대형 윈도우에서 다중 분기 예측법을 이용하는 수퍼스칼라 프로세서의 프로화일링 성능 모델

논 문
58-7-27

A Wide-Window Superscalar Microprocessor Profiling Performance Model Using Multiple Branch Prediction

이 종 복†
(Jong-Bok Lee)

Abstract - This paper presents a profiling model of a wide-window superscalar microprocessor using multiple branch prediction. The key idea is to apply statistical profiling technique to the superscalar microprocessor with a wide instruction window and a multiple branch predictor. The statistical profiling data are used to obtain a synthetical instruction trace, and the consecutive multiple branch prediction rates are utilized for running trace-driven simulation on the synthesized instruction trace. We describe our design and evaluate it with the SPEC 2000 integer benchmarks. Our performance model can achieve accuracy of 8.5 % on the average.

Key Words : Wide-window superscalar microprocessor, Multiple branch predictor, Profiling performance model, Statistical simulation

1. 서 론

마이크로 프로세서의 설계 초기 단계에서 다양한 하드웨어 사양에 대하여 모의실험을 통하여 그 성능을 정확하고 신속하게 측정하는 것이 매우 중요하다. 이러한 모의실험에는 실행 구동 모의실험 (execution-driven simulation)이나 트레이스 구동 모의실험 (trace-driven simulation)이 일반적이나, 최근에 이르러 프로화일링을 기반으로 하는 통계적 모의실험의 중요성이 대두되고 있다. 통계적 모의실험은 각 벤치마크에 대하여 프로화일링 기법에 의하여 그 특성을 추출하고 이것을 기반으로 벤치마크에 대하여 단축된 길이의 명령어 트레이스를 새로이 합성하여 모의실험을 수행하는 것이다. 이 방식을 채택하면 프로세서 성능에 대한 통찰력을 얻을 수 있으며, 벤치마크의 특성을 집약해서 추출해낸 결과를 이용하므로, 모의실험 시간이 대폭 단축되어 다양한 하드웨어를 갖는 마이크로 프로세서에 대하여 짧은 시간에 모의실험을 시행할 수 있다 [1-4]. 한편, 고성능 수퍼스칼라 마이크로 프로세서의 명령어 대역폭을 증가시키는 방편으로 다중 분기 예측법 또는 트레이스 캐쉬와 같은 방법이 널리 연구되고 있다 [5-7]. 다중 분기 예측법을 적용하면 한 싸이클에 2 개 및 3 개의 명령어 블럭을 동시에 인출 가능하므로, 이것을 효율적으로 활용하기 위하여 대형의 명령어 윈도우 크기가 필요하다. 본 논문에서는 크기 64에서 256에 이르는 대형 명령어 윈도우를 장착하고 다중 분기 예측법을 시행하는 수퍼스칼라 마이크로 프로세서에 대하여, 프로화일링 기법을 적용하여 통계적 모의실험을 적용하였다. 이 때, SPEC 2000 정수형 벤치마크 프로그램의 특성을 가장 잘 나타내는 부분을 발췌하여 이용함으로써 정확도를 높였으며, 기존의 트레이스 구동형 모의실험으로 측정한 결과와 비교하였을 때 평균 8.5 %의 정확도를 기록하였다.

† 교신저자, 정회원 : 한성대 공대 정보통신학과 부교수
E-mail : jblee@hansung.ac.kr
접수일자 : 2009년 4월 2일
최종완료 : 2009년 6월 3일

2. 통계적 프로화일링 기법

통계적 모의실험의 전 과정을 자세히 살펴보면 그림 1과 같이 크게 4 단계로 나누어지는데, 첫째, 일반적인 프로그램 트레이스의 발생, 둘째, 통계적 프로화일링 기법에 의한 분석 및 통계 데이터의 수집, 셋째, 통계적 트레이스의 합성, 넷째, 합성된 트레이스에 대한 통계적 트레이스 구동 모의실험이다. 첫 번째의 프로그램 트레이스 발생은 기존의 방법과 동일하다. 특정한 벤치마크 프로그램을 구체적인 명령어 트레이스 발생기를 통하여 명령어 트레이스를 생성한다. 두 번째의 통계적 프로화일링을 위한 분석 및 통계 데이터의 수집 단계는 다음과 같다. 우선, 첫 단계에서 얻은 프로그램의 명령어 트레이스를 분석하여 프로그램의 고유한 특성과 지역적 특성에 대한 통계값들을 추출해낸다. 프로그램의 고유한 특성은 프로그램을 구성하는 명령어의 유형별 구성비와 레지스터 피연산자의 개수 및 명령어 간의 데이터 종속에 대한 분포로 구성된다. 이 때, 명령어의 유형별 분포는 원래의 아키텍처가 갖는 명령어 집합보다 훨씬 적은 수의 간단한 명령어 집합으로 축소시킨다. 한편, 명령어 간의 상호 레지스터 종속은, 유형별 명령어의 소오스 레지스터와 그 레지스터를 목적 레지스터로 하여 종속을 부여하며 선행하는 명령어 간의 거리에 따라 측정한다. 이러한 특성은 주어진

