논문 2007-44CI-2-1

# 고성능 프로세서를 위한 분기 명령어의 동적 History 길이 조절 기법

## ( Dynamic Per-Branch History Length Fitting for High-Performance Processor )

곽 종 욱*, 장 성 태**, 전 주 식***

( Jong Wook Kwak, Seong Tae Jhang, and Chu Shik Jhon )

요 약

분기 명령어에 대한 분기 예측 정확도는 시스템 전체의 성능 향상에 중대한 영향을 미친다. 본 논문에서는 분기 예측의 정확도를 높이기 위한 방법의 하나로, 각 분기 명령어 별로 사용되는 History의 길이를 동적으로 조절할 수 있는 "각 분기별 동적 History 길이 조절 기법"을 소개한다. 제안된 기법은, 분기 예측에 있어서 관련된 레지스터들 사이의 데이터 종속성을 추적하여, 최종적으로 관련이 있는 레지스터를 포함하도록 유도하는 분기를 파악한 후, 관련 분기의 History만을 사용하게 해 주는 방식이다. 이를 위해 본 논문에서는, 데이터 종속성을 추적할 수 있는 알고리즘과 관련 하드웨어 모듈을 소개하였다. 실험 결과 제안된 기법은, 기존의 고정 길이 History를 사용하는 방식에 비하여 최대 5.96% 분기 예측 정확도의 향상을 가져 왔으며, 프로파일링을 통해 확인된 각 응용 프로그램 별 Optimal History 길이와 비교해서도 성능 향상을 보였다.

Abstract

Branch prediction accuracy is critical for the overall system performance. Branch miss-prediction penalty is the one of the significant performance limiters for improving processor performance, as the pipeline deepens and the instruction issued per cycle increases. In this paper, we propose "Dynamic Per-Branch History Length Fitting Method" by tracking the data dependencies among the register writing instructions. The proposed solution first identifies the key branches, and then it selectively uses the histories of the key branches. To support this mechanism, we provide a history length adjustment algorithm and a required hardware module. As the result of simulation, the proposed mechanism outperforms the previous fixed static method, up to 5.96% in prediction accuracy. Furthermore, our method introduces the performance improvement, compared to the profiled results which are generally considered as the optimal ones.

**Keywords**: 분기 예측, 분기 예측 정확도, 데이터 종속성, 히스토리, 히스토리 길이 조절

## I. 서 론

최근 마이크로 프로세서의 발전 경향은 파이프라인의 단계수가 늘어나고 단위 시간당 제시되는 명령어들의 수가 증가하는 추세이다. 이와 같은 환경 하에서는, 분기 명령어들의 분기 예측 실패에서 오는 예측 실패 비용(Miss-Prediction Penalty)도 더불어 증가한다. 따라서 분기 명령어들에 대한 분기 예측 정확도는 시스템 전체 성능 향상에 중요한 영향을 미친다[1]. 오늘날까지 제안된 다양한 분기 예측기 가운데 가장 대표적이며 이후의 설계에 큰 영향을 준 기법이 Yeh and Patt의 이단계 적응형 분기 예측 기법(Two-level Adaptive Branch Prediction)이다[2]. 이 방식은 각 분기 명령어들 사이의 존재하는 상호 연관성(correlation)을 이용하여 분기 방향을 예측하는 방식이다. 그리고 이와 같은 이

* 정회원, 삼성전자반도체, SOC 연구소, Processor Architecture Lab.,
(SOC R&D Center, Samsung Electronics)
** 정회원, 수원대학교 컴퓨터학과
(Department of Computer, The University of Suwon)
*** 정회원, 서울대학교 전기컴퓨터공학부
(Department of EECS, Seoul National University)
※ 이 논문은 2005년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2005-202-D00388)
접수일자: 2006년10월30일, 수정완료일: 2007년2월21일

단계 적응형 분기 예측 기법의 소개 이후로, 분기 명령어들의 History는 그들의 주소 값과 함께 오늘날의 분기 예측에 있어서 가장 중요한 입력 요소 가운데 하나로 자리 잡았다. 하지만, 지금까지 소개된 모든 분기 예측기들은 고정된 길이의 Branch History를 사용하였다.

한편, 분기 예측의 정확도를 높이기 위한 연구로서 새로운 분기 예측기의 설계도 있을 수 있지만, 특정 예측기 하에서 분기 예측의 정확도를 추가적으로 향상시키기 위한 연구들도 많이 진행되어 왔다. 그 가운데 가장 대표적인 방법이 분기 예측에 사용되는 입력 요소들의 효과적인 활용이다. 분기 예측기에 추가적으로 사용될 수 있는 새로운 입력 요소의 제안[3], 각 분기 History를 투기적으로 사용하는 방식(Speculative History)[4]등이 이에 해당하는 사례들이다.

본 논문에서는 이와 같이 분기 예측에 필요한 입력 요소들을 효과적으로 활용하고자 하는 노력의 연장선상에서, "각 분기 명령어 별 최적의 History를 사용하는 방법을 제안한다. 전술한 바와 같이, 기존의 모든 분기 예측기들은 고정된 길이의 History를 사용하였다. 하지만 이전의 여러 연구에서 보여주듯이, 각 분기 명령어가 사용하는 History는 나름대로 최적의 길이를 가지며, 많은 정보, 즉 긴 길이의 History를 사용한다고 항상 좋은 것은 아니다[5][6]. 분기 명령어의 주소 값은 각 분기 명령어를 유일하게 구분 짓는 요소이지만, 분기 History는 분기 명령어들 사이에 존재하는 상호 연관성을 반영하는 요소이기 때문이다.

결국, 다양한 형태로 존재하는 분기 명령어 별 최적의 상호 연관성을 추출하는 것이 문제 해결의 핵심이다. 본 논문에서는 분기 명령어들 사이에 존재하는 최적의 상호 연관성을 추출해 내기 위해, 분기 명령어의 수행과 관련된 레지스터들 사이 존재하는 데이터 의존성(Data Dependency)을 분석한다. 그리고 추출된 데이터 의존성을 바탕으로 각 분기 명령어 별 최적의 상호 연관성을 반영하는 History 길이를 찾고, 이를 수행 시간에 동적으로 조절하여 사용함으로써 분기 예측의 정확도 향상에 기여하고자 한다. 이하 본 논문의 구성은 다음과 같다. Ⅱ절에서는 배경지식에 대해 설명하며, 관련연구로 History 길이를 다양하게 조절하고자 하는 기존의 연구들에 대해 소개한다. Ⅲ절에서는 데이터 의존성에 기반 한 각 분기 명령어 별 동적인 History 조절 기법에 대해 제시한다. 그리고 Ⅳ절에서는 모의실험을 통해 기존 방식과의 성능상의 차이를 비교 분석하며, Ⅴ절에서 결론을 맺는다.

