논문 2009-46CI-5-1

모드 선택 비트를 사용한 필터 캐시 예측기

(Filter Cache Predictor Using Mode Selection Bit)

곽 종 욱*

(Jong Wook Kwak)

요 약

캐시 에너지의 소비 전력을 줄이기 위해 필터 캐시가 제안되었다. 이와 같은 필터 캐시의 사용으로 인해 50% 이상의 전력 사용 감소 효과를 가져왔으나, 상대적으로 시스템 성능은 평균 20% 가량 감소되었다. 필터 캐시의 사용으로 인한 이 같은 성능 감소를 최소화하기 위해서, 여러 가지 형태의 필터 캐시 예측가 제안 되었다. 본 논문에서는 기존에 제안된 주요 필터 캐시 예측 모델들을 소개하며, 각각의 방식에 있어서의 핵심 특징 및 해당 방식의 문제점을 분석한다. 분석 결과, 필터 캐시의 참조 실패를 야기하는 기존 방식의 중요한 문제점을 확인하였으며, 이를 바탕으로 본 논문에서는 개선된 형태의 새로운 필터 캐시 예측기 모델을 제안한다. 제안된 방식은 MSB라 불리는 참조 비트를 고안하여 이를 기존의 필터캐시와 BTB에 새롭게 활용한다. 본 논문에서 제안된 방식의 성능을 검증하기 위해 SimpleScalar 시뮬레이터와 MiBench 응용 프로그램을 활용하여 모의실험을 수행하였다. 실험 결과 제안된 방식은 기존 방식 대비, 필터 캐시 예측 실패율, 필터 캐시 활용률 및 전력 소모량·시간 지연 등 모든 면에서 평균 5%의 성능 향상을 가져 왔다.

Abstract

Filter cache has been introduced as one solution of reducing cache power consumption. More than 50% of the power reduction results from the filter cache, whereas more than 20% of the performance is compromised. To minimize the performance degradation of the filter cache, the predictive filter cache has been proposed. In this paper, we review the previous filter cache predictors and analyze the problems of the solutions. As a result, we found main problems that cause prediction misses in previous filter cache schemes and, to resolve the problems, this paper proposes a new prediction policy. In our scheme, some reference bit entries, called MSBs, are inserted into filter cache and BTB, to adaptively control the filter cache access. In simulation parts, we use a modified SimpleScalar simulator with MiBench benchmark programs to verify the proposed filter cache. The simulation result shows in average 5% performance improvement, compared to previous ones.

Keywords : 저전력 캐시, 필터 캐시, 레벨 0 캐시, 명령어 인출 예측, 분기 명령어

I. 서 론

내장형 기기와 이동 기기 시장이 급격하게 팽창함에 따라 내장형 프로세서에 대한 관심도 커졌다. 내장형 프로세서는 거의 무한하게 전력을 공급받는 데스크톱용 프로세서나 서버용 프로세서와는 달리 전지에 의해서 제한된 전력만을 공급받기 때문에, 성능뿐만 아니라 전력 소모량도 매우 중요한 고려의 대상이 된다[1].

내장형 프로세서의 발전과 더불어, 캐시 시스템은 내장형 프로세서의 전력소모에 매우 큰 비중을 차지하게 되었다. 표 1에서 보이는 바와 같이, 에너지 효율을 고려하지 않은 초기 StrongARM 110의 경우, 프로세서 전력의 43%를 캐시 시스템이 사용하고 있다[2]. 캐시 시스템이 전력을 많이 소모한다는 것은 곧 전력 소모를 줄일 수 있는 여지 또한 많이 가지고 있다는 뜻이므로, 캐시 시스템은 내장형 프로세서를 위한 저전력분야에서

* 평생회원, 영남대학교 컴퓨터공학과
(Department of Computer Engineering, Yeungnam University)
※ 이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-331-D00470).
접수일자: 2009년5월25일, 수정완료일: 2009년9월4일

표 1. StrongARM110에서의 전력 소모
Table 1. Power Consumption in StrongARM110.

Instruction Cache	27%
Data Cache	16%
Instruction Decode	18%
Execution Logic	8%
Clock	10%
Others (MMU, Write Buffer, Bus Interface Unit)	21%

중요한 연구의 대상이 되어 왔다. 캐시 시스템에서의 전력 소모량 감소를 위해서 반도체 공정의 향상, 메모리 셀 디자인의 변화, 전압 감소, 캐시 구조의 최적화 등과 같은 여러 기법이 사용되어 왔다

그 중에서 캐시 구조의 최적화를 통해서 전력 소모를 줄이기 위한 연구 가운데 프로세서와 레벨 1 캐시 사이에 보조적인 장치(auxiliary device)를 추가하는 시도가 있었다. 블록 버퍼링(Block Buffering)[3] 기법은 가장 최근에 접근된 캐시 라인을 저장해 놓는 작은 버퍼를 추가하는 방법이다. 현재 실행되는 명령어의 바로 다음에 실행될 명령어가 같은 캐시 라인에 있게 되면, 버퍼만을 읽음으로써 전력 소모량과 실행 시간을 줄일 수 있는 방법이다. 필터 캐시(Filter Cache)[4]는 프로세서와 레벨 1 캐시 사이에 작은 캐시를 추가하여서 레벨 1 캐시의 접근을 줄임으로써 전력을 줄이도록 하는 기법이다. 이 추가된 작은 캐시를 필터 캐시 혹은 레벨 0 캐시(L0 Cache)라고 부른다. 필터 캐시는 주로 내장형 시스템, 특히 멀티미디어와 통신 응용 프로그램을 위한 프로세서를 대상으로 연구되어 왔다.

본 논문에서는 기존의 필터 캐시의 발전 과정을 살펴보고, 필터 캐시 연구에서 가장 많이 쓰이는 필터 캐시 예측기 모델(Filter Cache Predictor Model)을 분석해 본다. 이 분석을 바탕으로 기존 방식의 문제점을 해결하는 효율적인 필터 캐시 예측기를 제안한 뒤, 모의실험을 통해 성능의 향상을 알아본다. 이하, 본 논문의 구성은 다음과 같다. Ⅱ장에서는 기존의 필터 캐시의 연구에 대해서 알아보고, 필터 캐시 예측기 모델에 대해 소개한다. Ⅲ장에서는 필터 캐시 예측기 모델의 특징을 분석한다. 그리고 이를 바탕으로 본 논문에서 제안하는 모드 선택 비트를 이용한 예측기 모델을 소개하고, 그 구조 및 동작 방식에 대해서 논의한다. Ⅳ장에서는 모의실험을 수행하고, 실험 결과를 통해서 기존의 기법들과 비교 해본다. 마지막으로 Ⅴ장에서는 결론을 내린다.

Ⅱ. 배경 지식 및 관련 연구

1. 필터 캐시

프로그램의 전체적인 실행 속도를 높이거나 프로그램의 실행에 사용되는 전력 소모량을 줄이는 기술들은 많이 개발되어 왔다. 그러나 많은 경우 실행 속도를 높이면 전력 소모량이 늘어나거나, 전력 소모량을 줄이면 실행 속도가 낮아지게 된다. 필터 캐시(Filter Cache)[4]는 실행 속도 향상과 전력 소모량 감소라는 두 가지 목표를 동시에 추구하는 대신 상호절충(trade off)적인 상황을 받아들이고, 적절하게 조화를 이루는 것을 선택하였다. 그래서 필터 캐시는 그 연구 목표를 일반적인 프로그램에서는 실행 속도의 손실을 보더라도, 내장형 시스템에서 자주 사용되는 멀티미디어나 통신 알고리즘을 최적화하는데 초점을 두었다.

필터 캐시는 그림 1에서 나와 있듯이 기존의 캐시 시스템과 매우 비슷한 구조를 가진다. 기존의 구조에서는 프로세서가 레벨 1 캐시에 직접 접근하지만, 필터 캐시에서는 프로세서가 레벨 1 캐시에 접근하는 대신에 필터 캐시에 접근하게 된다. 만약 프로세서가 인출(fetch)하려는 주소의 명령어가 필터 캐시에 있는 경우에는 필터 캐시에서 명령어를 제공하게 된다. 반대로, 필터 캐시에 존재하지 않는 경우는 레벨 1 캐시로 접근하여서 해당 명령어를 가져오고, 다시 그 명령어를 프로세서로 전송하게 된다.