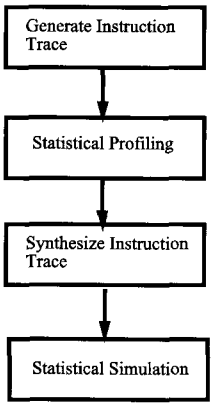

그림 1 통계적 모의실험의 흐름도
Fig. 1 Statistical simulation process

마이크로 프로세서의 구조와는 무관하고 단지 컴파일러와 마이크로 프로세서의 명령어 집합에만 영향을 받는 벤치마크 프로그램의 고유한 성질이다. 지역적 특성은 분기 미스율이나 캐쉬 미스율 등을 의미하며, 이것은 마이크로 프로세서 하드웨어 구조에 의하여 영향을 받는다. 이 단계는 각 벤치마크 프로그램에 대하여 단 한 번만 시행한다.

세 번째, 위에서 얻은 프로화일링 특성을 기반으로 하여 통계적 특성을 갖는 새로운 명령어 트레이스를 합성한다. 통계적 트레이스를 발생시키는 방법은, 0부터 1 사이의 난수를 발생시키고 이 값을 통계적 프로화일링에 의하여 얻은 누적 분포 함수에 대응시켜서 트레이스 합성에 필요한 값을 얻는 것이다. 즉, 명령어의 유형, 피연산자의 개수, 피연산자의 종속거리를 통계적 프로화일링을 기반으로 결정하여 명령어 코드를 만들어내는 작업을 반복하여 그 프로그램의 특성을 함축적으로 내포하는 새로운 통계적 트레이스의 합성이 가능하다.

마지막으로, 이렇게 합성된 트레이스를 분기 히트율 및 캐쉬 히트율에 대한 정보와 함께 수퍼스칼라 프로세서 모의실험기로 입력하여 실행함으로써 그 성능을 측정한다. 일단 통계적 프로화일링에 의하여 자료를 확보한 후에는, 하드웨어 조건을 바꿔서 전체 설계 공간에 대하여 다양한 시도를 할 수가 있다. 즉, 윈도우의 크기, 명령어의 인출율, 명령어의 실행 지연 싸이클, 파이프라인 단계의 수를 변화시켜가면서 새로운 결과를 단축된 시간에 간편하게 얻을 수가 있으므로 매우 유용하다.

3. 통계적 프로화일링과 다중 분기 예측법

3.1 다중 분기 예측법

다중 분기 예측법은 1 싸이클에 1 개의 분기 명령어의 결과를 예측하여 1 개의 기본블럭을 인출하는 것과 달리, 1 싸이클에 2 개 또는 3 개의 분기 명령어의 결과를 예측하여 각각 2 개 또는 3 개의 기본블럭을 동시에 인출하여 실행하는 것이다. 그림 2에 1 싸이클에 3 개의 분기 명령어의 결과를 예측하는 경우 관련된 기본블럭을 도시하였다. 첫 번째 분기 명령어의 결과에 따라 2 개의 기본블럭 중 1 개를, 두 번째 분기 명령어의 결과에 따라 4 개의 기본블럭 중 1 개를, 마지막으로 세 번째 분기 명령어의 결과에 따라 8 개의 기본블럭 중 1 개를 선택하게 된다. 만일 3 개의 분기 명령어에 대한 예측이 모두 옳다면, TTT 경로의 기본블럭을 선택하지만, 모두 틀리다면 NNN 경로의 기본블럭을 취한다.

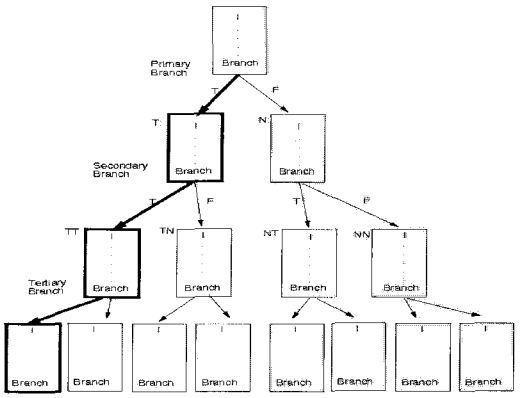

그림 2 다중 분기 예측과 기본블럭의 경로
Fig. 2 Multiple branch prediction paths with basic blocks

가장 간단한 다중 분기 예측법은 단일 분기명령어에 대하여 시행하는 2 단계 적응형 분기 예측법을 응용 및 확장한 것이다. 2 단계 적응형 분기 예측법은 분기 히스토리 레지스터에 최근의 분기 결과를 기록하고, 본 레지스터의 값으로 패턴 히스토리 테이블을 접근하여 2 비트 포화 카운터의 값에 따라 분기 여부를 예측하는 방법이다 [5]. 그림 3과 같이 다중 분기 예측을 수행하기 위하여, 히스토리 레지스터의 k 비트로 패턴 히스토리 테이블을 인덱스하여 첫 번째 분기명령어를 예측한다. 두 번째 분기명령어를 예측하기 위하여, 가장 우측의 k-1 분기 히스토리 비트로 패턴 히스토리 테이블을 인덱스한다. 이 때 인덱스를 위하여 k 비트 전부를 사용하지 않으므로, 패턴 히스토리 테이블 내의 두 개의 인접한 항목이 그 대상이 된다. 이 때 첫 번째 분기명령어를 이용하여 두 개의 항목 중에서 하나를 선택한다. 마찬가지로, 세 번째 분기명령어는 최우측의 k-2 개의 히스토리 레지스터를 사용하여 패턴 히스토리 테이블 내의 4 개의 인접한 항목을 접근한다. 그리고, 첫 번째와 두 번째 분기명령어의 예측을 바탕으로 세 번째 분기명령어에 대한 예측을 수행한다. 일반적으로, M 개의 다중 분기 예측을 수행할 때, 첫 번째, 두 번째, ..., M 번째 분기 명령어에 대하여, 히스토리 레지스터의 k 비트, k-1 비트, ..., k-M+1 비트로 패턴 히스토리 테이블을 각각 인덱스하여 각 분기 명령어를 예측한다.