## Ⅱ. 배경 지식 및 관련 연구

Yeh and Patt에 의해 제안된 이단계 적응형 분기 예측 기법 이후로, 분기 명령어의 History는 오늘날의 분기 예측에 있어서 중요한 입력 요소 가운데 하나로 자리 잡았다[2]. 이와 같은 분기 History는 각 분기 명령어들 사이에 존재하는 상호 연관성을 이용하는 방식이다. 다음과 같은 예제 코드를 살펴보자.

그림 1에 제시된 예제 코드의 경우, 마지막 분기 명령어인 Br3의 분기 방향은 Br1 혹은 Br2의 결과에 강하게 연관되어 있다는 것을 알 수 있다. 즉 Br1과 Br2의 분기 결과를 알고 있으면, Br3의 결과는 쉽게 예측이 가능하다는 것이다. 이처럼 분기 명령어들은 경우에 따라서 다른 분기 명령어들 사이에 강한 상호 연관성을 가질 수 있다. 그림 1의 예제 코드와 같은 경우를 분기 방향 연관성(Direction Correlation)이라고 한다[7].

앞선 예제 코드와는 달리 그림 2에 제시된 예제 코드에서는, BrC의 분기 방향이 BrD와 아무런 연관이 없다는 것을 알 수 있다. 즉 분기 방향 연관성은 존재하지 않는다. 하지만, 프로그램의 수행이 BrC에 도달하였을 경우, 마지막 분기 명령어인 BrD의 조건이 만족하리라는 것을 알 수 있다. 이처럼 프로그램 수행 경로 상에서 존재하는 또 다른 형태의 상호 연관성을 수행 경로 연관성(In-Path Correlation)이라고 한다[7].

일반적으로 위의 두 예제 코드에서 보여주듯이, 현재 예측하고자 하는 분기 명령어와 실행 순서상 가까운 이전 분기 명령어들일수록 보다 더 강한 상호 연관성을 가지게 된다. 하지만, 항상 그런 것은 아니다. 예를 들어, 비교적 자주 등장하지 않는 분기 명령어로, 제한된 길이의 Global Branch History 내에서 겨우 동작 패턴을 파악할 수 있을 정도로 드물게 수행되는 가장 바깥쪽의 커다란 Loop Body를 가지는 분기 명령어의 경우, 현재 수행하고자 하는 분기 명령어와 수행 순서상 거리

```
...
if (condA)        Br1
...
if (condB)        Br2
...
if (condA AND condB)   Br3
...
```

그림 1. 분기 방향 연관성 코드 예제
Fig. 1. Direction Correlation Example.

```
...
if (NOT(cond1)) BrA
...
else if (NOT(cond2)) BrB
...
else if (cond3)  BrC
...
...
if (cond1 AND cond2) BrD
...
```

그림 2. 수행 경로 연관성 코드 예제
Fig. 2. In-Path Correlation Example.

가 멀리 떨어져 있을 수 있으나, 이 역시 강한 상호 연관성을 가질 수 있다. 참고적으로 이 같은 경우는 Global Branch History 에서 보다 Local Branch History 내에서 보다 더 효과적으로 파악될 수 있다. 결국 문제의 핵심은, 각 응용 별로 그리고 각 분기 명령어 별로 상호 연관성을 가지는 분기 명령어들이 다양하며, 이들 사이의 동적인 수행 경로상의 거리(즉, 길이)도 다양하다는 것이다. 이와 같이, 분기 예측에 있어서 사용되는 분기 명령어들의 *History* 길이를 조절하기 위한 기존의 연구들은 다음과 같다.

History 길이를 조절하기 위한 연구의 최초로, DHLF (Dynamic History Length Fitting) 방법을 들 수 있다[8]. 이 방식은 프로그램 수행을 다수의 interval로 구분하여, 각 interval 별 분기 예측의 정확도 변화를 살핀다. 그리고 그 결과에 따라 다음 interval까지 사용될 History 길이를 결정한다. 하지만, 이 방식은 프로그램 수행 중간에 많은 interval이 포함되어 있기 때문에 오히려 전체 프로그램의 수행 시간을 증가시킬 수 있다. 또한 제안된 방식은 완전한 형태의 동적인 기법이라고는 할 수 없다. Elastic History Buffer를 이용한 History 길이 조절 기법도 제안되었다[9]. 이 방법은 각 분기 명령어 별(Per-Branch)로 최적의 History 길이를 찾는다는데 그 차이가 있다. 하지만 이 방법은 실제 프로그램의 수행 이전에, 사전 프로파일링(prior-profiling)을 요구하는 방식으로 동적인 History 길이 조절 기법이라 할 수 없다. History 길이를 변경시키는 연구와는 별도로, 분기 명령어들의 이전 목표 주소(Target Address)들의 조합으로 분기 예측 테이블(PHT, Pattern History Table)을 접근하는 방식이 제안되었다. 이 때 목표 주소의 조합을 매번 달리하는, 가변 길이 경로 분기 예측 기법(Variable Length Path Prediction)도 소개 되었다[10]. 이 같은 방법은 분기 History를 사용하는 방식이 아니며, 이 역시 프로그램 수행 이전에 프로파일링 단계를 필요로 하는 정적(static)인 방식이다. 한편, 오늘날에는 다중 분기 예측기(Hybrid Branch Predictor)들이 다수 존재한다. 그리고 이와 같은 다중 분기 예측기의 다양한 구성 요소(Sub-Predictor)들에서 사용되는 입력 History의 길이를 각 예측기 별로 서로 다르게 구성하여 그 결과를 선택적으로 사용(voting) 방법들이 제시되고 있다[11]. 하지만, 이같은 방식은 많은 하드웨어 자원의 활용이 가능한 다중 분기 예측기 하에서 가능한 방식이며, 어디까지나 History 길이의 조절 효과를 간접적으로 구현하는 방식이다.