필터 캐시는 그 적중률(Hit Rate)에 비례해서 시스템의 전력 소모량 감소 효과는 커지게 되며, 필터 캐시의 실패율(Miss Rate)에 비례해서 시스템의 실행 속도

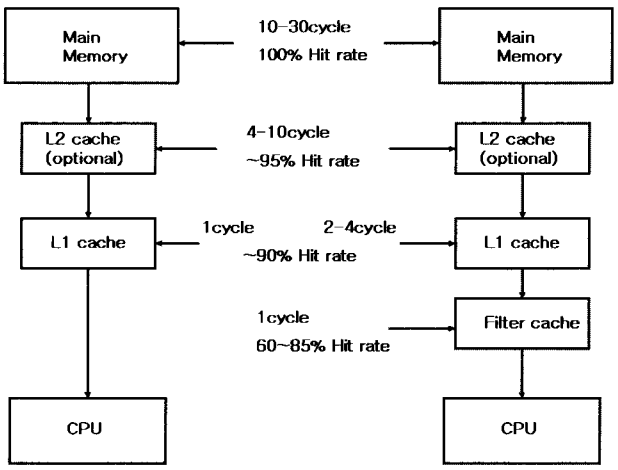

그림 1. 필터 캐시의 구조
Fig. 1. Filter Cache Architecture.

표 2. 256B 필터 캐시의 에너지 · 시간지연
Table 2. The Energy · Delay of 256B Filter Cache.

응용프로그램	전력소모 증가율	시간지연 증가율	에너지 · 시간 지연
cjpeg	0.4355	1.2845	0.5594
decode	0.3527	1.2703	0.4480
djpeg	0.5075	1.3940	0.7074
encode	0.3481	1.2827	0.4465
epic	0.3839	1.1360	0.4361
gs	0.4511	1.2953	0.5843
gsmdecode	0.3102	1.0327	0.3204
gsmencode	0.5263	1.3488	0.7098
mipmap	0.5237	1.2057	0.6314
mpeg2dec	0.5289	1.2501	0.6612
mpeg2enc	0.3865	1.3851	0.5353
osdemo	0.3770	1.0985	0.4141
pgpdecode	0.3644	1.1266	0.4105
pgpencode	0.3622	1.1335	0.4106
rasta	0.4630	1.1798	0.5462
rawcaudio	0.3368	1.2211	0.4113
rawdaudio	0.3396	1.1013	0.3740
texgen	0.4884	1.2197	0.5957
unepic	0.4617	1.0914	0.5039
Mean	0.4183	1.2135	0.5108

가 저하된다. 그러므로 필터 캐시의 적중률이 높을수록 전반적으로 전력 소모량이 감소하고, 실패율이 높을수록 실행 속도가 저하되는 결과를 나타내게 된다. 필터 캐시는 일반적으로 기존의 캐시 시스템과 비교하여 전력 소모량은 감소하지만, 실행 속도가 떨어지기 때문에 필터 캐시의 성능을 평가하기 위해서는 전력 소모량과 시간 지연을 모두 고려하는 전력 소모량과 시간 지연의 곱(Energy · Delay)을 구해야 한다. 필터 캐시를 IMPACT[5]와 MediaBench[6]로 활용하여 실험해 보면 표 2와 같은 결과가 나타난다. 라인의 크기가 32 비트이고, 직접 사상 캐시를 가지고 있는 기존의 시스템에 256 Byte의 필터 캐시를 추가하고 실험하였으며, 결과값은 필터 캐시를 사용하지 않았을 경우와 비교한 정규화된(Normalized) 상대적인 결과 값이다.

결과값을 살펴보면, 전력 소모량은 50% 이상 줄어들었으나, 실행 속도는 20% 이상 저하되었음을 알 수 있다. 그러나 전체적인 전력 소모량 · 시간 지연은 줄어들었기 때문에 성능이 향상되는 의미 있는 결과 값이다. 필터 캐시를 사용할 때 전력 소모량이 크게 감소하는 이유는 프로그램에서 사용했던 명령어들이 시간적 지역성(temporal locality) 때문에 필터 캐시에 남아있는 경우, 해당 명령어들을 재사용할 수 있기 때문이다. 원래는 레벨 1 캐시에서 읽어야 할 명령어를 크기가 작은 필터 캐시에서 읽기 때문에 레벨 1 캐시 접근에 소모되는 전력에서 필터 캐시 접근에 소모되는 전력을 뺀 만큼 전력 소모량을 감소시킬 수 있게 된다. 한편, 필터 캐시의 사용으로 인한 캐시 접근 시간이 증가하는 원인은 프로세서에서 캐시 시스템에 접근 시에 필터 캐시만을 접근하고, 그 적중 여부에 따라 재차 레벨 1 캐시로 접근하기 때문이다. 즉, 필터 캐시의 적중률이 낮을 경우는 그만큼 시간 지연이 발생하고, 레벨 1 캐시에 자주 접근해야 되기 때문에 전력 감소의 효과도 크지 않게 된다.

2. 필터 캐시 예측기

언급한 바와 같이, 필터 캐시 시스템은 전력 소모량의 감소 효과는 크지만, 전체적인 실행 속도의 저하가 두드러진다. 이를 극복하기 위해서, 필터 캐시의 연구 분야에서 그림 2와 같은 필터 캐시 예측기 모델(Filter Cache Predictor Model)이 등장하기 시작한다. 일반적인 필터 캐시에서는 프로세서에서 명령어를 인출할 때 필터 캐시에만 접근하고, 적중하지 못했을 경우에만 레벨 1 캐시에 접근하게 된다. 그러나 위의 구조에서는 예측기(predictor)가 프로세서와 필터 캐시 및 레벨 1 캐시 사이에 존재한다. 그래서 예측기가 현재 프로세서에서 접근하고자 하는 주소의 명령어가 필터 캐시에 있는지 없는지 여부를 먼저 예측해 보게 된다. 만약, 예측기가 현재 프로세서에서 접근하고자 하는 주소의 명령어가 필터 캐시에 있음이 확실하다고 판단되는 경우에는 필터 캐시에만 접근하게 된다. 반대로, 예측기가 현

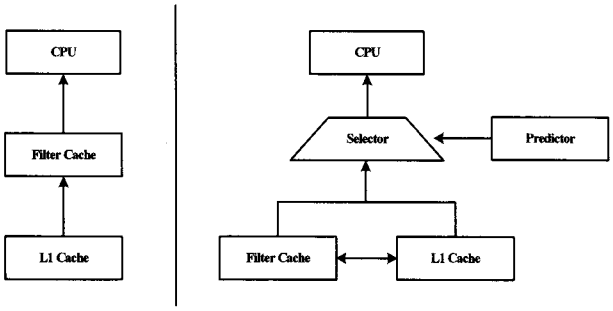

그림 2. 필터 캐시 예측기 시스템
Fig. 2. Filter Cache Predictor System.

재 프로세서에서 접근하고자 하는 주소의 명령어가 필터 캐시에 없는 것이 확실하다고 판단되는 경우에는 레벨 1 캐시로 바로 접근하여 시간 지연을 줄이게 된다. 예측기 모델은 다음의 4가지 시나리오를 따른다.

① 예측기가 다음에 실행할 명령어가 필터 캐시에 있다고 예측하였으며 그 예측이 적중했을 경우, 기존의 필터 캐시 적중 상황과 동일하며, 실행 속도와 전력 소모량의 이점은 없다.

② 예측기가 다음에 실행할 명령어가 필터 캐시에 있다고 예측하였으나 그 예측이 실패했을 경우, 기존의 필터 캐시 실패 상황과 동일하며, 실행 속도와 전력 소모량의 손해는 없다.

③ 예측기가 다음에 실행할 명령어가 필터 캐시에 없다고 예측하였으며 그 예측이 적중했을 경우, 기존의 필터 캐시보다 실행 속도가 향상되고, 전력 소모량이 감소한다.

④ 예측기가 다음에 실행할 명령어가 필터 캐시에 없다고 예측하였으나 그 예측이 잘못된 예측이었을 경우, 기존의 필터 캐시보다 실행 속도는 동일하나, 전력 소모량이 증가한다.

예측기를 사용함으로써 성능의 향상을 기대할 수 있는 것은 3번의 경우가 빈번할 때이다. 만약, 4번의 경우가 자주 나온다면, 기존의 필터 캐시보다 실행 속도는 동일하나 전력 소모량이 증가하여 결과적으로 성능이 저하되는 결과를 가져올 수 있다. 본 논문에서는 예측기 모델을 사용하는 필터 캐시 논문들에서 자주 비교의 대상이 되는 NFP(Next Fetch address Prediction table) 기법과 패턴 예측(PP, Pattern Prediction) 기법에 대해서 알아보고, 본 논문에서 제안하는 기법과 비교 평가해 본다.

가. NFP 기법의 소개

응용 프로그램을 실행하여 관찰하다 보면 짧은 구간을 반복해서 실행하는 짧은 루프(small loop)가 자주 나타난다. 이 짧은 루프가 실행되는 동안에 프로세서는 같은 구간의 명령어를 계속적으로 명령어 캐시에서 인출하게 된다. 루프 캐시(Loop Cache)기법은, 컴파일러를 사용하여 응용프로그램에 존재하는 짧은 구간의 루프를 찾아내고 이것을 미리 작은 캐시에 넣어두어 짧은 루프를 사용하는 구간에서 활용하는 기법이다[7]. 표 3은 이 기법을 사용하였을 경우, 전체 프로그램의 실행 중에서 루프 캐시에 접근하는 비율을 나타낸 것이다. 결과를 분석해 보면, 응용프로그램의 실행에서 짧은 루프의 실행 비율이 상당히 높다는 것을 알 수 있다.