그림 4는 3 차의 분기 예측을 위한 다중 분기 예측기, 명령어 캐쉬, 분기 어드레스 캐쉬의 구성도를 보인 것이다. 다중 분기 예측이 올바르게 수행되기 위하여, 분기 어드레스 캐쉬에는 예측이 수행되는 모든 분기 명령어에 대한 타겟 어드레스 및 순차 어드레스가 수록되어야 한다. 다중 분기 예측기의 예측 결과에 따라서 첫 번째 블럭, 두 번째 블록,

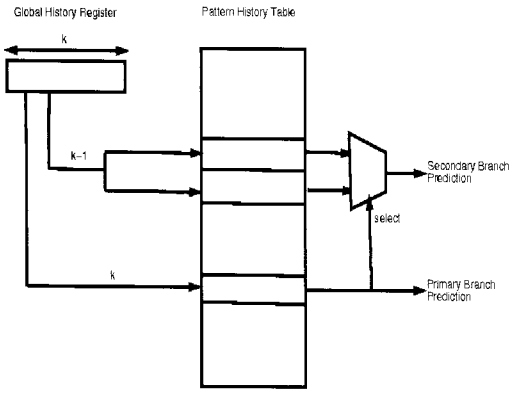

그림 3 다중 분기 예측법의 블럭도

Fig. 3 Multiple branch prediction block diagram

세 번째 블럭의 시작 어드레스를 분기 어드레스 캐쉬로부터 읽어서 명령어 캐쉬로 전송하여 세 개의 블럭을 동시에 인출한다. 세 번째 블럭의 어드레스는 명령어 캐쉬를 접근하는 동시에 분기 어드레스 캐쉬를 접근하여 다음 싸이클의 분기 예측을 대비한다.

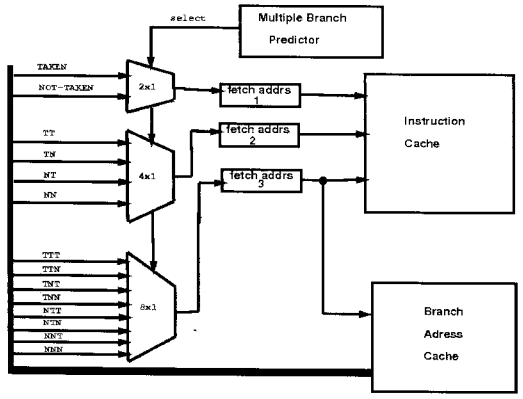

그림 4 다중 분기 예측기와 명령어 캐쉬, 분기 어드레스 캐쉬의 구성도

Fig. 4 Multiple branch predictor, instruction, data and branch address caches

3.2 다중분기예측법의 통계적 프로화일링 모의실험에 대한 적용

기존의 통계적 프로화일링 기법에서는 단일 분기명령어의 발생에 대한 분기 예측 정확도를 측정하여 트레이스를 합성하였으나, 본 논문에서 다중 분기 예측법을 통계적 프로화일링 기법에 적용하기 위하여 연속적인 분기 명령어의 예측에 대한 정확도를 측정하고 이것을 바탕으로 통계적 모의실험을 수행하였다. 즉, 2 차의 다중 분기 예측법을 시행할 때는 2 개의 분기명령어에 대한 예측이 연속으로 맞을 확률과 오직 1 개의 분기명령어에 대한 예측만 맞을 확률을 프로화일링한 후에, 통계적 모의실험에서 분기명령어를 만났을 때 난수를 발생시켜서 현재 분기명령어와 다음 분기명령어에 대하여 해당 비율만큼 2 개 연속 예측이 맞는 그룹과 1 개만 예측이 맞는 그룹으로 분류하고, 그 결과에 따라 분기 예측의 옳고 그름을 정하였다. 마찬가지로 3 차의 분기 예측에서는 각각 3 개, 2 개, 1 개의 분기명령어에 대한 예측이 옳을 확률을 프로화일링한 후에, 통계적 모의실험시 분기명령어를 만났을 때 현재 분기명령어와 두 번째, 세 번째 분기명령어에 대하여 3 그룹 중의 한 그룹으로 분류하여 해당 분기 예측의 옳고 그름을 결정하였다.

