오퍼랜드 참조 예측 캐쉬(ORPC)를 활용한 오퍼랜드 페치의 성능 개선

김 흥 준† · 조 경 산††

요 약

본 논문에서는 오퍼랜드 참조 지연과 자료 캐쉬에 대한 대역폭 요구를 줄이기 위하여, 명령어 페치 단계에서 오퍼랜드의 값과 주소 변환 정보를 예측하고 초기에 예측의 정확성을 검증하여 예측 실패에 의한 성능 손실을 최소화할 수 있는 오퍼랜드 참조 예측 캐쉬(ORPC) 구조를 제안하였다. 제안된 ORPC의 세 가지 운영 구조(ORPC1, ORPC2, ORPC3)에 의한 예측의 정확도와 성능 개선은 6개의 벤치마크 프로그램의 trace-driven 시뮬레이션을 통해 분석되었다. 512항목의 ORPC2, ORPC3은 평균적으로 오퍼랜드 적재 참조의 45.3%에 대해 정확한 오퍼랜드를 예측하여 오퍼랜드 적재 시간 및 자료 캐쉬의 대역폭 요구를 감소시키며, 또한 ORPC3은 전체 오퍼랜드 참조에 대해 98.1%의 주소 변환 정보를 제공하여 자료 TLB의 기능을 대신한다.

Performance Improvement of Operand Fetching with the Operand Reference Prediction Cache(ORPC)

HeungJun Kim† · Kyungsan Cho††

ABSTRACT

To provide performance gains by reducing the operand referencing latency and data cache bandwidth requirements, we present an operand reference prediction cache (ORPC) which predicts operand value and address translation during the instruction fetch stage. The prediction is verified in the early stage, and thus it minimizes the performance penalty caused by the misprediction. Through the trace-driven simulation of six benchmark programs, the performance improvement by proposed three ORPC structures (ORPC1, ORPC2, ORPC3)is analysed and validated.

1. 서 론

프로세서의 구조 변화 및 성능 향상과 응용 프로그램의 작업부하적 특성 변화에 따라 메모리 시스템의 구조 및 성능에 대한 요구가 컴퓨터 시스템의 설계에 커다란 영향을 미치고 있다. 프로세서의 클록 주기 및 명령어 처리 능력과 주메모리 사이의 성능적 차이는 이미 위험적 수위를 넘게되었으며, 컴퓨터 시스템의 성능은 메모리 참조의 지연에 의해 큰 영향을 받게된다[6]. 또한, 고성능 마이크로프로세서 구현에 사용되는 수퍼스칼라 구조에서는 각 사이클당 하나 이상의 명령어를 발생(issue)하여 ILP(instruction level parallelism)를 증가시키고 있다[8]. 따라서, 메모리 시스템은 참조 시간의 요구를 만족시키는 이외에도, 수퍼스칼라 프로

※ 이 논문은 1997년도 교내 연구비 지원에 의하여 연구되었음
† 종신회원 : 단국대학교 전산통계학과 박사과정
†† 종신회원 : 단국대학교 전산통계학과 부교수
논문접수 : 1998년 3월 17일, 심사완료 : 1998년 5월 11일

세서가 요구하는 높은 대역폭을 제공하도록 설계되어야 한다.

명령어 처리 시에 요구되는 메모리 참조 시간을 줄이고 대역폭을 증가시키기 위해서 참조의 공간적 및 시간적 지역성을 활용하는 계층적 메모리 구조가 사용되어왔다. 또한, 계층적 메모리 구조만으로는 쉽게 해결되지 않는 참조의 지연과 캐쉬의 대역폭 요구 문제를 지원하기 위해 계층적 메모리 구조에 더해지는 대표적 기법에 예측 기법이 있다.

예측 기법은 다음 순서에 참조되리라 예상되는 명령어 또는 자료 및 그 정보가 저장된 주소를 미리 예측하여 메모리 참조를 보다 신속히 처리하도록 할 수 있는 방법이다. 따라서, 예측 기법은 원하는 정보를 예측할 수 있는 근거를 필요로 하며 이는 이전에 수행된 메모리 참조에 대한 기록(history)이 될 수 있다. 캐쉬 또는 TLB 등은 대표적인 예측 기법의 활용 예이다[4]. 또한, 명령어는 참조의 공간적 지역성이 매우 강하므로, 현재 수행중인 명령어의 다음 저장 주소에 저장된 명령어를 미리 페치해오는 방법 등도 예측 기법을 활용하는 구조이다[2]. 파이프라인 구조의 프로세서에서는 분기 명령어에 의한 성능 저하가 매우 크므로, 명령어 페치 시에 동시에 BTB(이전 분기의 기록과 목적 주소 등이 저장된)에 접근하여 분기 목적지의 명령어 또는 분기 주소를 예측하는 분기 예측도 목적 명령어의 참조에 의한 지연을 줄이는 방법이다[14]. 최근에는 오퍼랜드를 갖는 명령어의 페치 시에 오퍼랜드의 값 또는 주소를 예측하여 오퍼랜드 참조의 시간을 감소하려는 여러 제안이 제시되었으며, 이는 다음 장에서 상세히 기술한다.

