오퍼랜드의 경우 예측된 값을 참조한 경우 TRUE로 세트되고, 데스티네이션 오퍼랜드인 경우에는 예측된 값이거나 소스 오퍼랜드 중의 하나의 spec 비트가 TRUE이면 TRUE로 세트되어 명령의 생성된 결과가 예측 값을 이용해 모험적 실행으로 생성된 결과임을 표시한다. data 필드에는 각 오퍼랜드의 값으로 최종 값이거나 예측 값 또는 모험적 실행으로 생성된 결과 값이 저장된다. issued 비트는 명령의 이슈여부를 표시하고 completed 비트는 명령의 수행이 완료되었음을 표시하는 비트이다. reissued 비트는 잘못된 예측 값을 사용하여 명령이 다시 이슈 되었음을 나타내는 비트이다. squash counter는 잘못된 명령으로 현재 수행 중인 명령을 취소하기 위해 사용되는 3 비트 카운터이다. 수행 중인 명령이 취소될 명령으로 판명되면 카운터는 증가하고 WB 스테이지에서 취소되었으면 감소한다. 그림 3은 WB 스테이지에서의 복구 동작을 나타낸 그림이다. 그림 4는 NS 스테이지에서의 값 예측 복구 동작을 보여주는 그림이다.

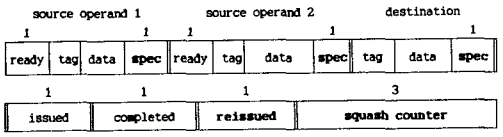


그림 2. 확장된 RUU 엔트리

```
1  // assume the execution of instruction i is completed
2  // and its execution result is result_i
3
4  // squash the incorrect issued instruction
5  if (ruu_i.squash > 0) {
6        ruu_i.squash--;
7        squash the execution of instruction i.
8        return;
9  }
10 ruu_i.dest.data = result_i;
11 ruu_i.completed = TRUE;
12 if (ruu_i.src_1.spec || ruu_i.src_2.spec)
13       ruu_i.dest.spec = TRUE;
14 if(instruction i is the mispredicted branch instruction
     && !ruu_i.dest.spec) {
15       recover mispredicted branch prediction;
16       return;
17 }
18  //  assume instruction j in ruu is data dependent on the
      instruction i
19 // and its source n(n=1,2) uses the result value of i
20 // broadcast the result to the waiting instruction j
21 if (!ruu_j.issued && !ruu_j.reissued) {
22       ruu_j.src_n.ready = TRUE;
23       ruu_j.src_n.data = result_i;
24       if(ruu_i.dest.spec) // if i was executed speculatively
25             ruu_j.src_n.spec=TRUE;  //propagate the speculative
                                        flag of i into j
26 }
27 // if j is already issued and i was executed non-speculatively
28 else if (!ruu_i.dest.spec) {
29      // propagate the non-speculative flag of i into j
30      ruu_j.src_n.spec = FALSE;
31     //if j referenced the mispredicted or incorrect value of i
32     if (ruu_j.src_n.data != result_i) {
33           ruu_j.issued = FALSE;
34           ruu_j.reissued = TRUE;    // for reissue
35           ruu_i.src_n.data = result_j; //result broadcast
36           if(!ruu_j.completed)//if j is executing and not completed
37                 ruu_j.squash++; // increment squash counter for
                                      squashing
38     }
39     //  if j referenced correct value of i
40     else {
41           if (!ruu_j.src_1.spec && !ruu_j.src_2.spec) {
42                 ruu_i.dest.spec = FALSE;
43                 // non-speculation flag will propagate through data
                     dependency
44                 //  chain in non-speculation queue
45                 insert instruction j into NS queue;
46                 if(instruction j is a mispredicted branch)
47                     recover mispredicted branch prediction;
48           }
49     }
50 }
```