4. 실 험

4.1. 모의실험 입력용 벤치마크

입력 벤치마크 프로그램으로는 표 1에 보인 바와 같이 8 개의 SPEC 2000 정수형 프로그램이 사용되었다. 이 프로그램을 Fedora 8 Linux 2.6 운영체제하의 Pentium4 프로세서에서 Simplescalar 3.0의 크로스 컴파일러를 이용하여 MIPS-IV 명령어를 기반으로 하는 실행 가능 화일을 얻었으며 [8], 이 화일을 실행하면서 각 벤치마크마다 MIPS-IV 명령어 트레이스 1억 개를 발생시켰다. 이 때, 1억 개의 명령어는 그 프로그램의 특성을 가장 잘 나타낼 수 있는 부분을 SimPoint 3.0 도구를 이용하여 발췌함으로써, 각 벤치마크 프로그램이 종료할 때까지 모의실험을 하지 않고도 최종 성능과 근접하게 나오도록 하였다 [9]. 통계적 프로화일링 방식에 의하여, 97 개의 명령어 집합으로 구성되는 MIPS-IV 명령어를 단지 13 개의 대표적 명령어로 축소하여 새로운 명령어 트레이스를 합성하였다. 이 13 개의 명령어는 INTEGER, LOAD, STORE, BRANCH, FP_ADD, FP_MUL, FP_DIVS, FP_DIVD, FP_LOAD, FP_STORE, FP_CONV, FP_COMP, FP_INT로 구성된다. 한편, 13 개의 명령어는 각 명령어 특성에 따라서 피연산자 레지스터가 0 개, 1개, 2개인 것으로 다시 분류된다.

표 1 SPEC 2000 정수형 벤치마크 프로그램

Table 1 SPEC 2000 integer benchmark programs

벤치마크	설명
bzip2	압축
crafty	체스 경기 놀이
gap	그룹 이론 해석기
gcc	C 프로그래밍 언어 컴파일러
gzip	압축
mcf	조합 최적화
parser	워드 프로세서
vpr	FPGA 회로 배치 및 배선

4.2. 모의실험 하드웨어 사양

표 2는 본 모의실험에 대한 수퍼스칼라 마이크로 프로세서의 하드웨어 사양을 나타낸 것이다. 대형 명령어 윈도우에 대한 모의실험을 수행하기 위하여 그 크기를 64, 128, 256로 정하였다. 한편, 정수형 벤치마크의 기본블럭의 크기

가 평균 5이므로 이것을 기준으로 3 개의 다중 분기 예측을 감안하여, 매 싸이클당 인출할 수 있는 명령어의 최대 수인 인출률을 16으로 모든 윈도우에 동일하게 적용하였다. 또한, 윈도우에서 매 싸이클당 연산유닛으로 이슈할 수 있는 명령어의 최대 개수인 이슈율과, 실행이 완료되어 윈도우에서 퇴거하는 명령어의 최대 개수인 퇴거율도 16으로 정하였다.

표 2 모의실험에 이용된 아키텍쳐 하드웨어 사양
Table 2 Architecture configuration for simulation

항목	값
명령어 윈도우의 크기	64, 128, 256
인출율, 이슈율, 퇴거율	16
정수형 ALU	16
로드/스토어	8
실수형 덧셈기	4
실수형 곱셈기/나눗셈기	4
1 차 명령어 캐쉬	64 KB, 2 차 연관, 16 B 미스 페날티 10 싸이클
1 차 데이터 캐쉬	64 KB, 직접, 32 B 미스 페날티 10 싸이클
분기 어드레스 캐쉬	2 K 엔트리
분기 예측기	14 비트 전역 히스토리 방식 미스 페날티 6 싸이클
이슈 지연 싸이클	정수형(1), 분기(1), 로드(1), 스토어(1), 실수덧셈(1), 단일 실수 곱셈(1), 2 배 실수 곱셈(1), 단일 실수 나눗셈(4), 2 배 실수나눗셈(7),
결과 지연 싸이클	정수형(1), 분기(1), 로드(1), 스토어(1), 실수덧셈(1), 단일 실수 곱셈(3), 2배 실수 곱셈(3), 단일 실수 나눗셈(6), 2 배 실수 나눗셈(9),

명령어 캐쉬와 데이터 캐쉬는 1 차에 한하여 적용하였으며, 2 차 캐쉬는 100 % 히트율을 가정하였다. 정수형 명령어 대부분의 지연은 1 싸이클이지만, 실수형의 경우 3 싸이클에서 9 싸이클까지의 다양한 값을 갖는다.

4.3. 트레이스 구동 모의실험기

본 논문에서 다중 분기 예측법을 시행하며 대형 윈도우를 갖는 수퍼스칼라 마이크로 프로세서에 대한 트레이스 구동 모의실험기가 필요하므로 같은 환경에서 C 언어와 gcc-4.1 컴파일러를 이용하여 자체 개발하였다. 그림 5는 트레이스 구동 모의실험기에 대한 전 과정을 나타낸다.