표 3. 루프 캐시에 접근하는 비율
Table 3. The Access Ratio of Loop Cache.

응용 프로그램	루프 캐시의 크기		
	128 instr.	256 instr.	1024 instr.
hydro2d	93.25%	94.48%	94.49%
su2cor	92.55%	98.17%	99.22%
swim	99.88%	99.93%	99.93%
tomcatv	43.01%	99.84%	99.94%
turb3d	74.44%	74.87%	77.90%
wave	70.35%	94.11%	96.16%
compress	15.30%	15.30%	15.30%
go	1.36%	1.36%	1.36%
applu	66.87%	73.01%	73.35%
apsi	79.65%	83.84%	86.76%

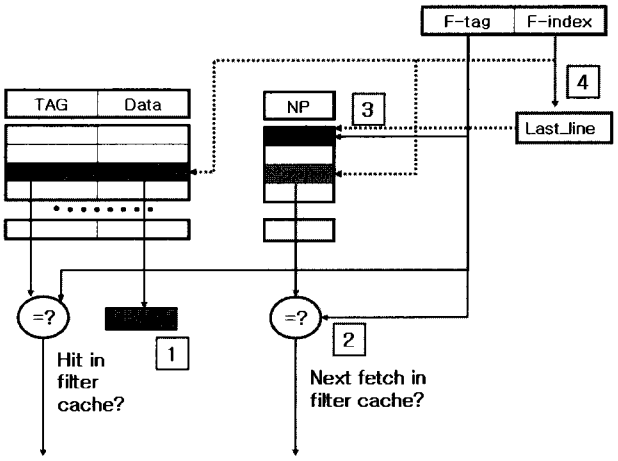

그림 3. NFP 구조
Fig. 3. NFP Architecture.

그림 3에 나타나 있는 NFP(Next Fetch address Prediction table)[4, 8] 기법은 이 점을 착안하여, 다음 명령어를 예측하는 테이블을 만들어서 참조하는 방식이다. 프로그램의 실행 흐름이 짧은 루프로 들어가게 되는 경우, 연속적으로 실행되는 명령어들 간의 주소 차이는 작다. 그렇게 되면, 현재 인출되는 명령어와 다음에 인출될 명령어들 사이의 태그(tag)간의 차이도 매우 작을 것으로 예상할 수 있다. 이를 활용하여, 이전에 실행된 명령어의 태그 값을 저장하고, 현재 실행된 명령어의 태그 값을 이와 비교한 후, 다음 명령어의 위치를 예측하게 된다. 이처럼 짧은 루프에서의 시간적인 지역성을 재활용하는 것이 NFP의 핵심이다.

NFP의 실험 결과를 통해 문제점을 살펴보면 다음과 같다. 필터 캐시보다 평균적으로 시간 지연은 12% 정도, 전력소모량은 5% 정도 감소하였다는 것을 알 수 있다[8]. 그러나 각 벤치마크별 필터 캐시의 적중률을 분석한 표 4를 참고해 보면, 원래 필터 캐시의 적중률이 70% 이상인 compress나 ijpeg, turb3d 벤치마크는 시간 지연은 거의 차이가 없으면서 전력 소모량만 늘어나게

표 4. NFP 기법의 성능
Table 4. The Performance of NFP Policy.

응용 프로그램	필터 캐시 적중률	시간지연 감소율	전력소모 감소율
compress	0.891	0.028	-0.06
gcc	0.392	0.119	0
go	0.446	0.081	-0.04
ijpeg	0.799	-0.008	-0.195
li	0.417	0.103	-0.062
perl	0.188	0.206	0.17
applu	0.572	0.1	0.023
apsi	0.538	0.04	-0.05
fpppp	0.026	0.178	0.233
hydro2d	0.347	0.165	0.069
su2cor	0.452	0.137	0.039
swim	0.217	0.498	0.331
tomcatv	0.314	0.145	0.059
turb3d	0.770	0.009	-0.128
wave	0.202	0.271	0.199

되는 것을 볼 수 있다. 이렇게 적중률이 높은 벤치마크에서 전력 소모량이 증가하는 것은 NFP 기법에서 예측기가 예측할 때, 필터 캐시에 해당 라인이 있음에도 불구하고, 레벨 1 캐시에서 읽어오는 중복읽기가 자주 발생하기 때문이다. 원래 필터 캐시 적중률이 높다는 것은 평균적으로 필터 캐시에 해당 내용이 들어있을 가능성이 크므로, 예측 기법이 정교하지 않았을 때, 예측기가 실행 속도만을 고려해서 무분별하게 레벨 1 캐시에서 명령어를 인출하게 하는 경우에 위와 같은 중복읽기가 많이 발생하는 것이다. 그런데, 내장형 시스템에서 쓰이는 프로그램들이나 멀티미디어 프로그램들의 경우 필터 캐시 적중률이 80% 이상이다. 그러므로 실제로 NFP는 내장형 시스템이나 멀티미디어용 프로세서에서는 채택되기가 힘들다. 레벨 1 캐시는 전력 소모량이 큰 캐시이므로, 필요 없는 레벨 1 접근은 전력의 낭비를 가져오게 되고, 잦은 중복읽기는 필터 캐시를 사용하지 않는 것보다 더 나쁜 성능을 가져오는 결과를 낳는다.

나. 패턴 예측 기법의 소개

명령어 수준의 병렬처리(ILP, Instruction Level Parallelism)와 파이프라인(pipleline)은 명령어의 실행 속도를 빠르게 하기 위해서 사용되어 왔다. 그런데, 명령어의 실행 중간 중간에 있는 분기 명령어들은 명령어의 진행 방향을 바꾸기 때문에, 효율적으로 파이프라인을 사용하는데 방해가 된다. 이에 따라서 프로세서의 실행 속도를 향상시키기 위해서 다양한 기법의 분기 예측기들을 사용하게 되었다. 이 중에서 가장 대표적인 예측기 가운데 하나가 2단계 적응형 분기 예측기(2 level adaptive branch predictor)[9]이다. 패턴 예측 기법(Pattern Prediction)[10]에서는 이 예측기를 필터 캐시의 예측기로 사용하는 것이다. 패턴 예측 기법은 기존의 예측기 모델에 2단계 적응형 분기 예측기에서 사용하는 하드웨어들을 추가하였다. 이 기법의 구조는 캐시 라인 접근의 적중/실패 패턴을 기록하기 위한 쉬프트 레지스터(SR)와 2-bit 포화 카운터(saturation counter)로 구성된 PHT(Pattern History Table)와 last_PC 저장용 레지스터 및 비교회로로 구성된다.

동작 방식을 살펴보면, 먼저 프로세서는 이전에 예측기에서 결정되어 있는 예측대로 명령어를 인출해 온다. 그리고 현재 실행되는 명령어의 주소 값과 이전에 실행된 명령어의 주소가 저장되어 있는 last_PC 레지스터의 값을 비교해서 서로 다른 라인인지 같은 라인인지를 판단해 본다. 다른 라인인 경우는 SR과 PHT를 갱신한다. 먼저, 이전에 필터 캐시의 적중/실패의 여부를 SR에 갱신한다. SR를 좌측으로 1 비트 쉬프트하고, 적중/실패 여부를 최상위비트로 입력한다. 그리고 SR로 찾은 PHT의 포화 카운터 값을 갱신한다. last_PC 레지스터의 값은 현재의 주소로 갱신한다. 같은 라인인 경우는 last_PC 레지스터의 값을 현재의 주소로 갱신하는 작업만을 수행한다.

위의 예측기는 현재 명령어가 라인의 마지막 명령어인지를 판단한다. 라인의 마지막 명령어인 경우는 SR를 이용해서 해당 PHT로 접근하여 다음 명령어를 어디서 인출해 올 것인지를 결정한다. PHT의 포화 카운터의 값이 1보다 큰 경우는 필터 캐시에서 인출해 오도록 하고, 1이하인 경우는 레벨 1 캐시에서 인출해 오도록 한다. 같은 라인인 경우는 무조건 필터 캐시에서 인출해 오도록 한다.

패턴 예측 기법의 전력 소모량과 시간 지연의 곱(Energy · Delay)의 결과값은 NFP의 결과값과 크게 다르지 않다. 필터 캐시의 크기가 256B일 때와 512B일 때 각각 3.39%, 2.39%씩 줄어든다. 그러나 이 기법은 필터 캐시의 크기를 변화시키지 않고 별도의 설정을 통해서 예측기의 성능을 조정할 수 있으며, 다른 분기 예측 기법을 확대 적용할 수 있는 유연성이 있으므로 발전 가능성이 크다.