이상에서와 같이, 기존의 방법들은 대부분 History 길이를 조정하기 위한 사전 프로파일링 단계를 필요로 하거나, 적어도 수행 중간에 History 길이 변환을 위한 interval을 필요로 한다. 이러한 방식은 실제 프로그램의 수행 시에 동적으로 History 길이를 조절하지 못하기 때문에, 현실적인 구현 가능성과는 거리가 멀다. 본 논문에서 제안될 기법은 각 분기 명령어 별 최적의 History 길이를 동적으로 조절하는 방식으로, 프로파일링을 통해 밝혀낸 최적의 *History*의 사용에서 보다 때로는 더 우수한 성능향상을 보인다.

## Ⅲ. 데이터 종속성에 기반한 History 길이 조절

이 절에서는, 각 분기 명령어 별로 사용되는 History 길이를 동적으로 조절할 수 있는 "데이터 종속성에 기반 한 각 분기별 동적 History 길이 조절 기법"을 소개한다.

### 1. 데이터 종속성과 분기 예측

본 논문에서는 설명의 편의를 위해 그림 3에 제시된 예제 제어 흐름도(Control Flow Diagram)를 기준으로 설명한다. 그림 3은 Basic Block(BB)들로 구성되어 있으며, 각 Basic Block들은 하나의 분기 명령어와 제어의 흐름을 변경시키지 않는 다수의 명령어들로 구성되어 있다. 그림 3의 마지막 Basic Block인 BB10의 if (R4)가 현재 우리가 예측하고자 하는 분기 명령어이다. if (R4)의 Global History는 그림 3에서 보여 주듯이 (...TNNTTN)의 값을 가진다.

한편, 그림 3에 나타나 있는 예제 제어 흐름도의 데이터 종속성(Data Dependency)을 분석해 보면 다음과 같다. 우선, BB10의 분기 명령어 if (R4)의 분기 결과는

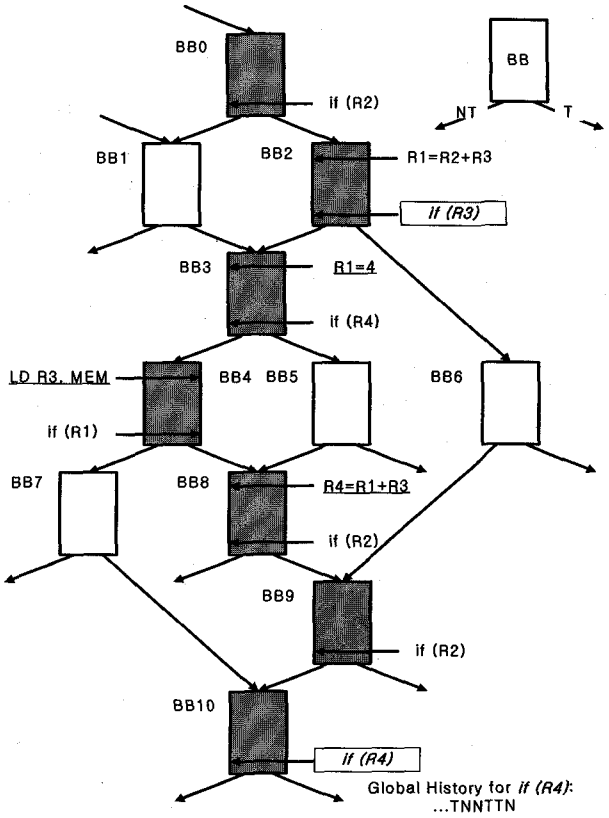


그림 3. 분기 예측을 위한 예제 제어 흐름도
Fig. 3. Control Flow Diagram.

레지스터 R4의 값과 밀접한 관련이 있다는 것을 알 수 있다. 또한 R4의 값은 BB8에서 R1과 R3의 값에 의해 결정되며(R4=R1+R3), 연속해서 R1의 값은 BB3에서 재결정된다(R1=4). 물론 R1의 경우 BB2의 R2와 R3에 의해 다시 재결정되지만 (R1=R2+R3), BB3의 R1=4 연산에 의해 새롭게 갱신되었기 때문에 이전의 데이터 종속성과는 단절된다. 따라서, BB10의 if (R4) 분기 입장에서는 R1의 값이 BB3에서 결정되었다고 할 수 있다. 한편 R4와 연관되어 있는 BB8에 나타난 또 다른 레지스터인 R3은, BB4의 LD R3, MEM 연산에 의해 새롭게 갱신됨을 알 수 있으며, 주어진 그림 3에서는 더 이상 R3과 연관된 데이터 종속성을 발견할 수 없다.

이처럼 BB10의 if (R4)의 분기 예측은 BB3의 "R1=4"와 BB4의 "LD R3, MEM"과 같은 데이터 종속성을 가지는 Basic Block의 해당 레지스터 값에 의해 크게 좌우되며, 결국 BB3과 BB4의 R1=4 와 LD R3, MEM의 수행 여부가 BB10의 if (R4)의 예측에 결정적이라 할 수 있다. 이를 Basic Block들의 관점에서 살펴보면, BB3과 BB4의 수행 여부가 BB10에 나타난 분기 예측에 큰 영향을 미치게 됨을 의미한다. 더 나아가, 이와 같은 BB3의 수행 여부는 BB2의 if (R3) 분기의 분기 예측 결과에, 그리고 BB4의 수행 여부는 BB3에 포함된 if (R4)분기의 분기 예측 결과에 큰 영향을 받는다는 것을 알 수 있다. 하지만 BB4의 수행에 영향을 주는 BB3의 수행 여부는 결국 BB2의 수행 여부에 따라 재결정되기 때문에, 분석 결과 보다 더 멀리 떨어져 있는 Basic Block에 근본적인 영향을 받는다고 할 수 있다. 이상의 분석을 통해, 현재 예측하고자 하는 분기와 강하게 연관되어 있는 레지스터와 분기 명령어들을 파악할 수 있었다.

## 2. 동적인 History 길이 조절

이상에서 언급한 바와 같이, 분기 예측과 관련된 레지스터들 사이의 데이터 종속성을 추적하여, 최종적으로 관련이 있는 레지스터를 파악한 후, 이를 포함하도록 유도하는 분기 명령어의 History를 선택적으로 사용하게 해 주는 알고리즘이 그림 4와 같다.

주어진 알고리즘은, 분기 명령어의 소스 오퍼랜드에 영향을 주는 레지스터를 파악하고, 해당 레지스터에 대한 쓰기 연산을 포함하고 있는 Basic Block으로의 분기를 가능하게 하는 직전 분기 명령어를 찾아내는 방식이다.

