공학 연구논문집 제26권 제1호 pp. 345~352, 울산대학교
Journal of Engineering Research Vol. 26, No. 1, pp. 345~352 1995.
University of Ulsan.

BTB(Branch Target Buffer)에서의 DTMR(Design Target Miss Ratios)의 유도

구자록
전자계산학과

<요 약>

컴퓨터 시스템을 설계함에 있어서 설계변수들에 따르는 성능평가를 추정하기 위한 기준값(이 논문에서는 BTB DTMR)을 구하는 것은 매우 중요하다. 이를 위해 Cache miss rate와 BTB miss rate의 관계를 규명한다.

Derivation of DTMR(Design Target Miss Ratios) in the BTB(Branch Target Buffer)

Koo, Ja-Rok
Dept. of Computer Science

<Abstract>

We provide for the computer system designers a set of numbers(called BTB DTMR) that they can use to estimate the performance impact of certain design choices. To derive the BTB DTMR, this paper discusses the relationships between Cache miss rate and BTB miss rate.

---

1. 개 요

파이프라이닝은 싱글프로세서의 성능을 향상시킬 수 있는 중요한 구조기법 중의 하나이다. 파이프라이닝은 하나의 명령어 실행을 파이프라인 단계(pipeline stage)라 불리는 여러 개의 조각으로 나눈다. 전형적인 파이프라인 구조는 instruction fetch, instruction decode, operand fetch, execution, result writeback의 5단계로 되어 있다. 각각의 파이프라인 단계의 작동시간은 거의 동일해야 하는데, 이는 파이프라인을 통하여 진행되는 명령어들의 실행속도가 가장 느린 파이프라인 단계에 의해 결정되기 때문이다. 이론상으로는 이러한 최적의 경우에는 파이프라인을 갖추지 않은 경우보다 5배의 실행속도를 얻을 수 있다[Lee 84, McFarling 88].

그런데 CPU 파이프라이닝을 설계함에 있어서 근본적인 문제점 중의 하나가 branch로 인한 파이프라인의 성능저하이다. branch명령어를 실행하기 위해서는 프로세서는 분지가 될 것인지를 결정하여 branch target address를 계산해야하고, 분지가 일어난다면, target address로부터 명령어를 가져오게 된다. 파이프라인 프로세서는 분지의 방향이 결정되기 전에 분지의 방향을 알아야 하기때문에 파이프라인 성능에 영향을 미친다. 그래서, 파이프라인 프로세서는 branch를 만나게 되면, 더 이상의 명령어를 가져오기 전에 branch 명령어의 실행이 끝날 때까지 기다리거나, 경우에 따라서는 잘못된 명령어를 가져올 수 있을지라도 계속되는 명령어를 가져와 실행을 진행시키는 경우의 선택을 해야한다. 각각의 경우 모두 성능을 저하시킬 수 있지만, 두 번째의 경우가 상대적으로 적게 영향을 미친다. 특히 분지의 방향을 정확히 예측할 수 있으면 더욱 뛰어난 기법이 될 수 있다[McFarling 88, Perleberg-Smith 93].

Branch Target Buffer(BTB)는 분지의 방향을 예측하고 branch에 관한 정보를 보관함으로써 branch로 인한 성능저하를 줄이기 위해 사용된다[Lee 84]. 4가지 형태의 정보가 BTB에 저장이 되는데, branch address를 나타내는 tag와 분지의 방향을 예측하는 정보, branch target address, 그리고 branch target address에 해당되는 명령어로 구성된다. 이러한 4가지 형태의 정보로 이뤄진 BTB는 다음과 같이 작동을 한다. 메모리로부터 각각의 명령어를 가져올 때 명령어의 번지가 BTB의 index로 사용되어, 그 번지에 해당하는 유효한 BTB entry가 존재하면 그 명령어는 branch명령어가 된다. 이 branch 명령어는 BTB에 저장되어 있는 예측정보를 이용하여 분지될 방향을 결정한다. 만약 branch의 분지가 taken으로 예측이 되면 BTB에 저장된 branch target address 및 branch target address에 해당하는 명령어를 가져와서 실행하면 된다. 만약 branch의 분지가 not-taken으로 예측이 되면 프로세서는 branch 다음의 명령어를 가져오게 된다. 그래서 branch의 실행이 끝나면 프로세서는 BTB가 branch의 실행을 제대로 예측했는지를 확인하여야 한다. 제대로 끝났으면 프로세서는 계속해서 명령어를 실행하지만, branch의 잘못된 예측이 밝혀지면 프로세서는 파이프라인에 존재하는 모든 명령어를 취소(flushing)하고 정확한 branch의 방향에 있는 명령어들을 실행해야 한다. 어떠한 경우든 branch 예측정보와 branch target address는 계속해서 갱신되어야 한다.