Ⅲ. 모드 선택 비트를 사용한 필터 캐시 예측기

앞 장에서 살펴본 바와 같이, 필터 캐시를 사용하는 시스템의 성능은 필터 캐시에서 어떤 알고리즘을 채택하는가에 크게 의존한다. 그러므로 효율적인 알고리즘 작성을 위해서는 예측기 모델에서 예측기의 정확한 역할에 대한 분석이 요구된다.

1. 예측기 모델 분석

내장형 시스템의 프로세서는 일반적인 마이크로프로세서와는 달리 한 번에 처리할 수 있는 명령어의 개수가 적고, 대부분 순차적인 실행(in-order)을 채택하고 있다. 즉, 한 사이클에 프로세서가 처리할 수 있는 명령어는 1개인 경우가 많다는 것이다. 내장형 시스템에서는 전력 소모량을 줄어야 하며, 칩의 크기도 작게 만들어야 하는 제약이 있기 때문이다. 이러한 제약 때문에 내장형 시스템은 파이프라인도 길지 않고, 사이클도 긴 편이다. 반면, 내장형 시스템에서의 캐시 시스템은 일반적인 마이크로프로세서와 동일하다. 캐시 시스템에서 상위 레벨로의 입출력은 몇 개의 명령어를 포함하는 라인단위로 이루어진다. 캐시 시스템이 라인 단위로 입출력을 수행한다는 것은 일정한 특징을 지니게 되는데, 이것은 예측기의 예측에 중요한 단서가 될 수 있다.

분석을 위해서 프로그램이 캐시에 저장되어 있는 상황을 생각해 보기로 하자. 본 논문에서는 작고 빠른 캐시의 동작을 보장하기 위해 직접 사상 캐시를 고려하여 설명한다. 우선, 프로그램이 실행되면, 중간에 분기 명령어가 없는 한, 실행된 명령어들은 필터 캐시에 그림 4와 같이 순차적으로 저장되게 된다. 프로그램의 흐름이 다시 처음 라인의 2번째 명령어로 돌아오게 되면, 현재 명령어 뒤에 있는 같은 라인의 명령어 2개는 다음번 그리고 그 다음번에 실행되는 것을 알 수 있다. 그러므로 예측기는 다음번과 그 다음번에 명령어를 필터 캐시에서 인출해 오도록 예측할 수 있다. 이를 일반화 해 보면, 분기 명령어를 고려하지 않았을 경우, 필터 캐시에서 특정 라인의 명령어가 실행된다면, 라인이 끝나지 않는 한 다음 명령어의 위치는 같은 라인에 위치하게 되고, 다음 명령어는 필터 캐시에 있다는 것을 알 수 있다. 만약, 현재 인출되는 명령어가 필터 캐시에 없었다고 하더라도 명령어가 들어있는 라인을 레벨 1 캐시에서 가져와서 필터 캐시에 저장해야 하므로, 다음 명령어는 반드시 필터 캐시에 들어가 있게 된다. 여기서 예측기 모델의 효율적인 동작 방식 하나를 찾을 수 있다.

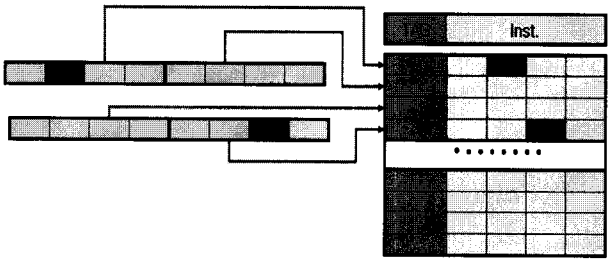

그림 4. 명령어의 흐름과 캐시의 구조
Fig. 4. Instruction Stream and Cache Structure.

• 현재 명령어의 위치가 라인의 마지막이 아니고, 조건 분기 명령어도 아닌 경우라면, 예측기는 다음 명령어를 필터 캐시에서 인출하도록 프로세서에 지시한다.

프로그램이 진행되어, 현재 인출되는 명령어가 해당 라인의 마지막에 위치할 경우를 가정하자. 분기 명령어가 아니라면 다음에 인출될 명령어는 다음 라인의 처음에 위치하게 된다. 그러나 다음 라인이 필터 캐시에 존재하는지 여부를 알 수 없으므로, 예측기는 어디서 인출할 것인지 판단할 수 없다. 일반적으로 멀티미디어와 통신 프로그램의 경우 짧은 루프를 가지는 경우가 많으므로, 필터 캐시에서 읽어오게 하는 쪽이 좀 더 높은 적중률을 보일 것으로 예상되지만, 정교한 예측은 불가능하다. 해당 기본 블록(basic block)이 최초에 필터 캐시에 들어가는 경우에는 연속적으로 인출에 실패할 것이기 때문이다. 이를 위해서, 필터 캐시 옆에 1 비트짜리 테이블을 추가하여 현재 라인이 필터 캐시에 최초로 저장된 라인인지 이전에 라인 끝까지 실행된 적이 있는 라인인지를 저장하도록 한다면 더 좋은 성능을 기대할 수 있을 것으로 예상된다.

• 현재 명령어의 위치가 라인의 마지막이고, 분기 명령어가 아닌 경우라면, 예측기는 다음 라인이 필터 캐시에 존재하는지를 최초 실행 여부를 활용하여 판단한다.

마지막으로 예측기가 고려해야 될 사항은 해당 명령어가 분기 명령어일 경우이다. 분기 명령어의 비율을 알아보기 위해서 내장형 프로세서를 위한 벤치마크인 MiBench[11]로 실험을 하여 그림 5와 같은 결과를 얻었다. 결과를 살펴보면 평균적으로 전체의 10% 이상의 명령어가 분기 명령어임을 알 수 있다. 또한 이같은 분

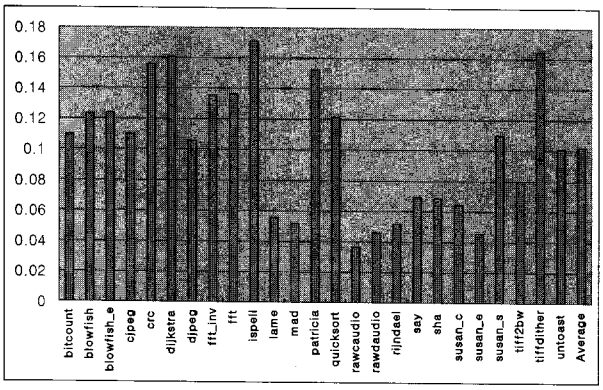

그림 5. 내장형 프로그램에서 분기의 비율
Fig. 5. The Portion of Branch Instructions in Embedded Applications.

기 명령어에 의해 캐시의 시공간적 지역성은 더욱 강화될 수 있다[12]. 따라서 분기 명령어를 무시하게 되면, 적지 않은 성능의 저하가 예상된다. 그러므로 예측기는 분기 명령어가 나타나게 되면 다음 라인을 예측해야 하는데 앞의 경우와는 다른 좀 더 복잡한 예측 기법이 필요하게 된다. 분기 명령어는 오래전부터 명령어 수준의 병렬처리를 저해하는 요인으로 정확한 분기 예측을 위한 많은 예측 기법들이 연구되어 왔다. 고성능 컴퓨터에 쓰이는 복잡하고 정교한 분기 예측 기법에서 부터 내장형 시스템을 위한 간단하고 효율적인 분기 예측 기법들이 존재하는데, 이러한 기법들을 필터 캐시 예측기에 적용하게 되면 예측율을 높일 수 있을 것이다.

• 현재 인출되는 명령어가 분기 명령어이면, 분기 예측 기법을 적용하여 다음 라인의 위치를 찾아낸다.

앞의 3가지 경우를 고려할 때, 실제적으로 예측기는 다음의 2가지 경우에 필요하며, 각각의 경우에 최선의 정책을 사용하게 되면 가장 좋은 성능을 보일 수 있음을 알 수 있다.

• 현재 인출되는 명령어가 라인의 마지막 명령어인 경우
• 현재 인출되는 명령어가 분기 명령어인 경우

2. 모드 선택 비트를 사용한 필터 캐시 예측기

본 논문에서 제안하는 모드 선택 비트를 사용한 필터 캐시 예측기 모델(Filter Cache Predictor Model using Mode Selection Bit)은 위의 예측기 모델의 분석에서 나온 2가지 경우를 고려해서 제안되었다. 그림 6에서 보는 바와 같이, 먼저 현재 인출되는 명령어의 주소와

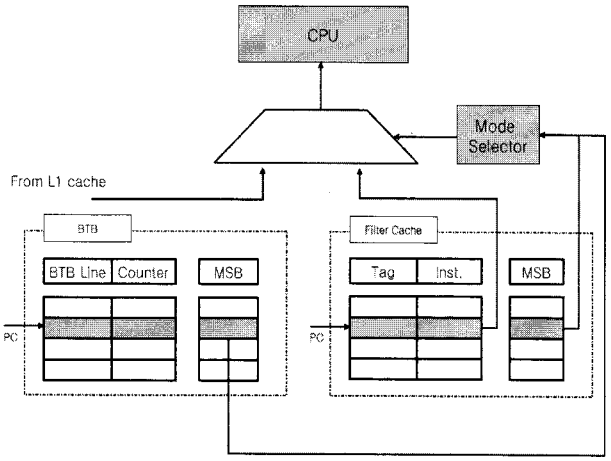

그림 6. 모드 선택 비트를 사용한 필터 캐시 예측기
Fig. 6. Filter Cache Predictor using MSB.