그림 5는 주어진 알고리즘을 지원하면서, 레지스터의

```
if (Conditional Branch Instruction)
{
Length_Indicator <= Br_RDT(Reg_src1) ( | Br_RDT(Reg_src2) |...);
All Br_RDT entry << 1;}

else if (Register Writing Instruction)
Br_RDT(Reg_dest) <= 1 ( | Br_RDT(Reg_src1) | Br_RDT(Reg_src2) |
...);

( ...) depends on the number of source operands in the branch instruction.
```

그림 4. History 길이 조절 알고리즘
Fig. 4. History Length Adjustment Algorithm.

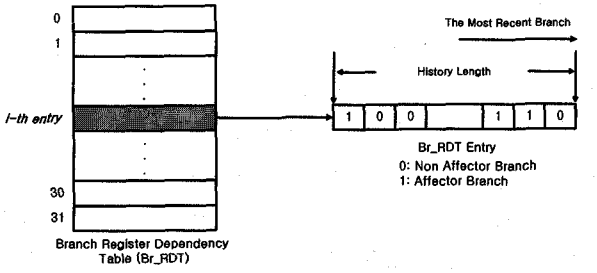


그림 5. 분기 레지스터 종속 테이블(Br_RDT)
Fig. 5. Branch Register Dependency Table (Br_RDT).

상호 종속성을 추적할 수 있는 하드웨어 구조인 분기 레지스터 종속 테이블(Br_RDT, Branch Register Dependency Table)을 보여준다. 해당 테이블은 실제 프로세서가 지원하는 구조 레지스터(Architectural Register) 개수만큼의 엔트리를 가지게 되며, 각 엔트리의 폭은 시스템에서 2의 n승 크기의 PHT와 연관된 최대 History 길이인 n이 된다.

각 Br_RDT의 i-th 엔트리들은 주어진 알고리즘과 연계하여 각 레지스터 Ri의 실행 여부에 영향을 주는 분기 명령어들의 정보를 포함하게 된다. 해당 엔트리의 특정 비트가 1이라는 의미는, 분기 History 상에서 해당 위치의 분기 명령어가 i-th 레지스터 Ri 값의 변경에 영향을 주는 분기임을 의미하며, 0은 그렇지 않는 분기임을 의미한다. 위의 알고리즘으로 BB10의 if (R4)에 대한 Length_Indicator는 5가 된다는 것을 알 수 있으며, 결국 해당 분기의 예측 시에 기존의 주어진 History (...TNNTTN) 길이에서 최근의 5개인 (NNTTN)만을 사용하게 된다.

한편, 분기 예측의 정확도를 높이기 위해서 투기적 분기 History(Speculative Branch History)를 사용하는 방식이 제안 되었다[12][13]. 이는 가장 최근 분기 명령어들의 상호 연관성을 일관되게 반영 할 수 있다는 점에서 유용한 방식이다. 본 논문에서 제안된 기법도 투기적 분기 History가 사용되는 환경에서 구현될 때, 레지스터들 사이에 데이터 종속성을 보다 더 효율적으로 추적 할 수 있다. 이상에서처럼 본 논문에서 제시된 기법은 동적인 기법임과 아울러, 각 분기 명령어 별로 데이터 종속성을 파악할 수있는 "Dynamic & Per-Branch 기법"이라 할 수 있다.

## 3. 분기 예측기의 구현 사례

본 논문에서 제시된 기법을 사용하여 실제 분기 예측기로의 구현 사례를 설명하면 다음과 같다. 본 논문에서는 기본적으로 gshare 분기 예측기를 이용하여 설명한다[14]. gshare 예측기는 분기 예측과 관련된 연구에서 성능향상의 기본 비교 대상으로 자주 사용되는 예측기 가운데 하나이다.

우선, 그림 6은 gshare 예측기의 기본 구조를 보여준다. 그림 6에서 보여 지듯이, 기존의 gshare 기법은 Global Branch History와 분기 명령어의 주소값(Program Counter)을 사용하여, 이를 인덱스 함수인 xor 함수에 대입하고 그 결과를 이용해 분기 예측 테이블(Pattern History Table)에 접근한다. 분기 예측 테이

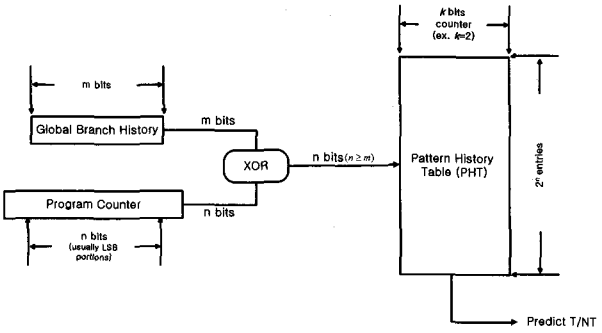


그림 6. gshare 분기 예측기
Fig. 6. gshare Predictor.

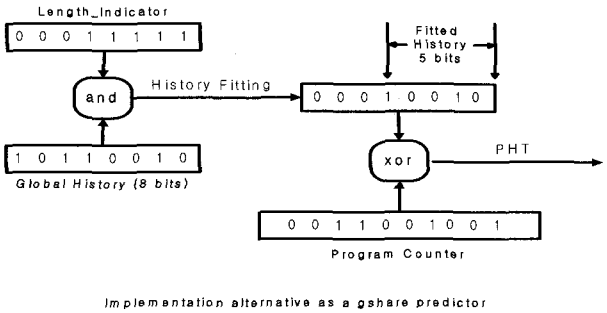


그림 7. History 길이 조절이 가능한 gshare 분기 예측기
Fig. 7. History Length Adjustable gshare Predictor.

블의 각 엔트리들은 k-비트 카운터로 구성이 되어 있으며, 주로 해당 카운터의 상위 비트 값에 따라 분기 예측을 수행한다[15].