2. trace를 이용한 시뮬레이션

Instruction stream과 branch의 실행에 대한 일반적인 모델이 없기때문에 여기서는 trace를 이용한 시뮬레이션 기법을 사용한다. 계층적인 메모리의 성능평가에 trace를 이용한 시뮬레이션이 자주 사용된다. 이 시뮬레이션 기법은 반복적으로 사용될 수 있으며, 캐쉬설계에 고려되는 변

수들의 변화에 따른 효과를 유도할 수 있다. 이러한 시뮬레이션 기법은 또한 하드웨어 모니터링 기법에 비하여 경제적으로 경비가 적게 들며, 연구하고자하는 대상기계가 없어도 실험이 가능하다. 시뮬레이션 결과는 분석적인 모델 기법(analytic model)에서는 단순화된 가정이 없이는 불가능한 여러 경우에 대해서도 잘 유도되어 진다 [Dinero 90].

이 논문에서 참조하고 있는 6개의 workload는 32개의 프로그램 trace로 구성되어 있는 데, 각각의 workload는 COMP(MIPS compiler related traces), FP(MIPS floating point traces), TEXT (MIPS text processing traces), SPARC (SPARC traces), 68k(68010 traces), 그리고 VAX(VAX traces)로 이뤄져 있다 [Perleberg 89-b].

Cache 시뮬레이터로서는 dinero III cache simulator를 사용한다. dinero III cache simulator는 stdin으로부터 din형태로 된 trace입력을 읽는다. din레코드는 label과 address로 된 쌍으로 이뤄져 있다. trace 화일의 각각의 줄은 한 개의 din레코드를 갖는다. 각 줄의 나머지 부분은 무시되는 데, trace화일에 코멘트가 들어갈 수 있도록 되어 있다. 각각의 label은 메모리 참조형태를 나타낸다.

0 -- 데이타 읽음(read data).
1 -- 데이타 씀(write data).
2 -- 명령어 페치(instruction fetch).
3 -- escape record(형태 미정).
4 -- escape record(cache flush를 유도).

address는 0 에서 ffffffff 사이의 16진수 형태의 byte단위의 adress이다.

캐쉬변수들은 command line option에 의해 정해진다. 변수들 중에서 block size 와 unified cache size 또는 data cache size 및 instruction cache size는 반드시 명시되어져야 한다. 그외 다른 변수들은 선택적이다. [표-1]은 average workload와 trace의 통계치를 나타낸다.

[표-1] average workload와 trace의 통계치

	COMP	TEXT	FP	SPARC	VAX	68k	Work Ave	Trace Ave
CPU	MIPS	MIPS	MIPS	SPARC	VAX	68k		
Dynamic Instr	2261788	2428066	2439411	1976181	1045501	681475	1805404	1666244
Dynamic Branch	401927	487369	247695	433537	283996	157743	335377	317079
Dynamic Basic Block Size	5.65	5.00	10.38	4.60	3.38	4.27	5.55	5.26