다음 주소의 라인과 같은지를 비교해서, 라인의 마지막 명령어인지를 판단하게 된다. 라인의 마지막 명령어인 경우, 모드 선택 비트 테이블의 해당 값을 참조하여 모드 선택기(Mode Selector)가 다음에 인출할 명령어를 어떤 캐시에서 읽어오는지를 판단하게 된다. 명령어의 주소가 BTB에서 읽히는 경우, 현재 명령어가 분기 명령어라고 판단하고, BTB(Branch Target Buffer)의 모드 비트를 보고 모드 선택기가 모드를 선택하게 한다. 모드 선택 비트를 사용한 필터 캐시 예측기 모델은 다음과 같이 구성된다.

• 모드 선택기(Mode Selector) : 제안된 모델은 필터 캐시 모드와 레벨 1 캐시 모드 중 하나의 모드로 동작하게 된다. 필터 캐시 모드는 다음 명령어를 필터 캐시에서 인출하게 되고, 레벨 1 모드는 다음 명령어를 레벨 1 명령어 캐시에서 인출하게 된다.

• 필터 캐시의 모드 선택 비트 테이블(MSB Table) : 필터 캐시의 엔트리 수와 동일한 수의 엔트리를 가지는 1비트짜리 테이블이 필터 캐시 라인에 추가된다. 모드 선택 비트는 필터 캐시에 라인이 저장될 때 리셋(0)되며, 한 번 라인의 마지막까지 실행이 되면 비트가 세팅(1)된다. 라인의 마지막 명령어에 접근할 때 해당 엔트리의 비트를 참조하여 모드 선택기가 모드를 선택한다.

• BTB의 모드 선택 비트 테이블과 카운터(MSB Table and Counter) : 인출되는 명령어가 분기 명령어일 때는 BTB의 모드 선택 비트를 참조하여 모드를 선택하게 된다. BTB의 모드 선택 비트는 함께 붙어있는 카운터에 의해 관리된다. BTB의 분기 명령어가 적중하게 되면 카운터 값이 1씩 증가하여 카운터 값이 임계값(threshold)에 이르게 되면 라인의 모

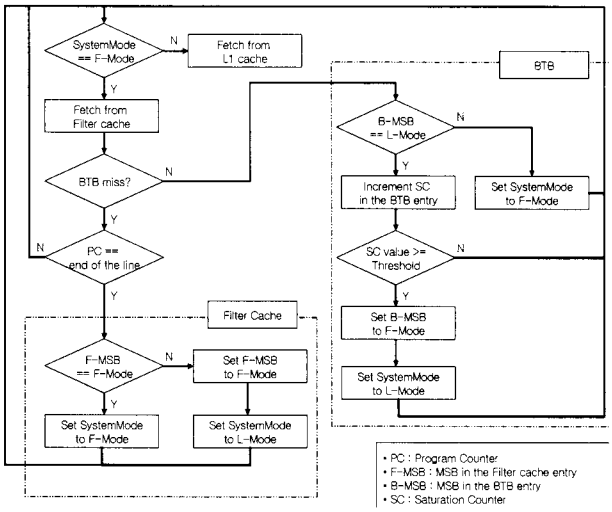

그림 7. 필터 캐시 예측기의 동작 알고리즘
Fig. 7. The Algorithm for Filter Cache Predictor.

드 비트는 세팅된다. BTB가 새로운 분기 명령어와 타겟을 라인에 저장하게 되면 모드 선택 비트는 리셋 된다.

제안된 모델의 이해를 돕기 위해서 그림 7에 동작 알고리즘이 나타나 있다. 먼저, 프로그램이 처음 시작하고 필터 캐시가 초기화 되면, 모드 비트는 모두 리셋 된다. 프로그램이 시작될 때는 필터 캐시가 비어있으므로 레벨 1 캐시 모드(L-Mode라 칭함)로 동작한다. 프로그램이 진행되다가 현재 인출되는 명령어가 해당 라인의 마지막 명령어라고 판단되는 경우에는 필터 캐시의 모드 비트를 참조하게 된다. 이때, 모드 비트가 리셋되어 있으면, 이 라인은 처음으로 필터 캐시에 들어온 라인이므로, 다음 라인도 역시 필터 캐시에 없을 가능성이 크다. 따라서 레벨 1 캐시 모드로 동작하도록 한 후, 해당 라인은 마지막 명령어까지 실행되었으므로 모드 비트를 세팅시킨다. 반면, 라인의 마지막 명령어이면서 모드 비

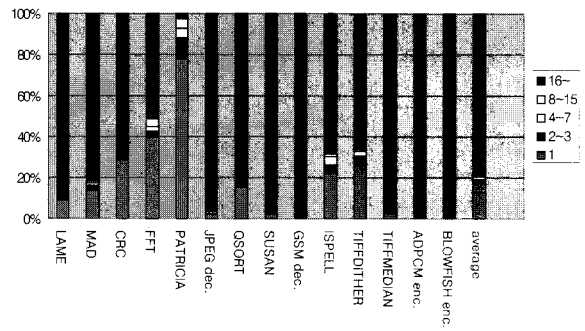

그림 8. 분기 명령어가 BTB에서 참조되는 횟수
Fig. 8. The Number of Branch References in BTB.

트가 세팅되어 있으면, 이 라인은 이전에 이미 한 번 실행되었던 라인이며 다음 라인도 역시 필터 캐시에 있을 가능성이 크므로, 모드 선택기가 다음 명령어를 필터 캐시에서 인출하도록 필터 캐시 모드(F-Mode라 칭함)로 모드를 변환하게 한다. 해당 라인에 새로운 캐시 라인이 들어올 경우 모드 비트는 다시 리셋된다.

분기 명령어로 인한 필터 캐시의 접근 실패율을 줄이기 위해서 본 논문에서는 추가적으로 Filter 분기 예측 기법[13]을 도입한다. 이 기법은 BTB의 적중 빈도를 사용하여 분기 명령어를 예측하는 기법이다. 우선, 기존의 BTB에 BTB의 엔트리의 개수만큼 포화 카운터와 모드 선택 비트를 추가한다. BTB에 새로운 분기 명령어 주소와 타켓 주소가 들어올 때마다 포화 카운터와 모드 선택 비트는 리셋된다. 프로그램의 실행 중에 BTB가 적중하게 되면 모드 선택기로 하여금 해당 라인의 모드 비트를 통해서 다음 명령어의 모드를 결정하게 된다. 동시에 적중한 BTB라인의 포화 카운터의 값을 증가시킨다. 그래서 일정한 수준의 임계값(threshold)에 이르게 되면 모드 선택 비트를 세팅한다.

임계값을 설정하는 이유는 필터 캐시의 접근 실패율을 줄이기 위해서이다. 자주 쓰이지 않는 분기 명령어의 경우, 모드 선택 비트를 세팅해 둘 경우 필터 캐시의 실패를 유발할 수 있으므로, 일정 임계값 기준 이하인 경우는 모드 선택 비트를 세팅하지 않고 좀 더 관찰하며, 임계값이 넘어가게 되면 모드 선택 비트를 세팅한다. 본 논문에서는 임계값의 결정을 위해서 분기 명령어가 BTB에 머무르는 횟수를 측정하는 실험을 실시하였다. 그림 8을 보면, PATRICIA 프로그램은 예외적으로 분기 명령어가 BTB에 4~7번 머무는 비율과 8~15번 머무는 비율이 10% 이상이지만, 대개의 경우는 2~3번, 4~7번, 8~15번의 경우는 그 비중이 미미함을 알 수 있다. 즉, BTB에서 2~3번 이상 쓰이는 명령어들은 결국은 16번 이상 쓰이게 되는 명령어가 될 확률이 높다. 그러므로 BTB의 포화 카운터의 크기는 2 비트로 결정하여 사용한다. 2 비트의 포화 카운터를 포화시키는 분기 명령어는 16번 이상 쓰이는 분기 명령어가 될 확률이 평균적으로 79% 정도에 이른다.