그림 7은 History 길이 조절이 가능한 ghsare 예측기의 구현도이다. 그림 7에서와 같이 gshare 예측기가 고정 길이의 8 비트 History를 사용한다고 가정하고, Length_Indicator의 값이 (00011100)이라고 하자. 이 경우, Length_Indicator의 MSB 부분으로부터 첫 번째 1을 기준으로 하위 비트를 모두 1로 셋팅한다. 이와 같이 변형된 Length_Indicator의 값을 Global History와 함께 and 함수에 대입하면, Length_Indicator에서 1의 값을 가지는 비트 수만큼 Global History를 걸러내는(filtering) 효과가 있다. and 함수에서는 한쪽의 입력이 1일 때 출력은 나머지 한쪽의 입력값에 의해 결정되고, 한쪽의 입력이 0일 때는 나머지 한쪽의 입력값이 무시되는 효과가 있기 때문이다. 이렇게 해서 생성된 결과인 (00010010)을 분기 명령어의 주소값 (0011001001)과 xor 함수에 대입한다. xor 함수의 경우는 한쪽이 0의 입력값을 가질 경우, 반대쪽 입력값과 동일한 값이 그대로 출력되어지기 때문에, History Fitting을 통해 걸러진 Fitted History 5 비트 이외의 0으로 채워진 상위 3 비트의 값은 PHT를 인덱싱(indexing)할 때 아무런 현상도 발생시키지 않는다.

## Ⅳ. 성능 평가

본 논문에서 제안된 방식의 성능을 평가하기 위해 다음과 같은 모의실험을 수행하였다. 우선, 각 응용 프로그램 별로 History 길이와 분기 예측 정확도 사이의 상관성을 살피고, 이를 바탕으로 최적의 History 길이에 대한 정보를 파악한다. 그리고 이를 본 논문에서 제안된 방식과 성능상의 비교를 하며, 그 결과를 분석한다

### 1. 실험 환경 및 벤치마크 프로그램

본 논문에서의 모의실험은 구동 기반 시뮬레이터인 SimpleScalar로 진행되었다[16]. 이벤트 구동형 시뮬레이터(Event-Driven Simulator)인 SimpleScalar는 빠른 모의실험과 높은 정확도의 결과를 보장하는 강력한 실험 환경이다. 표 1은 본 논문에서 사용된 실험 환경을 보여준다.

한편, 모의실험에 사용된 벤치마크 프로그램은 SPEC에서 제공되는 CPU 성능 측정 프로그램인 SPEC CINT2000프로그램들로써, 이 가운데 무작위로 선별된 정수형 프로그램들이다[17].

표 1. 모의 실험 인자
Table 1. Simulation Parameter.

| Parameter | Value |
|---|---|
| Fetch Queue | 4 entries |
| Fetch, Decode Width | 4 instructions |
| ROB entries | 16entries |
| LSQ entries | 8entries |
| Functional Units(integer) | 4 ALUs, 1 Mult/Div |
| Functional Units(floating point) | 4 ALUs, 1 Mult/Div |
| Instruction TLB | 64(16 × 4-way)entries, 4K pages, 30 cycle miss |
| Data TLB | 128(32 × 4-way)entries, 4K pages, 30 cycle miss |
| Predictor Style | gshare |
| BTB entries | 2048(512 × 4-way) entries |
| RAS entries | 8 entries |
| Extra Branch Miss-prediction Penalty | 3 cycles |
| L1 I-Cache | 16 KB, direct map, 32B line, 1 cycle |
| L1 D-Cache | 16 KB, 4-way, 32B line, 1 cycle |
| L2 Cache(unified) | 256 KB, 4-way, 64B line, 6 cycles |
| Memory Latency | first_chunk=18 cycles, inter_chunk=2 cycles |

### 2. 실험 결과

본 논문에서 제안된 각 분기별 동적 History 길이 조절 정책의 성능 평가에 앞서, 각 응용 프로그램 별 History의 길이에 따른 분기 예측 실패율(Miss-

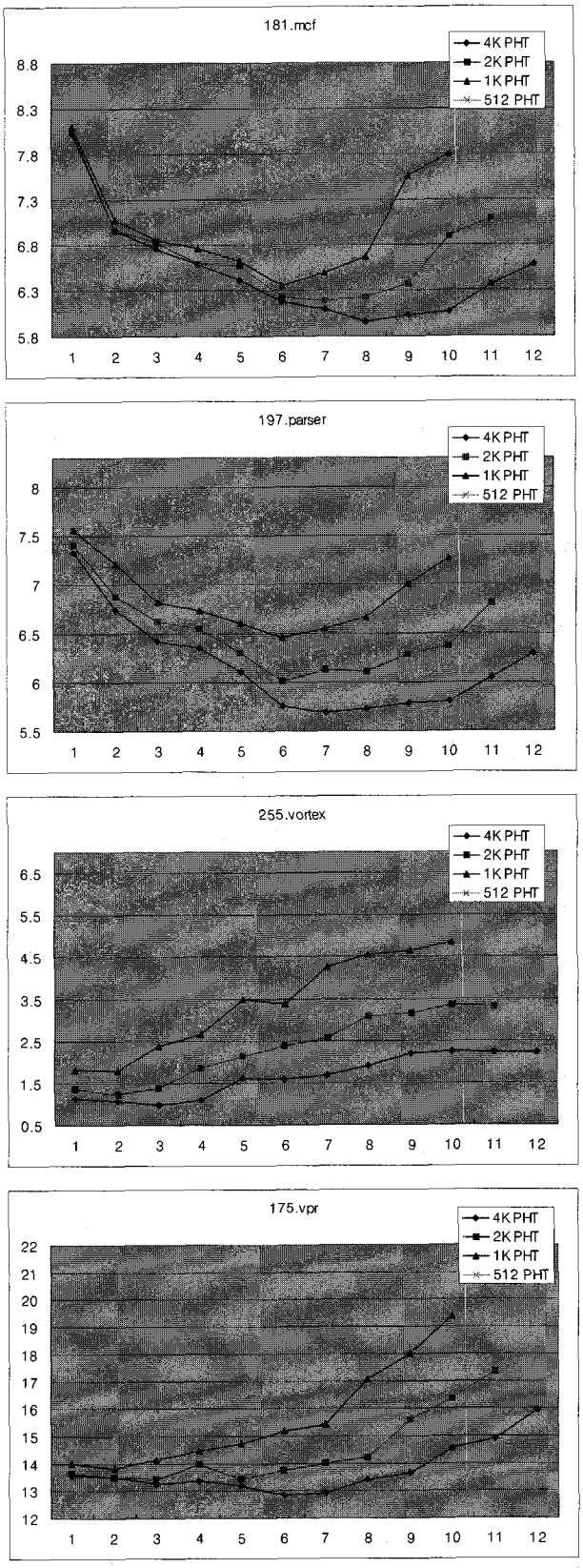


그림 8. History 길이에 따른 분기 예측 실패율(%)
Fig. 8. Misprediction Rate vs. History Length.