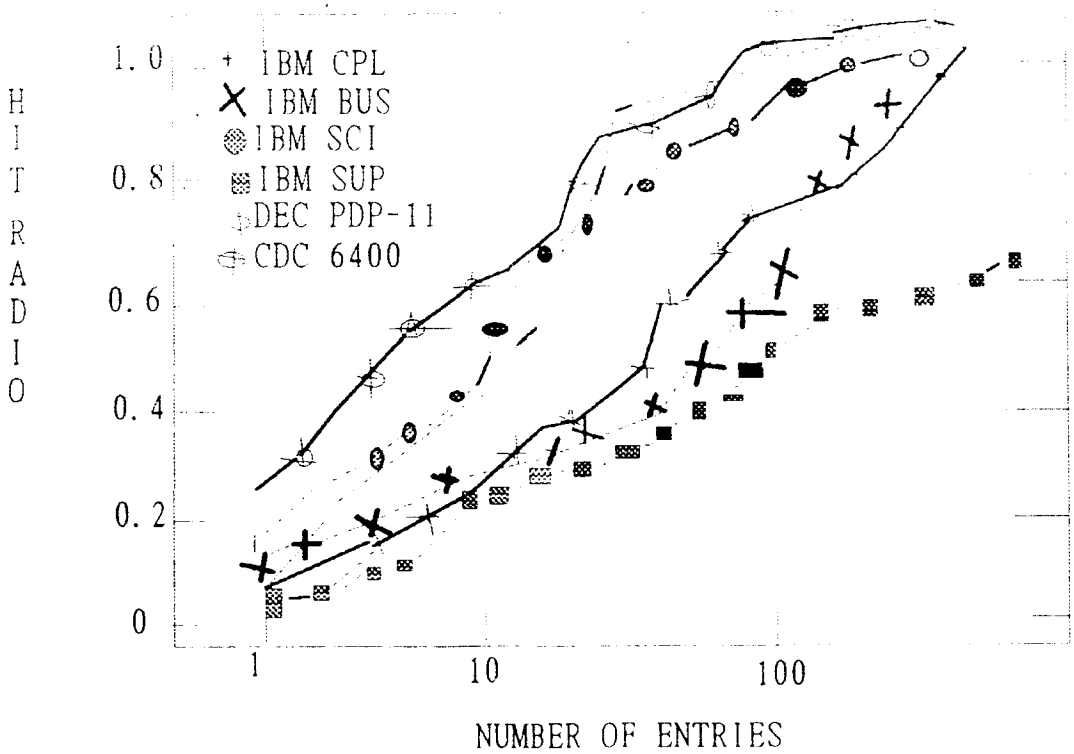
참고)

Dynamic Inst = 프로그램 실행시 페치된 명령어의 총 수
Dynamic Branches = 프로그램 실행시 페치된 branch명령어의 총 수
Dynamic Basic Block Size = 프로그램 실행시 페치된 basic block

3. Cache DTMR(Design Target Miss Ratios)

컴퓨터 시스템을 설계함에 있어서 설계 변수들에 따르는 성능평가를 추정하기 위한 기준값을 구하는 것은 매우 중요하다. 특히 BTB miss ratio를 BTB size에 대한 함수그래프로써 나타내고자 한다. 이러한 작업은 Cache miss ratio를 구함에 있어서 cache size와 line size를 변수로한 함수그래프로 나타낸 논문 [Smith 87]을 참고로 하였다. 이 실험[Smith 87]에서 관측된 Cache miss ratio는 workload에 따라 매우 가변적이다. 또한 BTB에서도 workload에 따라 branch명령어의 시간적인 지역성(temporal locality)의 변동폭이 매우 큼을 [그림-1]에서 알 수 있다[Lee 84]. 그래서 여기서는 대표되는 standard BTB miss ratios를 구하는 방법에 대하여 고려해 보고자한다. 다음 [표-2]은 Cache Design Target Miss Ratios를 나타내는데, 3가지의 Cache type(Unified, Instruction, Data Cache)에 대해, cache line size가 4바이트에서 128바이트까지, cache size가 32바이트에서 32768바이트까지 변함에 따른 miss ratio를 나타내고 있다.

[그림-1] BTB hit ratios



[표-2] Cache Design Target Miss Ratios

Cache Type	Miss Ratio					
Unified	Line Size:					
Size	4	8	16	32	64	128
32	0.717	0.556	0.500	0.750		
64	0.686	0.488	0.400	0.480	0.720	
128	0.674	0.467	0.350	0.330	0.428	0.686
256	0.643	0.420	0.300	0.258	0.276	0.386
512	0.596	0.390	0.270	0.216	0.197	0.257
1024	0.473	0.309	0.210	0.162	0.137	0.151
2048	0.405	0.258	0.170	0.124	0.098	0.093
4096	0.329	0.193	0.120	0.082	0.059	0.050
8192	0.232	0.135	0.080	0.050	0.033	0.025
16384	0.182	0.103	0.060	0.036	0.023	0.016
32768	0.124	0.070	0.040	0.024	0.014	0.009
Cache Type Instructions						
32	0.725	0.478	0.330	0.247		
64	0.674	0.438	0.300	0.222	0.191	
128	0.615	0.397	0.270	0.197	0.164	0.157
256	0.592	0.373	0.250	0.177	0.138	0.129
512	0.562	0.348	0.230	0.159	0.119	0.108
1024	0.504	0.308	0.200	0.134	0.098	0.084
2048	0.391	0.234	0.150	0.098	0.068	0.057
4096	0.271	0.161	0.100	0.063	0.043	0.032
8192	0.172	0.100	0.060	0.037	0.023	0.016
16384	0.148	0.085	0.050	0.029	0.018	0.012
32768	0.091	0.052	0.030	0.017	0.010	0.007
Cache Type Data						
32	0.731	0.611	0.550	0.715		
64	0.660	0.515	0.450	0.495	0.693	
128	0.561	0.412	0.350	0.351	0.467	0.677
256	0.470	0.337	0.280	0.272	0.326	0.456
512	0.345	0.246	0.200	0.191	0.215	0.282
1024	0.283	0.211	0.160	0.138	0.140	0.161
2048	0.256	0.169	0.120	0.094	0.083	0.089
4096	0.247	0.153	0.100	0.070	0.054	0.048
8192	0.214	0.129	0.080	0.053	0.039	0.032
16384	0.161	0.097	0.060	0.039	0.026	0.019
32768	0.108	0.065	0.040	0.025	0.017	0.012