IV. 모의실험 및 성능분석

1. 모의실험 환경 및 벤치마크 프로그램

내장형 프로세서를 대상으로 하는 본 논문의 검증을

위해서, 슈퍼스칼라 프로세서의 사이클 수준 시뮬레이터인 SimpleScalar ARM 버전을 사용하여 모의실험을 실시하였다[14]. SimpleScalar ARM 버전은 기존의 SimpleScalar 시뮬레이터에 ARM 바이너리로 컴파일된 프로그램을 실행하기 위해 ARM ISA(Instruction Set Architecture)의 지원이 가능하도록 개조되었다. 또한, 전력소모량의 측정을 위해서 CACTI 4.0을 사용하였으며, 0.18um 기술을 가정하였다[15].

모의실험을 위한 환경 인자는 표 5와 같다. 정확한 실험을 위해서 기존에 쓰이고 있는 내장형 프로세서와 가장 근접한 형태로 만들었다. 기존의 내장형 프로세서들과 같이 명령어의 실행은 순차적(in-order) 실행을 선택했으며, 레벨 2 캐시는 삭제하였다. 필터 캐시의 크기를 256B, 512B, 1024B로 변화시켜 가면서 모의실험을 수행하였다.

한편, 일반적으로 모의실험에 많이 쓰이는 벤치마크 프로그램은 SPEC(Standard Performance Evaluation Corporation) 프로세서 벤치마크이다[16]. 이는 범용 프로세서 디자인의 실험을 위해서 많이 사용되는데 특히 서버와 하이엔드 유저용 시스템을 위한 실험에 많이 쓰인다. 이러한 시스템은 파이프라인이 깊고, 복잡한 분기명령어 예측기와 대용량 캐시의 사용 등을 특징으로 한다. 그러나 빠르게 성장하고 있는 내장형 프로세서 는 저렴한 가격과 전력소모의 감소를 위해서 단순한 구조의 작은 캐시를 가진다. 내장형 프로세서는 단순한 마이크로컨트롤러에서 핸드폰에 들어가는 무선통신기능을 내장한 프로세서에까지 다양한 용도로 쓰인다. 본 논문에서는 EEMBC[17]를 참고하여 만든 공개용 벤치마크인 MiBench[11]를 사용하였다. MiBench는 프로그램들을 산업용 제어 시스템(automotive and industrial control), 네트워크(network) 영역, 보안(security) 영역, 소비자 기기(consumer devices), 사무 자동화(office automation), 통신(telecommunications) 영역으로 구분한다. 본 논문에서는 가장 대표적인 내장형 시스템 응용 분야인 멀티미디어와 통신 영역 위주로 프로그램들을 선택하여 실험하였다.

표 5. 모의실험 환경
Table 5. Simulation Environments.

인자 종류		인자 값
필터 캐시	크기	256B , 512B , 1024B
	집합 연관	직접 사상
	라인 크기	32B
	접근 성공 지연	1 사이클
레벨 1 명령어 캐시	크기	32KB
	집합 연관	32-way 세트 연관사상
	라인 크기	32B
	접근 성공 지연	1 사이클
레벨 1 데이터 캐시	크기	32KB
	집합 연관	32-way 세트 연관사상
	라인 크기	32B
	접근 성공 지연	1 사이클
메모리 접근 성공 지연 사이클		32 사이클
정수 수치 연산기(IALU)		1
부동 소수점 수치 연산기(FPALU)		1
정수 곱셈/나눗셈 연산기(IMUL)		1
부동 소수점 곱셈/나눗셈 연산기		1
1 사이클 당 명령어 인출 속도		1
RUU(Register Update Unit) 크기		8
LSQ(Load Store Queue) 크기		8
분기 예측기의 종류		Bimodal
분기 예측기의 테이블 크기		128
분기 예측 실패 손실(branch penalty)		3
분기 타겟 버퍼(BTB)의 엔트리 수		128
분기 타겟 버퍼(BTB)의 집합 연관		직접 사상

2. 실험 결과 및 분석

모의실험은 Ⅱ장에서 소개한 NFP 기법(Next Fetch address Prediction)과 패턴 예측 기법(Pattern Prediction, 이하 PP라고 약칭함)기법, 그리고 3장에서 소개한 모드 선택 비트를 이용한 필터 캐시 기법(Filter Cache using Mode Selection bit, 이하 MS라고 약칭함)을 상호 비교 평가한다. 성능 평가의 기준을 위해, 인출할 명령어가 필터 캐시에 있을 경우는 필터 캐시에 접근하고 없을 경우는 레벨 1 캐시에 접근하는 100% 정확한 이상적인 예측기(OPtimal predictor, 이하 OP라고 약칭함)도 설정하여 함께 실험하였다. PP의 경우, SR 레지스터의 크기는 해당 논문[10]에서 가장 좋은 성능을 보였던 5bit를 사용했으며, PHT의 엔트리 수는 32개로 설정하였다. 필터 캐시의 크기는 결과의 순서대로 각각 256B, 512B, 1024B이다.

그림 9는 각 예측기의 예측 실패율을 나타낸 것이다. 예측 실패 비율이 높을수록 필터 캐시에서 레벨 1 캐시로 명령어가 포함된 라인을 요청하고 다시 저장하는 시간이 발생하여 시간지연이 발생하게 된다. 그림 9에서처럼, 256B 필터 캐시인 경우에는 NFP가 예측 실패율이 가장 높고, PP가 다음이며 MS가 가장 낮다. 또한, 필터 캐시의 크기가 증가할수록 PP가 NFP보다 예측

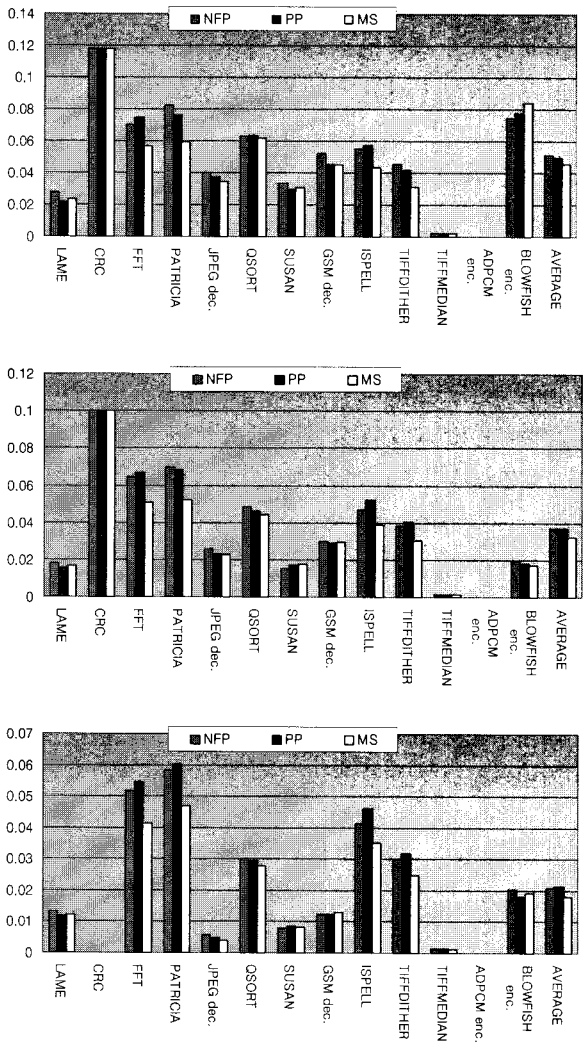

그림 9. 예측기의 예측 실패율
Fig. 9. Miss-Prediction Ratio.

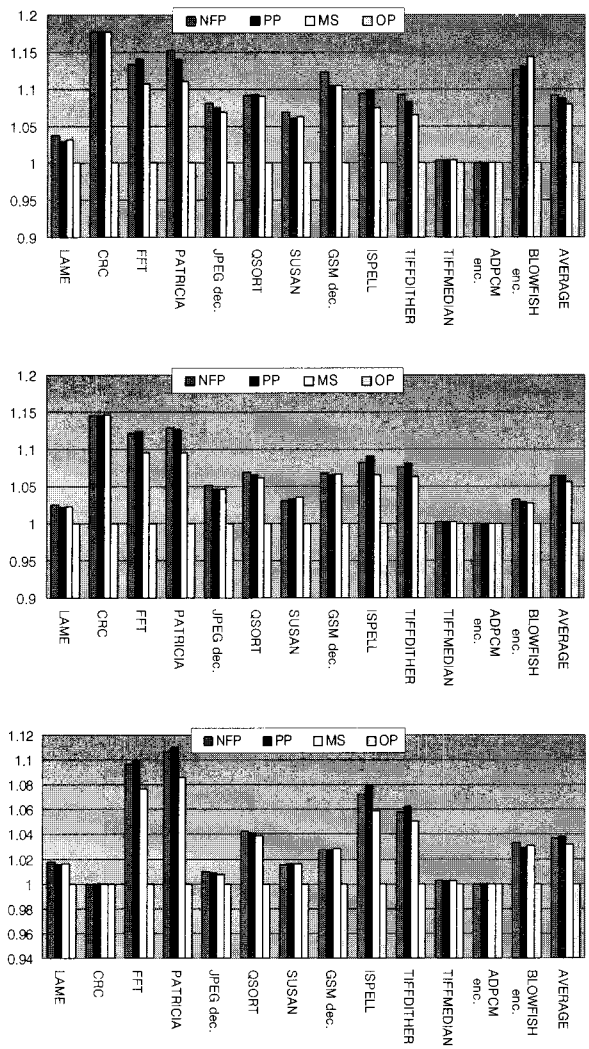

그림 10. 정규화된 지연 시간
Fig. 10. Normalized Delay.