그림 3. WB 스테이지에서의 값 예측 복구 동작

```
1  get instruction j from the NS queue;
2
3  //  assume instruction k in ruu is data dependent on the
     instruction j
4  //  and its source n(n=1 or 2) uses the result value of j
5
6  //  if k referenced the mispredicted or incorrect value of j
7  if (ruu_k.src_n.data != ruu_j.dest.data) {
8        if(!ruu_k.issued)
9              ruu_k.src_n.ready = TRUE;
10       else {
11             ruu_k.issued = FALSE;
12             ruu_k.reissued = TRUE;
13       }
14       //  result broadcast
15       ruu_k.src_n.data = ruu_j.dest.data
16       //  propagate the non-speculative flag of i into j
17       ruu_k.src_n.spec = FALSE;
18 }
19 // if k referenced correct value of j
20 else {
21       // propagate the non-speculative flag of i into j
22       ruu_k.src_n.spec = FALSE;
23       if (!ruu_k.src_1.spec && !ruu_k.src_2.spec) {
24             ruu_k.dest.spec = FALSE;
25             //  non-speculation flag will propagate through
                   data dependency chain in
26             //  non_spec_prop stageinsert instruction k into
                       non-speculation queue;
27             insert instruction k into NS queue;
28             if(instruction k is a mispredicted branch)
29                   recover mispredicted branch prediction;
30       }
31 }
```

그림 4. NS 스테이지에서의 값 예측 복구 동작

그림 5는 앞의 그림 1의 (a) 코드 예에서 명령 a,b,d가 값을 예측하고 명령 a,b는 예측이 실패하고 명령 d만 값 예측이 성공한 경우를 가정하여 제안된 값 예측 실패 복구 기법을 적용한 예이다. 그림에서 EX!는 재이슈되어 수행됨을 표시한다. 먼저 (1,a) (1,b) (1,d) - (i,j)는 클럭 사이클 i에서의 명령 j의 파이프라인 상태를 표시 - 에서 값을 예측한다. (3,c)에서 a, b의 예측값을 사용하여 명령 c가 모험적 이슈 수행되고, (4,b)에서 b의 예측 실패가 검증 되어 c의 재이슈 비트 세트하고 (5,c)에서 c는 재이슈된다. (4,e)에서 d의 예측값을 사용하여 명령 e가 모험적으로 이슈 수행된다. (5,a)에서 a의 최종 값이 예측 실패로 검증되어서 실행 중인 c를 취소하고 명령 c의 재이슈를 위해 reissued 비트를 TRUE로 세트하고 (6,c)에서 재이슈된다. (5,g)에서 명령 g는 (4,c)에서 모험적으로 수행된 명령 c의 결과가 전파되어 모험적으로 수행된다.

클럭 사이클	명령 a	b	c	d	e	f	g
1	IF*	IF*	IF	IF*			
2	ID	ID	ID	ID	IF	IF	IF
3	EX	EX	EX$	EX	ID	ID	ID
4	EX	WB*	WB$	EX	EX$		
5	WB*		EX$	WB*	WB$		EX$
6	CM	CM	EX!		NS	EX$	WB$
7			WB			WB$	
8			CM	CM	CM	CM	EX!
9							WB
10							CM

그림 5. 그림 2의 (a)의 코드에 대한 순차적 값 예측 복구 예

(6,e)에서는 (5,d)에서 NS 큐에 삽입된 e에 대해 비모험적 상태 플래그를 f에 전파한다. (6,f)에서 명령 f는 (5,e)에서 전파된 모험적 수행 값을 가지고 모험적으로 이슈를 시작하고 (7,f)에서 수행을 완료한다. 그리고 잘못 수행된 종속명령 g를 다음 사이클에 재이슈하도록 reissued 비트를 세트한다. 명령 g는 (9,g)에서 재이슈되어 수행된다. 만약 분기명령 g가 잘못 분기예측된 명령이라면 (9,g)에서 분기예측 실패 복구 과정을 시작한다. 즉, 모험적 수행 상태인 (6,g)에서는 분기예측 실패 복구를 하지 않고 비모험적 상태를 판명될 때 복구를 한다.