Config 함수에서는 모의실험기가 수행해야하는 명령어의 개수, 윈도우의 크기, 다중 분기 예측 차수, 인출율, 이슈율, 퇴거율에 대한 입력을 받아들여서 설정한다. Init 함수에서는 레지스터 화일을 비롯한 각종 하드웨어를 초기화한다. Fetch_insr 함수에서는 다중 분기 예측법에 의하여 한 싸이클에 최대 인출율에 해당하는 명령어를 명령어 캐쉬로 부터 인출하여 디코더로 가져온다. Decode 함수에서 명령어들을 해독하고 각 명령어의 타임스탬프 (timestamp)를 부여한다. 타임스탬프는, 명령어간에 실제 종속 (true dependency)을

```
main()
{
 Config();
 Init();
 Fetch_instr();
 Decode();
 Issue();
 Execute();
 Commit();
}
```

그림 5 다중 분기 예측 모의 실험기
Fig. 5 Multiple branch predictor simulator

제외한 종속 관계를 제거하는 재명명(renaming)과 더불어, 윈도우 내의 각 명령어가 이슈되는 시점을 설정하는 효과가 있다. 타임스탬프를 이용하여 명령어의 종속관계를 구현하기 위하여 디코드된 명령어마다 타임스탬프를 가져야하고, 또한 각 레지스터의 타임스탬프를 기록하는 레지스터-타임스탬프(RT) 테이블을 유지한다. 초기에 명령어의 타임스탬프는 현재 싸이클로 설정되고, 각 레지스터의 타임스탬프는 0으로 설정된다.

$$\begin{aligned} I_1 &: R_2 \leftarrow R_1 + K \\ I_2 &: R_3 \leftarrow R_2 \wedge L \\ I_3 &: R_4 \leftarrow R_2 \oplus R_3 \\ I_4 &: R_4 \leftarrow R_1 - M \\ I_5 &: R_5 \leftarrow R_4 \vee N \end{aligned} \tag{1}$$

표 3 싸이클의 변화에 따르는 각 레지스터와 명령어의 타임스탬프 값의 변화
Table 3 Register and instruction time stamp values as cycle advances

싸이클 및 타임스탬프	t	t+1	t+2	t+3	t+4
R_1	0	0	0	0	0
R_2	**t+1**	t+1	t+1	t+1	t+1
R_3	0	**t+2**	t+2	t+2	t+2
R_4	0	0	**t+3**	**t+1**	t+1
R_5	0	0	0	0	**t+2**
I1	**t+1**	t	t	t	t
I2	t	**t+2**	t+2	t+2	t+2
I3	t	t	**t+3**	t+3	t+3
I4	t	t	t	**t+1**	t+1
I5	t	t	t	t	**t+2**

수식 1 프로그램에 나타낸 명령어가 싸이클 t에서 싸이클 t+4까지 수행될 때의 레지스터와 명령어의 타임스탬프 값의 변화를 표 3에 나타냈다. 이 때, 모든 명령어는 1 싸이클의 실행 지연 시간을 갖는다고 가정한다. 임의의 명령어는 RT 테이블을 조회함으로써 자신의 소오스 레지스터의 타임스탬프를 알 수 있다. 어떤 명령어의 소오스 레지스터 1 개 또는 2 개의 타임스탬프 값을 조회한 결과, 모두 명령어 자신의 타임스탬프보다 그 값이 같거나 작다면, 이 명령어는 독립이고 이슈 가능하다. 모든 이슈 가능한 명령어는 현재 싸

이클에 그 명령어의 결과 지연 싸이클(result latency)을 더한 값으로 그 명령어의 타임스탬프를 설정한다. 그리고, 그 명령어의 결과 레지스터의 타임스탬프도 이 값으로 갱신하여 RT 테이블에 수록한다. 한편, 어떤 명령어의 소오스 레지스터의 타임스탬프가 명령어 자신의 타임스탬프보다 그 값이 크다면, 이 명령어는 선행하는 명령어에 종속이다. 이 때는, 해당 소오스 레지스터의 타임스탬프에 자신의 결과 지연 싸이클 수를 더하여 명령어 자신의 타임스탬프 및 결과 레지스터의 타임스탬프를 갱신해야 한다. 이렇게 함으로써, 선행하는 명령어가 실행을 완료한 후에, 그것에 종속인 명령어가 실행되는 것을 정확히 모델링할 수 있다. 만일 임의의 명령어의 1 개 또는 2 개의 소오스 레지스터의 타임스탬프가 모두 명령어 자신의 타임스탬프보다 크다면, 둘 중 큰 값을 명령어 및 결과 레지스터의 타임스탬프로 갱신해야한다. 이같이 각 명령어들은 RT 테이블을 통하여 서로 레지스터의 타임스탬프 값을 주고받음으로써 명령어 간의 종속관계에 기반한 실행이 가능하다.

한편, 타임스탬프에 의하여 각 명령어의 실행 가능 시점이 계산되었더라도, 분기 예측 오류가 발생하였거나, 연산 유닛의 개수의 제한 또는 메모리 종속 제한이 발생하였을 때는 그 명령어 및 그 명령어에 종속인 모든 명령어의 실행이 금지된다. 또한 비순서방식(Out-of-order)에 의하여 실행 시점이 계산되었더라도, 프로그램의 순차성을 보존하기 위하여 명령어들은 윈도우에서 순서퇴거(In-order retirement)된다.