실패율이 커지는 것을 볼 수 있다. NFP는 동작원리상 필터 캐시 엔트리의 수가 성능에 영향을 더 많이 미치기 때문에, 엔트리 수가 증가하면 예측 실패율이 비례하여 떨어지게 된다. 이에 반해, 필터 캐시의 엔트리 수가 성능에 영향을 덜 주는 PP는 예측 실패율이 떨어지는 비율이 낮아진다. 본 논문에서 제안된 기법인 MS는 모든 경우에 있어서 예측 실패율이 가장 낮다.

그림 10은 OP를 기준으로 하여 얻어진 정규화된 시간 지연(Normalized delay)을 표시한 것이다. 예측 실패율과 시간 지연은 거의 같은 경향을 보이고 있다. 평균적으로 256B 필터 캐시에서 각 예측기 시스템은 이상적인 상황보다 NFP는 9%, PP는 8.6%, MS는 7.5%의 추가적인 시간 지연을 보이며, 512B 필터 캐시에서는 6.4%, 6.3%, 5.6% , 1024B 필터 캐시에서는 3.7%, 3.9%, 3,1%의 추가적인 시간 지연을 보인다. 이것은 필터 캐시의 용량이 증가함에 따라 예측 실패율 자체가 낮아지며, 실패율의 차이 또한 작아지기 때문이다. 결과적으로 실험을 통해 알 수 있듯이, 본 논문에서 제안된 MS 기법은 낮은 예측 실패율로 인해, 다른 비교 대상보다, 더 적은 시간 지연을 가진다는 것을 확인 할 수 있다.

필터 캐시에서의 전력 소모 감소량은 필터 캐시 활용률에 많은 영향을 받는다. 필터 캐시 활용률은 프로그램의 실행 도중 인출된 전체 명령어의 개수 중에서 필터 캐시에서 인출된 명령어 수의 비율이다. 그림 11에 필터 캐시 활용률이 소개되어 있다. 필터 캐시의 활용률이 높다는 것은 프로그램 실행 중에 필터 캐시를 많이 사용하여, 레벨 1 명령어 캐시의 접근을 줄였다는 것을 의미한다. 따라서 필터 캐시의 활용률이 높을수록 전력 소모량이 감소하게 된다. 결과를 살펴보면, 필터 캐시의 용량이 증가할수록 PP의 활용률이 좀 더 많이

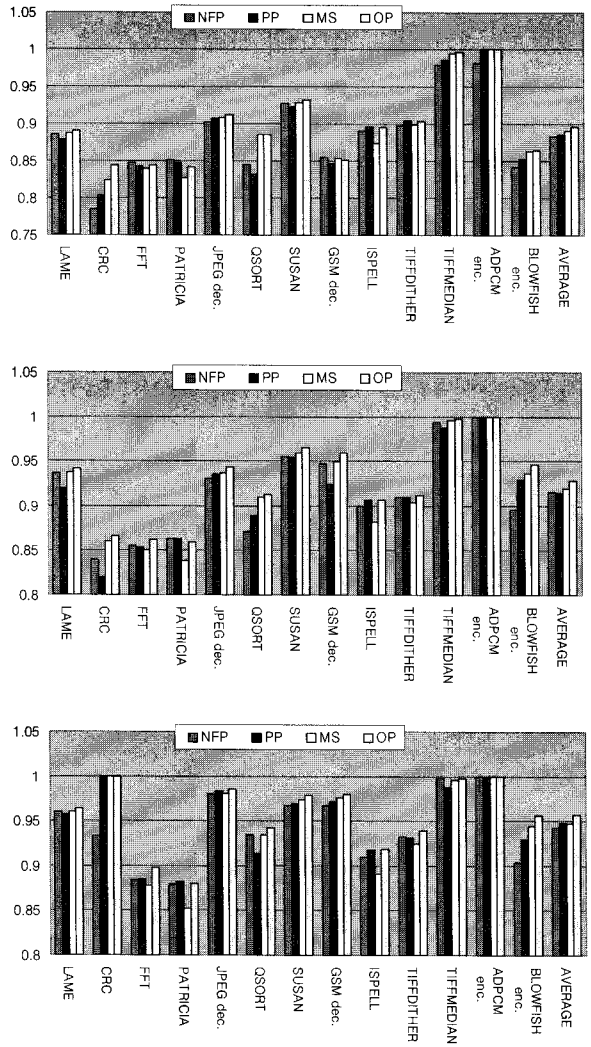

그림 11. 필터 캐시 활용률
Fig. 11. The Utilization Ratio of Filter Cache.

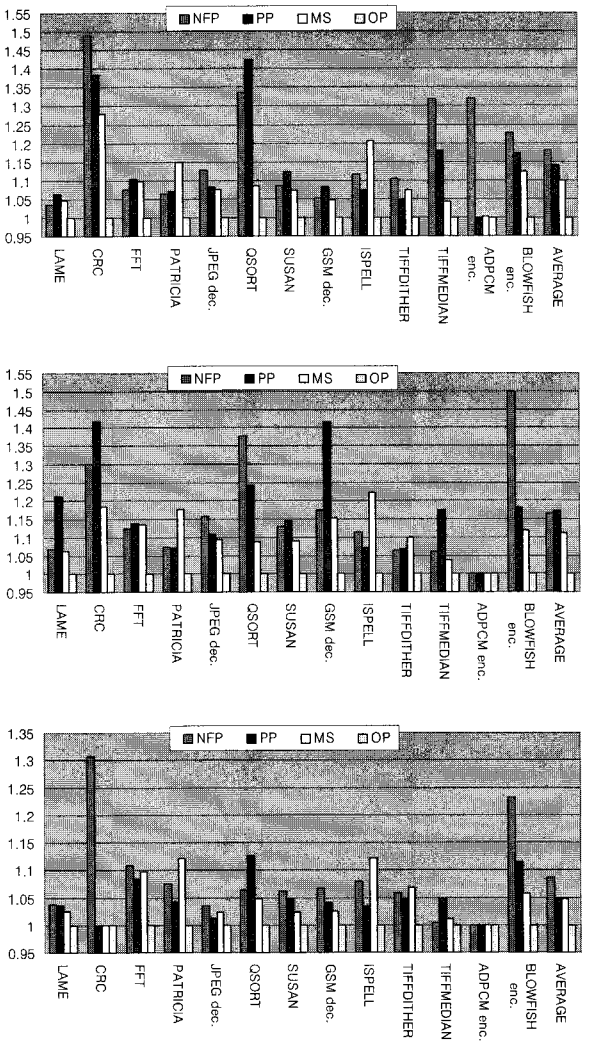

그림 12. 정규화된 전력 소모량
Fig. 12. Normalized Power Consumption.

높아지고 있다. CRC의 경우 특이한 결과를 보여주는데, 1024B 필터 캐시에서는 거의 1에 가까운 활용률을 보이고 있다. 이는 CRC 프로그램의 실행 흐름이 대부분 1024B 내에서 이루어지고 있음을 의미한다.

그림 12는 각 예측기 시스템의 전력 소모량을 OP를 기준으로 정규화하여 나타낸 것이다. 실험 결과를 살펴보면, 필터 캐시 활용률이 높을수록 전력 소모량이 줄어드는 것을 알 수 있다. 256B 필터 캐시에서 각 기법들은 OP 대비 NFP는 18%, PP는 14%, MS는 10%의 추가 전력 소모량이 발생하였으며, 512B 필터 캐시에서는 16%, 17%, 11%, 1024B 필터 캐시에서는 8.6%, 4.9%, 4.8%의 추가 전력 소모량이 발생하였다. 한편, NFP 기법 대비 평균 5.8%의 성능 향상을 가져온 [18] 기법과의 비교에 있어서도, 256B에서 NFP 기법은 OP 기법 대비 18%, MS 기법은 OP 기법 대비 10%의 추가 전력 소모가 발생하는 것을 볼 때, MS 기법이 [18]에서 제안된 기법보다도 상대적으로 우수하다는 것을 알 수 있다.