본 논문에서는 기존 연구를 분석하여, 예측 기법에 의한 오퍼랜드 참조 과정이 명령어 처리의 다른 과정과 동시에 수행되도록 함으로써 오퍼랜드 참조의 효율성을 극대화 할 수 있는 구조를 제안하고, 이에 의한 성능적 이점을 분석 제시한다. 이러한 목적을 위해 본 논문은 다음과 같이 구성된다. 2장에서 오퍼랜드 참조의 효율성을 위한 관련 연구를 소개하고, 3장에서 예측 기법을 활용한 오퍼랜드 참조 예측 캐쉬(ORPC)의 세 가지 운영 구조를 제안하고, 4장에서 trace-driven 시뮬레이션을 통해 제안된 구조의 다양한 운영에 의한 성능 개선을 분석하고, 5장의 결론으로 마무리한다.

2. 효율적인 오퍼랜드 참조를 위한 관련 연구

메모리 참조의 특성은 다음과 같이 분석될 수 있다.

오퍼랜드 참조나 실행 제어 명령어(분기, 함수 호출 명령어 등)의 페치가 아닌 경우에는 순차적 주소에 저장된 명령어만을 참조하므로, 메모리 참조 주소는 쉽게 예측할 수 있다. 따라서, 메모리 참조의 특성 연구는 실행 제어 명령어와 오퍼랜드의 참조가 주 대상이 되었다.

조건부 분기 명령어를 제외한 실행 제어 명령어는 다음 실행할 명령어의 주소가 명령어 내에 지정되므로 간단히 처리할 수 있다. 하지만, 조건 코드의 값에 의해 목적 주소가 정해지는 조건부 분기 명령어의 목적 주소는 과거 분기의 기록을 이용하여 높은 확률로 예측이 가능하다. 즉, 명령어의 페치 시에 동시에 BTB/BPT(branch target buffer/branch prediction table)을 참조하여 분기의 가능성을 예측하고, 미리 예측된 다음 실행 명령어를 페치하여 참조 지연을 감소시킬 수 있다[14].

오퍼랜드는 명령어와는 다른 참조의 특성을 갖는다. 일단 명령어가 페치되고, 그 명령어로부터 오퍼랜드의 주소를 구한 후에야 오퍼랜드의 참조가 가능하다. 따라서, 명령어 페치부터 오퍼랜드 참조 완료까지의 처리 시간을 단축하는 것이 효율적이다. 대부분 명령어에서 메모리 참조가 가능한 CISC 프로세서와는 달리 RISC 프로세서에서는 메모리 참조가 적재(load) 및 저장(store) 명령어로 제한되므로, 적재 명령어에 대해 명령어를 페치할 때 동시에 오퍼랜드를 페치하거나 또는 오퍼랜드의 주소를 얻을 수 있다면 효율적이라는 다음과 같은 여러 제안이 제시되었다.

[5]는 각 적재 명령어에 대해 직전에 참조된 오퍼랜드 주소와 두 오퍼랜드 주소의 차이(stride)를 각각 별도로 저장하여 그 적재 명령어의 새로운 오퍼랜드 주소를 예측함으로써 주소 계산 과정을 hiding할 수 있는 특수 버퍼 구조인 LTB(load target buffer)를 제안하였다. LTB에는 각 적재 명령어의 연속적인 참조 시에 오퍼랜드 주소가 일정한 차이를 갖는 경우에 이를 이용하여 다음 오퍼랜드 주소를 예측하는 방법으로, 시뮬레이션을 통해 512개 이상의 항목을 갖는 LTB의 주소 예측율은 80%에 달한다고 제시하였다. 이와 유사한 연구로 명령어 페치와 동일한 사이클에 오퍼랜드 주소를 신속히 계산하여 오퍼랜드 참조의 시간을 줄이려는

시도도 있었다[1].