5. 모의실험 결과

표 4에 8 개의 벤치마크에 대하여 2 단계 적응형 다중 분기 예측법을 이용하여 측정한 정확도를 나타내었다. 2 차와 3 차의 분기 예측의 경우, 개별 분기 명령어에 대한 예측 정확도가 아닌, 2 개 및 3 개 분기명령어에 대한 예측이 연속적으로 맞을 정확도로 나타내었다. 2 차 분기 예측에서, 2 개 분기명령어가 연속으로 예측이 맞을 확률은 평균 87.2 % 이며, 1 개 분기명령어의 경우는 4.7 %이다. 한편, 3 차의 분기 예측에서는 3 개, 2 개 및 1 개 분기명령어가 연속적으로 올바르게 예측될 평균 확률은 각각 80.8 %, 4.5 %, 2.5 %이다. 이 결과에서 알 수 있듯이, 2 차 및 3 차 분기 예측법에서 각각 2 개와 3 개 분기명령어가 연속으로 예측이 옳을 확률이 1 개 또는 2 개가 부분적으로 옳을 확률보다 지배적이다.

그림 6은 윈도우의 크기가 64일 때 프로화일링에 의하여 정수형, 로드, 스토어, 분기 명령어가 선행하는 명령어에 종속일 확률을 거리에 대하여 측정한 것을 8 개의 벤치마크에 대하여 그 평균값을 도시한 것이다. MIPS-IV 명령어 집합에서 정수형은 피연산자 레지스터가 2 개이므로, 첫 번째 레지스터가 선행하는 명령어에 종속일 확률과 두 번째 레지스터가 선행하는 명령어에 종속일 확률을 별도로 측정하였다. 이 그래프에 의하면, 정수형 명령어의 첫 번째 또는 두 번째 레지스터가 거리가 1 떨어진 선행하는 명령어에 종속일 확률은 약 60 %이고, 거리가 2만큼 떨어진 명령어에 종속일

표 4 다중 분기 예측 정확도 (%)

Table 4 Multiple branch prediction accuracies (%)

벤치마크	2 차		3 차		
	2-블럭	1-블럭	3-블럭	2-블럭	1-블럭
bzip2	99.6	0.1	99.5	<0.1	<0.1
crafty	86.4	6.5	80.2	6.2	5.9
gap	89.7	4.3	86.4	3.3	2.7
gcc	81.8	8.2	75.0	6.8	5.5
gzip	87.4	5.4	79.3	6.3	1.2
mcf	88.8	4.9	84.7	3.4	1.3
parser	85.2	7.0	79.1	6.0	2.1
vpr	78.9	1.5	61.9	3.6	1.2

확률은 약 20 %이다. 로드 명령어의 경우 피연산자는 첫 번째 레지스터만 존재하며, 거리가 1만큼 선행하는 명령어에 종속일 확률은 45 %이나, 거리가 2인 경우에는 6 %로 급격히 하강하였다. 스토어 명령어의 경우, 소오스 레지스터와 결과 레지스터가 종속인 경우를 각각 구분하여 종속확률을 측정하였다. 대부분의 명령어는 거리 3 이내에서 선행하는 명령어에 종속이며, 명령어 간의 거리가 3 이상으로 증가할수록 종속일 확률은 급격히 감소하는 양상을 보였다.

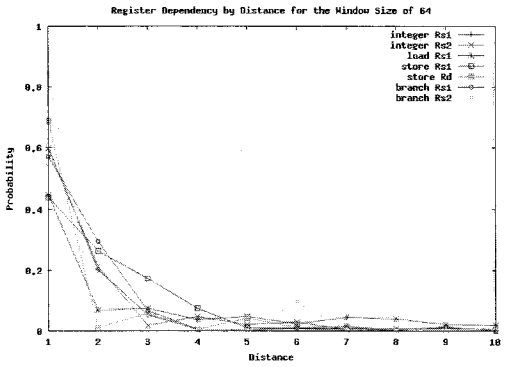

그림 6 윈도우 크기가 64일 때, 거리별 명령어간 레지스터의 종속율을 평균한 결과

Fig. 6 Register dependence rates by distance for window size of 64

그림 7은 모의실험에 이용된 벤치마크에 대하여 프로화일링으로 측정한 명령어의 유형별 분포율을 정수형, 로드, 스토어, 분기어 순으로 나타낸 것이다. 대부분의 벤치마크에서 정수형 명령어의 비율이 가장 높게 나타났으며, 특히 crafty와 gzip에서 54 % 이상을 선회하였다. 각 명령어 유형의 평균값을 고찰하면, 정수형 명령어의 평균 분포율이 가장 높은 44.8 %를 기록하였으며, 로드형 명령어가 26. 8%, 분기 명령어가 15.0 %가 그 뒤를 이었고, 스토어 명령어의 비율이 11.3 %로 가장 낮았다. 한편, 각 명령어 피연산자 개수의 분포율도 측정하였으며, 정수형 명령어의 경우 2 개의 소오스

레지스터가 모두 존재하는 빈도율이 55. 3 %,로 가장 높았으며 1 개의 소오스 레지스터는 39.1 %로 나타났고, 소오스 레지스터가 존재하지 않는 특수 명령어는 5 % 미만의 낮은 값을 기록하였다. 각 벤치마크의 명령어 유형 분포율은 각 명령어의 피연산자 개수와 함께, 축소된 명령어 집합으로 새로운 명령어 트레이스를 합성할 때 이용된다.