4. BTB DTMR의 유도

BTB miss ratio는 예측정확도와 더불어 branch가 성공적으로 정확하게 예측될 수 있도록 결정하는 중요한 요소중의 하나이다. BTB와 instruction cache miss ratios의 관계가 파악되면 BTB DTMR을 유도할 수 있을 것이다. standard Instruction-Cache miss ratios로 부터 standard BTB miss ratios를 유도하기 위해 다음과 같은 가정을 하는데, BTB는 branch만을 저장하는 instruction cache로 간주할 수 있기때문이다.

1) 하나의 cache line은 하나의 명령어만을 갖는다.

2) 그러면 cache size는 BTB size(entry의 수)와 average basic block size의 곱으로 정해진다.(basic block은 branch명령어로 시작되어 다음 번 branch명령어가 나오기 직전까지의 명령어들로 이뤄진 프로그램의 한 부분을 의미한다.)

BTB 시뮬레이터는 LRU replacement기법을 갖는 4-way set associative 맵핑의 구조를 하며 line size를 4 bytes로 가정한다. 또한 average basic block size를 n (instructions)이라고 하면 각각의 basic block마다 한개의 branch가 존재하게 된다. 즉, BTB에서 1개의 miss가 발생하면, Cache에서는 n개의 misses가 발생하게되므로 다음의 식의 유도에 의해, 하나의 workload에서는 BTB miss rate와 cache miss rate가 동등하게 취급될 수 있을 것이다.

Cache miss ratio와 BTB miss ratio는 다음과 같다.

$$Cache\ miss\ ratio = \frac{no.\ of\ misses}{Dynamic\ Inst},$$

$$BTB\ miss\ ratio = \frac{no.\ of\ misses\ \text{for}\ branches}{Dynamic\ branches}.$$

여기서, Cache miss ratio와 BTB miss ratio의 비를 구하면 다음과 같다.

$$\frac{Cache\ miss\ ratio}{BTB\ miss\ ratio} = \frac{no.\ of\ misses}{no.\ of\ misses\ \text{for}\ branches} \times \frac{Dynamic\ Branches}{Dynamic\ Inst}$$

그런데, no. of misses와 no. of misses for branches의 비는 n이고, [표-1]의 Dynamic Branches와 Dynamic Inst의 값을 대입하여 계산하면,

$$\frac{Cache\ miss\ ratio}{BTB\ miss\ ratio} = n \times \frac{317,079\ instructions}{1,666,244\ instructions}$$ 가 된다.

여기서, BTB miss ratio를 유도하면,

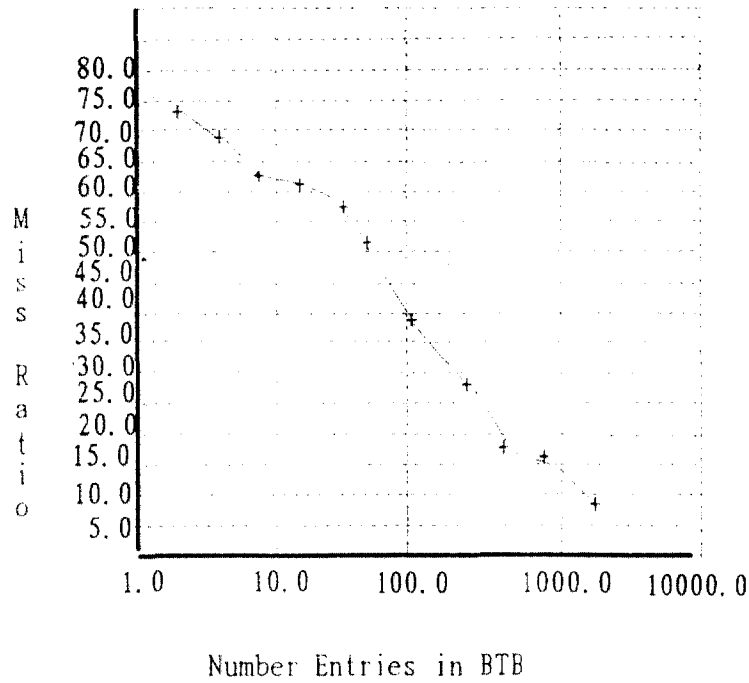
$$BTB\ miss\ ratio = \frac{Cache\ miss\ ratio \times 5.2549}{n}$$ 가 된다.