끝으로, 그림 13은 각 예측기의 에너지·시간 지연을 나타낸 것이다. 결과에 보이듯이, 본 논문에서 제안한 MS기법은 기존의 NFP나 PP보다 좋은 성능을 보이고 있다. 평균적으로 256B 필터 캐시일 때는 전력 소모량 ·시간 지연이 각각 9.1%, 5.2% 감소하였으며, 512B 필터 캐시일 때는 5.8%, 6.3%, 1024B 필터 캐시일 때는 4.1%, 0.6% 감소하였다. 본 논문에서 제안된 MS 기법은 NFP 기법에서 나타나는 특징인 필터 캐시의 용량이 증가함에 따라 꾸준하게 성능이 향상되는 경향성을 함께 유지하고 있다. 그러면서 동시에 분기 명령어를 고려한 예측 기법을 사용하고 있기 때문에, 패턴 예측 기법의 장점인 분기 명령어에서 쓰이는 기

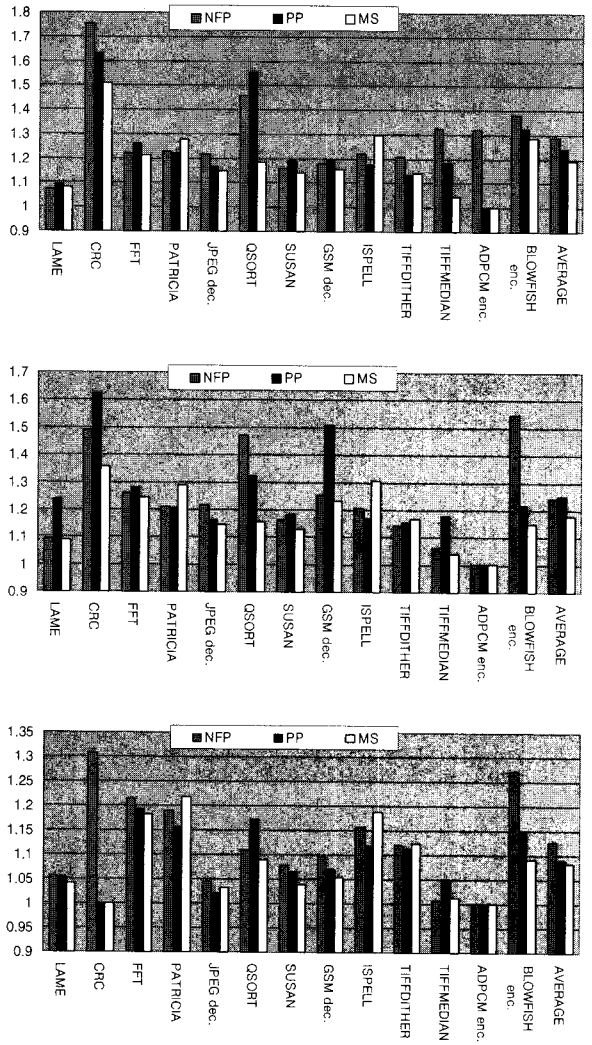

그림 13. 에너지 · 시간 지연
Fig. 13. Energy · Delay.

법들을 필터 캐시 시스템에 접목시킬 수 있는 유연성도 가지고 있다.

V. 결 론

필터 캐시는 캐시 시스템의 전력 소모량을 최소화하기 위해서 설계된 기법으로, 프로세서와 레벨 1 캐시 사이에 작은 캐시를 하나 추가하여, 레벨 0 캐시의 역할을 수행토록 한다. 이를 통해, 레벨 1 캐시의 접근을 줄임으로써 소비 전력을 줄이도록 하는 기법이다.

본 논문에서는 기존의 필터 캐시 시스템에서 자주 쓰이는 필터 캐시 예측기 시스템을 분석하여 성능 향상을 위한 원칙을 세우고, 이를 토대로 기존의 예측기보다 좀 더 정교한 예측이 가능한 "모드 선택 비트를 이용한 필터 캐시 예측 시스템"을 제안하였다. 제안된 새로운 시스템은, 필터 캐시에 모드 선택 비트를 추가하고, BTB에는 포화 카운터와 모드 선택 비트를 추가한다. 이를 활용하여, 현재 인출되는 명령어가 라인의 마지막 명령어이거나 분기 명령어인 경우, 해당 라인의 모드 선택 비트를 참조하여 다음 명령어의 위치를 예측한다.

모의실험을 통해서 검증한 결과 본 논문에서 제안된 기법은, 기존의 NFP 기법이나 패턴 예측 기법보다 좋은 성능을 보이고 있다. 평균적으로 256B 필터 캐시일 때는 전력 소모량 · 시간 지연이 각각 9.1%, 5.2% 감소하였으며, 512B 필터 캐시일 때는 5.8%, 6.3%, 그리고 1024B 필터 캐시일 때는 4.1%, 0.6% 감소하였다. 결과에서 보듯이 모드 선택 비트 기법은, NFP 기법에서 나타나는 필터 캐시의 용량이 증가함에 따라 꾸준하게 성능이 향상되는 경향성을 여전히 유지하고 있다. 그러면서 동시에 분기 명령어를 고려한 예측 기법의 장점을 함께 사용을 하고 있어서, 패턴 예측 기법의 장점인 분기 명령어에서 쓰이는 기법들을 필터 캐시 시스템에 접목시킬 수 있는 유연성도 가지고 있었다.

참 고 문 헌

[1] D. Patterson, and J. Hennessy, "Computer architecture: a quantitative approach" 4th edition, Morgan Kaufman, 2007.

[2] J. Montanaro et al. "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor", IEEE Journal of Solid-State Circuits, 32(11):1703-14, 1996.

[3] C.-L. Su and A. Despain. "Cache design tradeoffs for power and performance optimization: A case study", In Proceedings of International Symposium on Low Power Design, April 1995.

[4] W. Tang, A. Kejariwal, A. Veidenbaum and A. Nicolau, "A Predictive Decode Filter Cache for Reducing Power Consumption in Embedded Processors", ACM Transactions on Design Automation of Electronic Systems, Vol. 12, No. 2, April 2007.

[5] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J.Warter, and W.-m. W. Hwu, "IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors", Proc. of ISCA, 1991.

[6] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating

Multimedia and Communications Systems", Proc. of Micro 30, 1997.
[7] Anderson, T., and Agarwala, S.: "Effective hardware-based two-way loop cache for high-performance low-power processors", Proc. Int. Conf. on Computer Design, pp. 403 - 407, 2000.
[8] Tang, W., Gupta, R., and Nicolau, A.: "Design of a predictive filter cache for energy savings in high-performance processor architectures". Proc. Int. Conf. on Computer Design, pp. 68 - 73, 2001.
[9] Yeh, T.Y., and Patt, Y.N.: "Alternative implementation of two-level adaptive branch prediction", Proc. 19th Int. Symp. on Computer Architecture, pp. 124 - 124, 1992.
[10] K. Vivekanandarajah, T. Srikanthan, S. Bhattacharyya, "Energy-delay effficient filter cache hierarchy using pattern prediction scheme", IEE Proceedings - Computers and Digital Techniques, Vol. 151, Issue 2, March 2004.
[11] Ernst, D., Austin, T.M., Mudge, T., and Brown, R.B. "MiBench: a free commercially representative embedded benchmark suite", Proc. 4th IEEE Int. Workshop on Workload characterization, pp. 3 - 14, Dec. 2001.
[12] Chia-Lin Yang, Chien-Hao Lee, "HotSpot cache: joint temporal and spatial locality exploitation for i-cache energy reduction", ISLPED pp. 114-119, 2004
[13] P.-Y. Chang, M. Evers, and Y. Patt. "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference", Proc. Int. Conf. on Parallel Architectures and Compilation Techniques, Oct. 1996.
[14] D. Burger, A. Kagi, and M. Hrishikesh. "Memory hierarchy extensions to SimpleScalar 3.0", Technical Report TR99-25, Department of Computer Science, University of Texas at Austin, April 1999.
[15] David Tarjan, Shyamkumar Thoziyoor, Norman P. Jouppi, "CACTI 4.0", HP Laboratories Palo Alto HPL-2006-86, June 2, 2006.
[16] B. Case. SPEC2000 Retires SPEC92, The Microprocessor Report, vol. 9, 1995.
[17] EDN Embedded Microprocessor Benchmark Consortium, http://www.eembc.org.
[18] K. Vivekanandarajah et al., "Incorporating pattern prediction technique for energy efficient filter cache design", The 3rd IEEE International Workshop on SoC for Real-Time Applications, Vol. 30, pp. 44-47, 2003.

저 자 소 개

곽 종 욱(평생회원)
1998년 경북대학교 컴퓨터공학과 학사 졸업
2001년 서울대학교 컴퓨터공학과 석사 졸업
2006년 서울대학교 전기·컴퓨터 공학부 박사 졸업
2006년~2007년 삼성전자 SOC 연구소 책임 연구원
2007년~현재 영남대학교 컴퓨터공학과 조교수
<주 관심분야 : 컴퓨터 구조, 저전력 내장형 시스템, 모바일 멀티미디어 SOC 설계, 고성능 병렬처리>