[7]은 메모리 참조의 특성을 분석하여 참조의 공간적 및 시간적 지역성 특성에 덧붙여 오퍼랜드 값의 예측에 활용할 수 있는 참조의 값 지역성(value locality of reference: 저장 위치내의 저장 값을 다시 참조한다는 특성) 을 이용하여, 적재 명령어의 주소로 직접 그 명령어의 오퍼랜드 값을 예측하는 적재 값 예측(load value prediction) 기법을 제시하였다. 시뮬레이션을 통해 직접 매핑된 1K 항목에 대해 전체 오퍼랜드 참조 중에서 50%의 적재값 예측이 가능함을 보이고, 이를 오퍼랜드 참조에 활용하였다. 적재 값 예측과 동일한 개념으로 [11]은 40억 개의 오퍼랜드 참조 주소를 분석하여 실제 참조된 오퍼랜드 주소의 대부분은 10,000개 이하라는 결과를 이용하여, 명령어의 주소로 오퍼랜드 값을 미리 예측할 수 있는 오퍼랜드 선취 캐쉬(OPC: operand prefetch cache)를 제안하였다. 명령어 페치와 동시에 예측이 정확할 확률이 높은 경우에 OPC로부터 그 명령어의 오퍼랜드를 페치하여 실제 오퍼랜드 페치과정이 hiding되는 효과를 갖게된다. CISC 구조를 갖는 x86 프로세서를 CPU로 갖는 PC (윈도우 3.1하에서 운영되는)에 대해 8-way 어소시어티브 매핑의 512항목의 OPC에서는 전체 적재 명령어 중에서 35.59%에 대해 오퍼랜드를 예측하여 95.30%의 예측 정확도(전체 오퍼랜드에 대한 선취 비율은 35.59x0.953=33.91%)로 분석되었다. [11]에서는 RISC 프로세서에 대한 오퍼랜드 예측의 성능 영향을 향후 과제로 제시하였다.

3. 오퍼랜드 참조 예측 캐쉬 구조(ORPC)의 제안

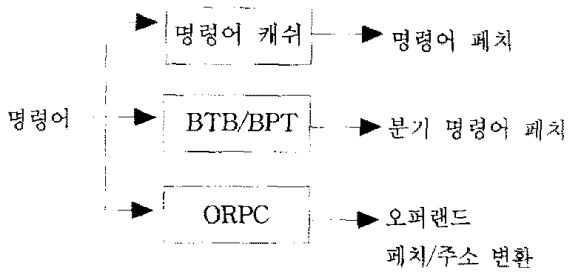
본 장에서는 앞에서 제시된 오퍼랜드 참조 관련 기존 제안들의 성능적 문제점을 개선하여, 오퍼랜드 참조 시간의 감소뿐 아니라 예측의 실패에 의한 성능 손실을 극소화하고 오퍼랜드 캐쉬 및 TLB에 대한 대역폭 요구도 현저히 감소시킬 수 있는 구조를 성능 분석과 함께 제시한다.

기존의 오퍼랜드 예측 기법들은 예측된 오퍼랜드로 명령어의 파이프라인 처리를 진행하는 동시에 자료 캐쉬로부터 실제 오퍼랜드를 페치하여 예측을 검증하는 방법을 사용한다. 따라서, 예측이 실패한 경우에는 파이프라인의 정지에 의한 성능 손실이 비교적 크고, 예측이 성공한 경우에도 예측의 검증으로 인해 자료 캐쉬 및 TLB의 참조 수는 줄지 않는다는 문제가 있다.

기존 연구에서는 명령어 페치 시에 오퍼랜드가 정확히 예측된다면 명령어 페치와 오퍼랜드 페치가 동시에 수행되므로 오퍼랜드 참조를 위한 별도의 시간이 필요하지 않다. 하지만 예측된 오퍼랜드가 올바른가를 검증하기 위하여 유효주소 계산 -> 주소 변환 -> 오퍼랜드 페치(자료 캐쉬)의 과정을 별도로 수행하여 자료 캐쉬로부터 페치된 오퍼랜드로 예측의 정확성을 검증하는 과정이 병행되어야한다.