그런데, n은 average basic block size로서 [표-1]의 Dynamic Basic Block Size(5.26)를 의미하므로, 거의 BTB miss ratio = Cache miss ratio 가 유도된다.

참고) [표-1]에 의해,
Dynamic Inst = 1,666,244
Dynamic Branches = 317,079
Dynamic Basic Block Size = 5.26(instructions)

따라서, [표-2]의 Instruction Cache type에서 line size가 4인 경우의 miss ratios만을 사용하여, BTB element(1 element = 4 바이트)에 따른 miss ratios를 그래프로 그리면 [그림-2]과 같이 BTB DTMR을 유도할 수 있다.

[그림-2] BTB Design Target Miss Ratio



5. 결 론

컴퓨터 시스템을 설계함에 있어서 설계변수들에 따르는 성능평가를 추정하기 위한 기준값을 구하는 것은 매우 중요하다. 특히 BTB DTMR(Design Target Miss Ratios)을 제공함으로써 BTB를 설계함에 있어서 설계변수들(BTB size, mapping 기법, line size, no. of entries 등)에 따른 BTB의 성능, 나아가 BTB를 이용한 파이프라인 프로세서의 성능을 예측하는 데 유용할 것이다. 이 논문에서는 기존의 Cache DTMR(Design Target Miss Ratios)을 이용한 BTB DTMR을 유도하였다. 이를 위해 cache line, cache size, program block size의 한계를 미리 지정했으며, BTB 시뮬레이터는 LRU replacement기법을 갖는 4-way set associative 맵핑의 구조의 line size가 4 bytes라고 가정하였다. 또한 average basic block size의 개념을 적용하였다.

6. 참고문헌

[Smith 82] Alan Jay Smith, "Cache Memories", in the Computing Surveys, vol.14, No.3, Sept. 1982, pp.473-530.

[Lee 84] Johnny K.F. Lee, Alan Jay Smith, "Branch Prediction Strategies and Branch Target Buffer Design", in Computer, Jan. 1984. pp.6-22.

[Smith 85] Alan Jay Smith, "Cache Evaluation and the Impact of Workload Choice", in the 12th Annual International Symposium on Computer Architecture, June 17-19, 1985, pp.64-73.

[Smith 87] Alan Jay Smith, "Line(Block) Size Choice for CPU Cache Memories", in the IEEE Transactions on Computers, vol. C-36. No.9, Sept. 1987. pp.1063-1075.

[McFarling 88] Scott McFarling, John Hennessy, "Reducing the Cost of Branches", in the 13th Annual Inter national Symposium on Computer Ar chitecture, June 2-5, 1986, pp.396-403.

[Perleberg 89-a] Chris H. Perleberg, Alan Jay Smith, "Branch Target Buffer Design and Optimization", in the U.C. Berkeley Comput. Sci. Divis ion Technical Rep. UCB/CSD 89/552, Dec. 1989.

[Perleberg 89-b] Chris H. Perleberg, "Branch Target Buffer Design", in the U.C. Berkeley Comput. Sci. Division Technical Rep. UCB/CSD 89/553, Dec. 1989.

[Dinero 90] DineroIII - cache simulator, version III, Aug. 1990.

[Cragon 92] Harvey G. Cragon, Branch Strategy Taxonomy and Performance Models, IEEE Computer Society Press, 1992.

[Perleberg-Smith 93] Chris H. Perleberg, Alan Jay Smith, "Branch Target Buffer Design and Optimization", in the IEEE Transactions on Computers, vol. 42. No.4, April. 1993. pp.396-412.