그림 8에서부터 그림 10은 기존의 트레이스 구동형 모의실험으로 다중 분기 예측법을 이용하는 수퍼스칼라 마이크로 프로세서의 성능을 측정한 결과와, 본 논문에서 제안하는 프로화일링 기반의 통계적 모의실험에 의하여 측정한 성능을 IPC(Instruction Per Cycle)로 나타내어 비교한 것이다. 각 벤치마크 프로그램에 대하여, 1 차, 2 차와 3 차의 다중 분기 예측 정확도에 대하여 두 가지 방식으로 측정한 프로세서의 성능을 각각 마디 및 실선과 점선으로 연결한 그래프로 함께 도시하였다. 이 때 성능의 상대오차 ΔIPC는 IPC_{real}이 트레이스 구동 방식으로 측정한 성능이고, IPC_{model}이 프로화일링 방식에 의하여 측정한 값일 때 수식 2와 같이 나타냈다.

$$\Delta IPC = \frac{|IPC_{real} - IPC_{\mathrm{model}}|}{IPC_{real}} \times 100(\%) \qquad (2)$$

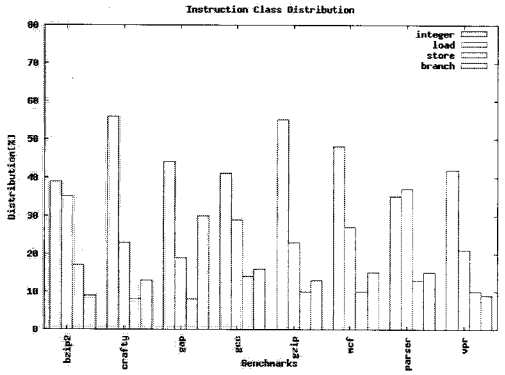

그림 7 벤치마크 프로그램의 정수형 명령어 유형 분포
Fig. 7 Instruction class distribution for integer benchmark programs

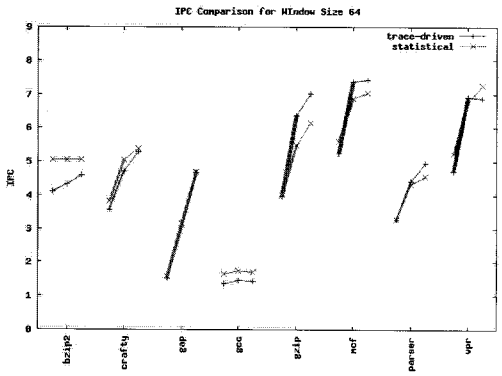

그림 8 윈도우 크기 64일 때 성능의 비교
Fig. 8 IPC comparison for window size of 64

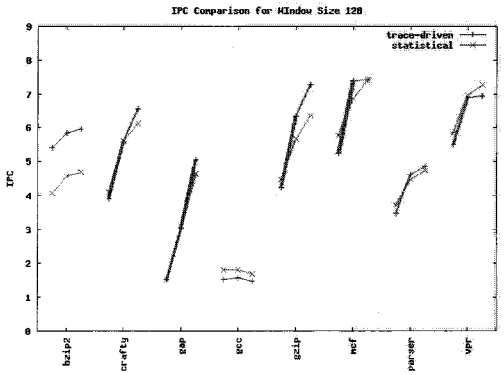

그림 9 윈도우 크기 128일 때 성능의 비교
Fig. 9 IPC comparison results for a window size of 128

그림 8은 윈도우의 크기가 64인 경우의 8 개의 벤치마크에 대한 실험 결과로서, 기하평균을 구하였을 때 실측값은 2 차 분기 예측에서 4.4 IPC, 3 차 분기 예측에서 4.8 IPC를, 모델값은 각각 4.5 IPC, 4.9 IPC를 나타내었다. 1 차에서 2 차의 다중 분기 예측으로 이행하였을 때 1.4 배, 1 차에서 3 차로 이행하였을 때 1.5 배 성능이 향상되었다. Parser의 2 차 분기 예측의 경우 실측값과 모델값의 상대오차가 2.3 %로 최저를 나타냈으며, gcc의 3 차 분기 예측에서 최고 19.8 %의 상대오차를 기록하였다. 이 때 2 차와 3 차의 다중 분기 예측에 의한 모델값의 상대오차의 평균값은 8.4 %를 나타내었다.

그림 9는 윈도우의 크기가 128로 확장되었을 때의 결과를 동일한 방식으로 나타낸 것이다. 윈도우의 크기가 2 배로 확장됨에 따라, 3 차의 분기 예측에 대하여 실측값과 모델값이 각각 5.2 IPC, 5.0 IPC를 기록하였다. 실측값과 모델값을 비교하면, Bzip2 2 차의 분기 예측에서 최고 21.9 %의 오차를, mcf 3 차에서 최저 0.3 %의 오차를 기록하였다.