Prediction Rate)을 분석하면 다음과 같다. 그림 8은 512(History_Length=9)에서 4K(History_Length =12)크기의 PHT하에서, 각 경우에 대해 사용되는 History 길이 별 분기 예측 실패율(%)을 보여준다. 즉 각각의 PHT 에서, History_Length의 길이를 최소 길이인 1부터 각각의 최대 길이인 9에서 12까지 변화를 주었을 경우의 실험 결과이다. 그림 8에서, X축은 변화되는 History 길이를 나타내며 Y 축은 각각의 경우에 대한 분기 예측 실패율(%)을 나타낸다.

그림 8에서 보여 지듯이, 사용되는 History 길이에 따라, 각각의 응용 프로그램 별로 분기 예측 실패율에 있어서 많은 차이가 남을 알 수 있다. 물론 이와 같은 결과는 응용 프로그램의 특성과 밀접한 관련이 있지만, 실험 결과에서도 알 수 있듯이 모든 응용 프로그램에 있어서 사용되는 History 의 길이에 따라 분기 예측 실패율에 있어서 차이가 크게 나는 것을 알 수 있다. 특히 181.mcf와 197.parser의 경우, 분기 예측 정확도가 사용되는 History 길이에 상당한 영향을 받는다는 것을 알 수 있다.

한편, 각 응용 프로그램 별 512 PHT의 실험 결과에서 보여주듯이, 비교적 작은 크기의 PHT가 사용되는 환경일수록 분기 예측 실패율이 사용되는 History 길이에 보다 더 민감하게 영향을 받는다는 사실을 알 수 있다. 또한, 그림 8의 실험 결과는, 각 응용 프로그램 별 최적의 History가 응용 프로그램 종속적일 수도 있지만, PHT 의 크기와도 연관이 있다는 사실을 보여준다. 가령 181.mcf의 경우, 512 크기의 PHT 에서는 최적의 History 길이가 6이지만, 4K 크기의 PHT에서는 최적의 길이가 8이라는 사실을 확인 할 수 있다. 이 같은 결과는, 최적의 History 길이가 단순하게 응용 프로그램의 영향만을 받는 것이 아니라, 시스템 환경과도 상호 연관되어 있음을 의미한다.

전체적인 실험 결과를 분석하면 다음과 같다. 175.vpr의 경우 분기 예측 실패율에 있어서 사용되는 History 길이에 따라 최대 7.33%의 차이를 보이며 그 외의 응용 프로그램들도 정확도에 있어서 상당한 차이를 보였다. 이 같은 수치는, 분기 예측의 정확도 향상과 관련하여 최근에 발표되는 연구 결과에서 제시된 성능 향상의 정도와 비교해 볼 때 비교적 큰 수치이다. 이는, 새로운 분기 예측기의 제안도 주된 성능 향상의 연구 방법이 될 수 있겠지만, 주어진 분기 예측기 내에서 History 길이를 효율적으로 변화시켜 사용하는 것도 분기 예측과 관련된 성능 향상 기법의 좋은 대안이 될 수 있음을 의미한다.

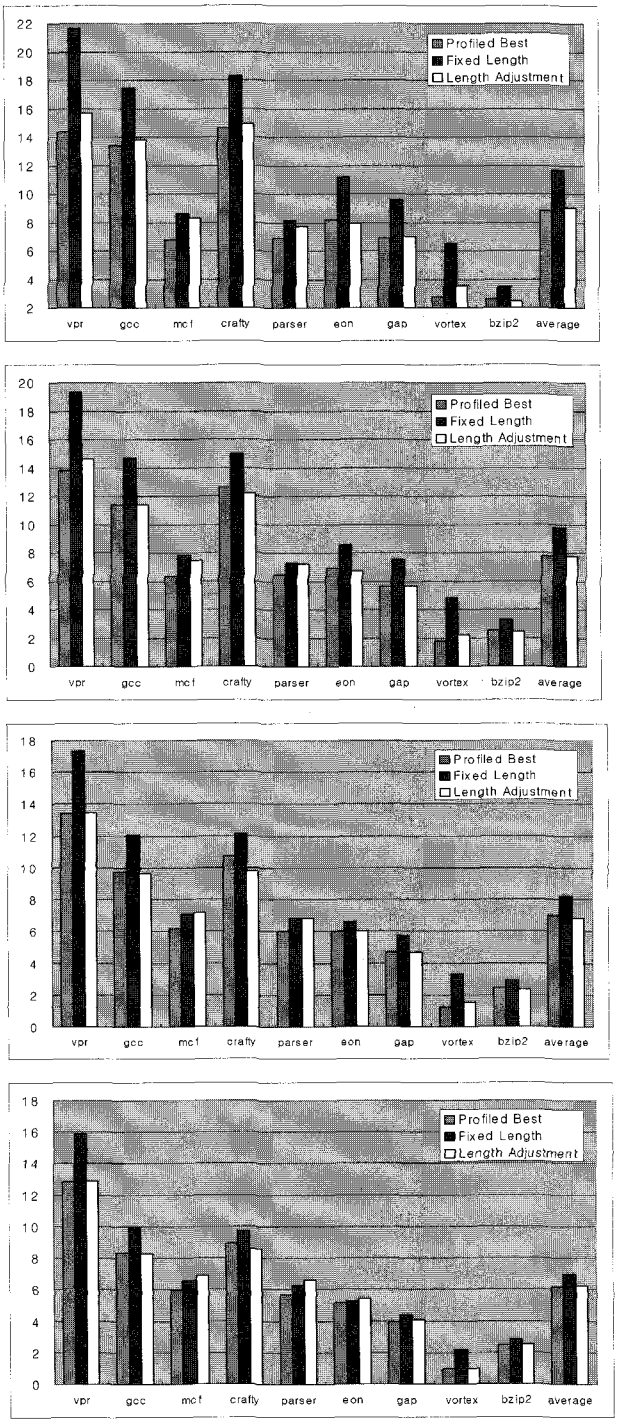


그림 9. Misprediction Rate (%)
Fig. 9. Misprediction rate. (%)

그림 9는 본 논문에서 제안한 방식을 적용한 기법(Length Adjustment)에 대한 분기 예측 실패율을 보여준다. 실험에 있어서 기본적인 비교 대상은, 그림 8에서 프로파일링 결과를 통해 얻어진 최적의 History 길이에 대한 분기 예측 실패율(Profiled Best)과, 주어진 PHT의 크기에 맞게 사용된 기존의 고정 길이 History를 사

용하는 방법에 대한 분기 예측 실패율(Fixed Length)이다. 본 실험에서는 512에서 4096까지 PHT 엔트리 수를 변화시켰으며, 그림 9에 순서대로 이들의 결과가 제시되어 있다. 그림 9에서 보여 지듯이, Length Adjustment 기법이 주어진 모든 응용에 있어서 기존의 Fixed Length 기법보다 성능상 우위에 있다는 것을 알 수 있다. 실험결과, Length Adjustment 기법은 Fixed Length 기법에 비해 최대 5.96% 분기 예측 실패율의 개선을 가져왔다.