예측이 실패한 경우에는 오퍼랜드 참조를 위한 처리 과정은 동일하고, 또한 예측된 오퍼랜드로 진행된 파이프라인을 정지해야하는 성능 손실이 추가된다. 예를 들어, 2장에서 소개된 512 항목으로 구성된 OPC에 대한 시뮬레이션한 결과를 적용하면, 예측이 성공하면 오퍼랜드 참조 지연은 적재 명령어당 평균 0.356 x 0.953 x 2 = 0.68 사이클이 감소된다. 하지만 적재 명령어당 0.356 x 0.047 = 0.017, 즉 1.7%의 예측 실패 시에는 2사이클동안 명령어 처리 파이프라인은 정지된다.

본 연구에서는 오퍼랜드 예측 기법의 검증을 자료 캐쉬를 참조하지 않고, 초기에 수행할 수 있도록 주메모리와 일관성을 유지할 수 있는 오퍼랜드 참조 예측 캐쉬(ORPC: operand reference prediction cache)를 제안한다. ORPC는 (그림 1)과 같이 명령어 페치 단계에 명령어 캐쉬 및 BTB와 동시에 명령어 주소로 참조되어 오퍼랜드 값 또는 주소변환의 정보를 제공하며, 주메모리와 일관성이 유지된다.



(그림 1) 명령어 주소에 의한 동시 처리
(Fig. 1) Parallel processing by an instruction address

저장 명령어의 처리 시에 ORPC를 갱신하는 방법으로 주메모리와 일관성이 유지된다면, ORPC로부터 페치된 오퍼랜드 예측의 검증을 위해 자료 캐쉬에서 실제

오퍼랜드 값을 페치 할 필요가 없다. 대신에, 페치된 명령어로부터 계산된 유효 주소를 ORPC 캐쉬에서 페치된 오퍼랜드 주소와 비교하여 일치하면 ORPC로부터 예측된 오퍼랜드가 정확한 것이다. 따라서, 예측의 검증을 위하여 유효 주소 계산 과정만이 필요하고 주소 변환과 자료 캐쉬 참조의 필요성이 없게되므로, TLB와 자료 캐쉬의 참조 수는 예측이 성공한 수만큼 감소하게 된다. 또한, 예측의 검증 시점이 앞당겨지므로, 잘못 예측된 오퍼랜드 값으로 이미 진행된 파이프라인이 정지되는 사이클 수를 감소시키는 장점이 있다. 이와 같은 구조와 운영을 갖는 ORPC를 뒤에서 제안되는 다른 구조와 구별하기 위해 ORPC1라 하자. ORPC1내의 각 항목은 (그림 2)와 같이 구성된다.

명령어 주소	오퍼랜드 주소	오퍼랜드값	유효 비트	카운터 비트	LRU 비트

(그림 2) ORPC1의 항목 구성
(Fig. 2) ORPC1 entry format

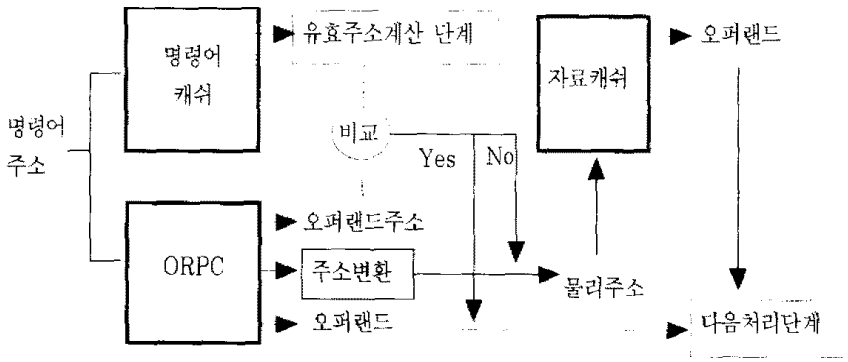
ORPC1에서는 예측의 실패에 의한 성능 저하를 줄이는 것이 중요하므로, 예측 성공의 비율을 높이기 위해 예측의 예상 성공률이 높은 경우에만 예측하도록 카운터를 사용하여 카운터 값이 일정값(2) 이상인 경우만 오퍼랜드를 예측하도록 한다.