마지막으로 그림 10은 윈도우의 크기를 256으로 최대 확장하였을 때의 결과이다. 주어진 조건의 다중 분기 예측법을 활용할 수 있는 윈도우의 크기가 이미 포화되어서 윈도우의 크기가 64에서 128이 되었을 때보다 성능이 크게 개선되지 않음을 알 수 있다. 그러나 vpr의 경우 윈도우의 크기가 128에서 256으로 확대되었을 때 성능이 약 1.1 배 증가하였다.

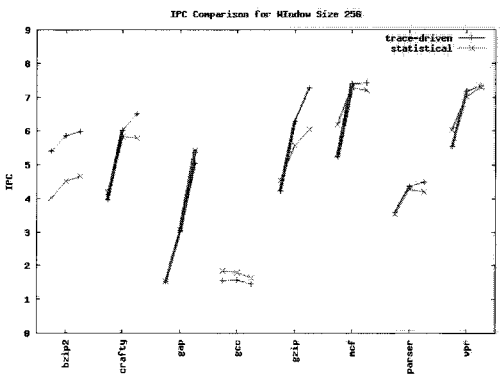

그림 10 윈도우의 크기 256일 때의 성능 비교
Fig. 10 IPC comparison results for a window size of 256

이상 3 가지 대형 윈도우에서 2 차 및 3 차의 다중 분기 예측법을 이용하는 수퍼스칼라 마이크로 프로세서에 대한 프로화일링 방식의 통계적 모의실험에 대한 정확도를 분석하면, 모델에 의한 성능이 실측값과 동일한 경향을 보이고 매우 근사함을 확인할 수 있다. 비록 특정 벤치마크에서 최대 22 %의 오차를 보이고 있지만, 이와 반대로 최소 0.3 의 오차로 거의 일치한 경우도 있으며, 3 가지 윈도우에 대하여 각각 평균하면 8.2 %에서 8.8 % 범위의 정확도를 기록하였다.

6. 결 론

고성능 수퍼스칼라 마이크로 프로세서에서 명령어 대역폭을 증가시키기 위한 방법으로 다중 분기 예측법이 이용되고 있으며, 그 성능을 예측하기 위하여 짧은 시간에 모의실험을 수행할 수 있는 모델이 필요하다. 본 논문에서는 다중 분기 예측기를 효율적으로 활용할 수 있도록 대형 명령어 윈도우를 이용하는 수퍼스칼라 마이크로 프로세서를 대상으로 하여, 프로화일링을 기반으로 하는 통계적 모델을 제안하였다. 이것을 위하여 64, 128, 256의 크기를 갖는 명령어 윈도우를 대상으로 매 싸이클마다 다중 분기 예측법에 의하여 최고 16 개의 명령어를 인출하여 기존의 트레이스 구동 방식으로 측정한 값과 비교하였다. 그 결과, 평균 8.5 %의 높은 정확도를 보여 그 우수성을 확인 할 수 있었다. 향 후의 연구과제로는, 명령어 쌍 사이의 종속성을 강화한 프로화일링 기법에 대한 연구를 수행하여 모델에 의한 정확도를 더욱 향상시키는 것이다.

감사의 글

본 연구는 2008년도 한성대학교 교내연구비 지원과제임.

참 고 문 헌

[1] R. Carl and J. E. Smith, "Modeling Superscalar Processors via Statistical Simulation," Workshop on Performance Analysis and Its Impact on Design, Jun. 1998.

[2] L. Eeckout, K. D. Bosschere, and H. Neefs, "Performance Analysis through Synthetic Trace Generation," International Symposium on Performance Analysis of Systems and Software, Apr. 2000.

[3] S. Nussbaum and J. E. Smith, "Modeling Superscalar Processors via Statistical Simulation," International Conference on Parallel Architectures and Compilation Techniques, pp.15-24, Sep. 2001.

[4] L. Eckout, R. H. Bell Jr., B. Stougie, K. D., Bosschere, and L. K. John, "Control Flow Modeling in Statistical Simulation for Accurate and Efficient Processor Design Studies," International Symposium on Performance Analysis of Systems and Software, 2004.

[5] T.-Y. Yeh, D. Marr, Y. Patt, "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache," The 7th International Conference on Supercomputing, pp. 67-76. 1993,

[6] R. Rakvic, B. Black, and J. P. Shen, "Completion time multiple branch prediction for enhancing trace cache performance," Annual International Symposium on Computer Architecture", pp. 47-58. 2000.

[7] D. M. Koppelman, "The Benefit of Multiple Branch Predicition on Dynamically Scheduled Systems," Annual International Symposium on Computer Architecture", pp. 42-51, May. 2002,

[8] T. Austin, E. Larson, and D. Ernest, "SimpleScalar : An Infrastructure for Computer System Modeling," Computer, vol. 35, no. 2, pp. 59-67, Feb. 2002.

[9] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0 : Faster and More Flexible Program Analysis," Workshop on Modeling, Benchmarking and Simulation, Jun. 2005.

저 자 소 개

이 종 복 (李 鍾 馥)

1964년 8월 20일생. 1988년 서울대 컴퓨터공학과 졸업. 1998년 동 대학 전기공학부 졸업(공박). 1998~2000 LG반도체 선임연구원. 2000년~현재 한성대 정보통신공학과 부교수

Tel : 02-760-4497

Fax : 02-760-4435

E-mail : jblee@hansung.ac.kr