한편, 주어진 응용 프로그램에 있어서 사실상의 최적 History 길이라 할 수 있는 Profiled Best의 실험 결과와 비교해서도, Length Adjustment 기법은 각 응용 별로 차이는 있지만 나름대로 우수한 성능을 보여준다. 특히 모든 응용에 있어서 평균 수치(average)를 살펴보면 Profiled Best의 실험 결과와 Length Adjustment의 실험 결과가 대체적으로 근사한 정확도 차이를 보이면서 아주 유사한 결과를 보인다는 것을 알 수 있다.

특히, 186.crafty의 경우는 Length Adjustment 방식이 Profiled Best에 비해 오히려 분기 예측 정확도의 향상을 보여준다. 그 외에도 252.eon 이나 256.bzip2의 경우도 Length Adjustment 기법이 Profiled Best 기법에 비해 오히려 더 우수한 분기 예측 정확도를 나타낸다. 이와 같은 결과는, 주어진 Profiled Best 기법이 나름대로 최적의 방식이기는 하지만, 진정한 의미의 최적 History 길이는 아니라는 것을 의미한다. 그리고 이는, 각 분기 명령어 별로 History 길이를 동적으로 사용하는 기법이 최적의 History 길이에 오히려 더 근접할 수 있다는 사실을 보여준다.

이처럼 본 논문에서 제안된 방식은, 기존의 분기 예측기와 비교 했을 경우 그에 대한 성능 향상의 정도를 비교하는 것 보다, 다양하게 사용 될 수 있는 Branch History 길이 하에서의 새롭게 제안된 방식이 분기 예측 정확도 측면에서 항상 최적의 History 길이에 대한 성능과 유사하거나 오히려 우수한 성능을 나타낼 수 있다는 사실에 의의가 있다. 즉, 어떠한 응용 프로그램이나 어떠한 시스템 환경 (가령, 분기 예측 테이블의 크기) 하에서도 항상 최적의 History 길이에 대한 성능을 제공할 수 있다는 사실이 본 논문의 의의이다.

또한, 주어진 결과에서 중요한 사실은, 512와 1K 크기의 PHT를 사용하는 분기 예측 결과에서 보여주듯이, 해당 경우 Length Adjustment를 사용하는 기법의 분기 예측 정확도가 2배 크기의 PHT를 가지는 Fixed Length 기법에 비해 보다 더 우수하다는 것이다. 그리고 4배 크기의 PHT 결과와 비교하여 약간 열등하다는 것이다. 이는 본 논문에서 제시한 기법이 가지는 추가 하드웨어 복잡도의 적절성을 보여주는 실험결과라 할 수 있다. 즉, 동일 하드웨어 복잡도 내에서는 성능 면에서 본 논문에서 제시된 기법이 기존의 방식보다 더 우수하다는 사실이다. 가령 2 비트 카운터를 사용하며 512개의 엔트리를 가지는 PHT 환경에서 시스템이 32개의 구조 레지스터를 지원한다고 가정하면, 전체 1024 비트(512 entries X 2 비트 counter)에서 288 비트(32 registers X 9 history length)의 추가 복잡도를 가진다. 이는 약 28%의 추가 복잡도(1.28배)로서, 2배 혹은 4배 크기의 PHT가 가지는 하드웨어 요구량과 비교하여 보았을 때, 오히려 성능상의 향상을 이룬 것이다. 게다가 이 같은 추가 복잡도의 비율은 PHT의 크기가 증가할수록 감소한다.

한편, 일반적으로 큰 크기의 PHT를 사용할 경우, 분기 예측의 정확도는 증가한다. 하지만, Jimenez et al.은 대규모 크기의 PHT 사용을 통한 분기 예측 정확도의 추가적 향상이 전체 시스템의 성능 향상과 직결되지 않는다는 사실을 보였다[18]. 그리고 저자들은 이와 같은 사실에 대해 큰 PHT를 접근하는 데서 오는 긴 예측 시간(Prediction Time)을 그 원인으로 꼽았다. 저자들은 논문에서, 2 사이클의 예측 시간이 소요되는 PHT 환경에서 100% 정확도를 가지는 분기 예측기(ideal predictor)의 사용이, 1 사이클의 예측 시간이 소요하는 일반적인 정확도의 분기 예측기(normal predictor)의 사용보다 오히려 전체 수행 시간 측면에서 더 열등하다는 사실을 보였다. 이 같은 사실은 이후의 연구들에 의해 검증되고 인정받은 사실이다.

본 논문에서 제시된 기법은 그림 10의 실험 결과에서도 알 수 있듯이, 2배 크기의 PHT와 비교하여 우월한 결과를 보이고 대략 3배 크기의 PHT와 유사한 결과를

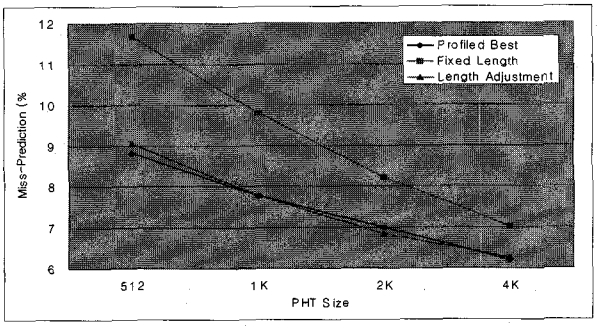


그림 10. Average Misprediction Rate (%)
Fig. 10. Average Misprediction Rate.(%)

보임으로, 추가 하드웨어 복잡도의 적절성을 보여 줄 뿐만 아니라 실제 예측 시간적 측면에서도 보다 더 효율적으로 사용될 수 있음을 보여준다. 이 같은 결과는, 분기 예측 시간의 감소를 위해 최근 제안되고 있는 Overriding Predictor에서, 높은 예측 정확도와 빠른 예측시간이 요구되는 Overrided Predictor로서도 제안된 기법이 우수하게 사용 될 수 있음을 뜻한다[18].