4. 성능측정 및 분석

제안된 순차적 값 예측 실패 복구 메커니즘(value misprediction recovery mechanism)의 성능을 측정하기 위해서 SPECint95와 SPECint2000 벤치마크를 적용하여 성능을 분석하였다. 그림 6은 4, 8, 16의 각 이슈에 대해 제안된 순차적인 복구 기법을 적용하여 값 예측을 한 경우에 값 예측을 하지 않은 경우에 비해 향상된 성능향상을 보여주는 그림으로 4, 8, 16 이슈에 대해 각각 평균 1.1%, 5.1%, 5.4%의 성능 향상이 있음을 알 수 있다. 즉 이슈되는 명령 수가 커지면 값 예측을 이용한 모험적 실행이 더 효과가 있음을 알 수 있다. 그림 7은 8 이슈 머신 구성에서 값 예측을 하지 않는 경우 페치된 명령의 비율을 1로 보았을 때 병렬검색(Parallel8), 명령 재페치(Refetch8) 및 제안된 순차적인 복구 방식(VP8)의 명령 페치 비율을 측정한 그림이다. 병렬검색 방식, 제안된 방식, 명령 재페치 방식이 각각 평균 0.82, 0.95, 1.84로 병렬검색 방식과 제안된 방식은 페치된 명령의 수가 오히려 줄었음에 반해 재페치 방식은 크게 증가함을 알 수 있다.

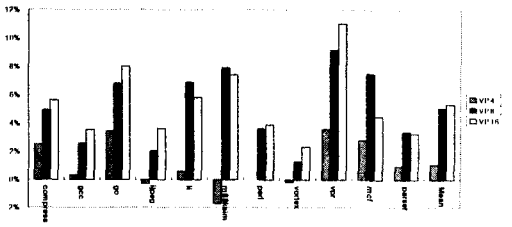


그림 6. 값 예측 시 성능 향상

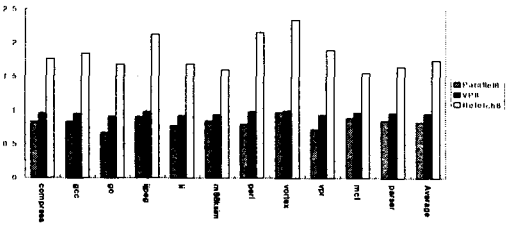


그림 7. 각 값 예측 복구방식 적용 시 페치된 명령 비율

5. 결론

본 논문에서는 고성능 슈퍼스칼라 프로세서에서 값 예측 실패 시에 잘못 예측된 값을 사용하여 모험적으로 수행된 명령들만을 순차적으로 취소하고 복구한 후에 재이슈하는 값 예측 실패 복구 메커니즘을 제안하였다. 실험결과 값 예측시 성능향상은 4, 8, 16 이슈에 대해 평균 1.1%, 5.1%, 5.4%의 성능향상을 알 수 있다. 또한 병렬검색 방식과 제안된 방식은 페치된 명령의 수가 오히려 줄었음에 반해 재페치방식은 크게 증가함을 알 수 있었다.

[참고문헌]

[1] D.Burger and T.Austin, The SimpleScalar tool set, version 2.0 ,Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997

[2] S. McFarling, "Combining branch predictors," Technical Report TN-36, Digital Western Research Laboratory, June 1993.

[3] K.Wang, M.Franklin, "Highly accurate data value predictions using hybrid predictor," Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30), Dec. 1997.

[4] T. Yeh and Y.Patt, "Two-level adaptive branch prediction," Proceedings of the 24th International Symposium Microarchitecture (MICRO-24), Nov. 1991.

[5] H. Zhou, C. Fu, E. Rotenberg, T. Conte, "A study of value speculative execution and misspeculation recovery in superscalar microprocessors", Technical Report, ECE Department, N.C. State University, Jan., 2001