## V. 결 론

분기 명령어에 대한 분기 예측 실패율은 시스템 전체의 성능 향상에 큰 영향을 미친다. 이러한 분기 예측 실패율은 전체 프로세서의 성능향상에 있어서 단일 요소로는 가장 큰 성능 제약 요소 가운데 하나로 알려져 있다. 본 논문에서는 분기 예측의 정확도를 높이기 위한 방법의 하나로, 각 분기 명령어 별로 사용되는 History의 길이를 동적으로 조절할 수 있는 "각 분기별 동적 History 길이 조절 기법"을 소개 하였다. 제안된 기법은, 분기 예측에 있어서 관련된 레지스터들 사이의 데이터 종속성을 추적하여, 최종적으로 관련이 있는 레지스터를 포함하도록 유도하는 분기를 파악한다. 그리고 그 결과로 얻어진 관련 분기 명령어들의 History만을 사용하게 해 주는 방식이다.

이를 위해 본 논문에서는 데이터의 종속성을 추적할 수 있는 알고리즘을 소개하고, 이를 지원하는 하드웨어 모듈을 제시하였다. 그리고 제안된 기법을 실제 gshare 예측기에 적용한 구현 사례를 소개하였다. 실험 결과 제안된 방식은, 기존의 고정 길이를 사용하는 방식에 비해 분기 예측 정확도에 있어서 최대 5.96%의 성능 향상을 가져왔다. 한편 프로파일 방식을 통해 알아낸 최적의 History 길이에 대한 실험 결과와 비교해서도 오히려 정확도 향상을 가져왔으며, 항상 최악의 경우를 피하면서 최적의 경우와 유사한 성능을 제공함을 확인하였다.

## 참 고 문 헌


[1] E. Sprangle and D. Carmean. "Increasing processor performance by implementing deeper pipelines". In Proc. 29th Int'l Symp. on Computer Architecture, pp. 25-34, 2002.

[2] T.-Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," In Proc. of the 19th ISCA, pp. 124-134, May, 1992.

[3] J. W. Kwak, J.-H. Kim, and C. S. Jhon, "The Impact of Branch Direction History combined with Global Branch History in Branch Prediction", IEICE Transactions on Information and System, Vol. E88-D, No. 7, pp. 1754-1758, July 2005.

[4] K. Skadron, M. Martonosi, and D. Clark. "Speculative updates of local and global branch history: A quantitative analysis", JILP, vol. 2, Jan. 2000.

[5] P. Michaud, A. Seznec and R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors", 24th Intl. Symp. on Computer Architecture, pp. 292-303, 1997.

[6] A. Seznec and P. Michaud. De-aliased Hybrid Branch Predictors. Technical Report No. 3618, Institut National de Recherche en Informatique et n Automatique (INRIA), February 1999.

[7] M. Evers, S. J. Patel, R. S. Chapell, and Y. N. Patt, "An analysis of correlation and predictability: What makes two-level branch predictors work", In Proceedings of the 25th Annual Intl. Symposium on Computer Architecture, pages 52-61, June 1998.

[8] T. Juan, S. Sanjeevan, and J. J. Navarro, "Dynamic history length fitting: A third level of adaptivity for branch prediction", In Proc. 25th Int'l Symp. on Computer Architecture, pages 155-166, 1998.

[9] M.-D. Tarlescu, K. B. Theobald, and G. R. Gao, "Elastic history buffer: A low-cost method to improve branch prediction accuracy", In Proc. Int'l Conf. on Computer Design, pages 82-87, 1997.

[10] J. Stark, M. Evers, and Y. N. Patt, "Variable length path branch prediction", In Proc. 8th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, pages 170-179, 1998.

[11] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeid'es. "Design tradeoffs for the ev8 branch predictor", In Proc. of the 29th ISCA, pp. 295-306, May 2002.

[12] A. R. Talcott,W. Yamamoto,M. J. Serrano, R. C.Wood, and M. Nemirovsky, "The impact of unresolved branches on branch prediction scheme performance," in Proceedings of the 21st Annual International Symposium on Computer Architecture, pp. 12-21, Apr. 1994.

[13] E. Hao, P.-Y. Chang, and Y. Patt, "The effect of speculatively updating branch history on

branch prediction accuracy, revisited," in Proceedings of the 27th Annual International Symposium on Microarchitecture, pp. 228-232, Nov. 1994.
[14] McFarling, S., "Combining branch predictors. Tech. Rep. TN-36m", Digital Western Research Lab., June, 1993
[15] A. R. Talcott, M. Nemirovsky, and R. C. Wood, "The Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance", International Conference on Parallel Architectures and Compilation Techniques, 1995.
[16] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future micro-processors: the SimpleScalar tool set", Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., 1997
[17] SPEC CPU2000 Benchmarks, http://www.specbench.org
[18] D. A. Jimenez, S. W. Keckler, and C. Lin, "The impact of delay on the design of branch predictors", In Proc. 33rd Int'l Symp. on Microarchitecture, pp. 67-76, 2000.

## 저 자 소 개

곽 종 욱(정회원)
1999년 경북대학교 컴퓨터공학과 학사
2001년 서울대학교 컴퓨터공학과 석사
2006년 서울대학교 전기·컴퓨터 공학부 박사
2006년~현재 삼성전자반도체, SOC 연구소, Processor Architecture Lab., 책임연구원
<주관심분야 : Mobile Multimedia SOC 설계, 컴퓨터 구조, 고성능 병렬 처리, 저전력 내장형 시스템>

전 주 식(정회원)
1974년 서울대학교 수공학과 학사
1976년 한국과학기술원 전산학 석사
1982년 University of Utah 전산학 박사
1982년~1984년 University of Iowa 연구원
1985년~현재 서울대학교 전기·컴퓨터공학부 교수
<주관심분야 : VLSI, CAD, 병렬 컴퓨터 구조>

장 성 태(정회원)
1986년 2월 서울대학교 전자계산기공학과 공학사.
1988년 2월 서울대학교 대학원 컴퓨터공학과 석사.
1994년 2월 서울대학교 대학원 컴퓨터공학과 박사.
1994년 3월~현재 수원대학교 정보공학대학 컴퓨터학과 교수
<주관심분야 : 다중 프로세서 시스템, 컴퓨터 구조, 병렬 처리, 캐쉬 구조, 메모리 모델>