ORPC의 또 다른 변형으로, 예측된 오퍼랜드를 ORPC로부터 페치된 직후가 아닌 예측의 검증이 완료된 시점에서 예측이 정확한 오퍼랜드만을 처리하는 운영 구조를 갖는 ORPC2를 제안한다. ORPC2는 예측이 실패하더라도 잘못 예측된 오퍼랜드로는 아무 처리도 하지 않으므로 이에 따른 파이프라인 정지의 손실이 전혀 없다는 장점이 있다. 예측의 검증은 계산된 유효 주소로 가능하므로, ORPC2에서는 예측이 성공한 경우에도 ORPC1에 비해 유효 주소 계산의 시간만큼만 오퍼랜드 참조의 지연이 발생한다. 하지만, 제시된 처리 과정과 같이 유효 주소 계산이 명령 코드의 디코딩과 동시에 수행된다면 처리 지연은 없게된다. ORPC2에서는 전체 적재 명령어에 대한 예측의 비율을 높이기 위해, 카운터의 값이 1이상인 경우에 예측을 하도록 한다. ORPC2의 항목 구성은 ORPC1과 같다.

주메모리와 ORPC와의 일관성 유지를 위해서는 저장 명령의 수행 시에 쓰기 주소와 동일한 오퍼랜드 주소를 갖는 ORPC의 항목을 갱신(또는 무효화)시켜야하며, 이를 위해 ORPC는 명령어 주소와 오퍼랜드 주소에 의해 각각 참조가 가능하도록 구현한다. 이 구조에서, ORPC에 각 오퍼랜드에 대해 가상 주소와 함께 변환된 물리 주소 항목을 추가하면, TLB 대신에 주소 변환 정보의 제공이 가능하다. 또한, ORPC는 명령어 주소에 의한 오퍼랜드의 예측과 주소 변환이 모두 실패한 경우에, 계산된 유효주소에 의한 주소 변환을 위해 사용될 수 있으므로 별도의 자료 TLB의 구현은 필요 없게된다. 이와 같이, ORPC2에 주소 변환 기능을 추가한 구조를 ORPC3라하고, (그림 3)과 같은 항목 구성을 갖는다.

명령어 주소	오퍼랜드 주소(가상)	오퍼랜드 주소(물리)	오퍼랜드 값	유효 비트	카운터 비트	LRU 비트

(그림 3) ORPC3의 항목 구성
(Fig. 3) ORPC3 entry format



(그림 4) ORPC3의 동작
(Fig. 4) Operation of ORPC3

ORPC3에서 적재 명령어의 오퍼랜드 참조 동작은 다음과 같이 요약된다(그림 4 참조).

단계 1) 명령어 페치 단계에 ORPC를 동시에 병렬로 참조하여, 명령어 주소에 해당하는 항목으로부터 오퍼랜드가 유효하면 오퍼랜드와 오퍼랜드 가상주소 및 물리주소를 페치한다.

단계 2) 페치된 명령어로부터 계산된 오퍼랜드 주소를 ORPC의 오퍼랜드 주소(가상)와 비교하여 일치하면 예측된 오퍼랜드를 이용하여 다음 단계의 작업을 수행한다.

단계 3)만약 일치하지 않으면, 계산된 오퍼랜드 주소와 ORPC로부터의 가상 및 물리주소를 이용하여 주소를 변환한다. 변환된 주소로 자료 캐쉬로부터 오퍼랜드를 페치한다.

4. 시뮬레이션 및 성능 분석

4.1 작업 부하 및 시뮬레이터

본 연구에서는 제안된 오퍼랜드 참조 예측 캐쉬의 영향을 분석하기 위해, 컴퓨터 구조의 시뮬레이션을 위해 C++로 작성된 객체 지향 시뮬레이터[13]를 일부 수정하여 trace-driven 시뮬레이션을 수행하였다. 시뮬레이션에 사용된 메모리 참조 주소는 SUN사에서 제공하는 shade를 사용하여 trace 하였다[3][9][10]. 작업부하는 shade를 활용한 trace 프로그램과 benchmark 프로그램들을 다음의 실행 환경을 통해 수집하였다.

커널 구조: Sun4m
응용 구조: Sparc
커널 버전: SunOS 5.5.1

작업부하의 생성을 위해 사용된 여섯 개 벤치마크 프로그램의 특성[12]과 수행 특성은 〈표 1〉 및 〈표 2〉와 같다.

시뮬레이션은 ORPC의 구조(항목의 수, 항목의 구성, 셋의 수, 블록의 수, 매핑 방법, 페이지 크기) 및 관리 방법(교체 방법)과 예측 대상(오퍼랜드 값, 오퍼랜드 주소, 주소 변환 정보)등의 여러 조건에서 분석이 가능하도록 구현하였다.

4.2 결과 분석

예측의 실패에 의한 손실을 줄이기 위해 예측이 성

〈표 1〉 사용된 벤치마크 프로그램
〈Table 1〉 Benchmark descriptions

벤치마크 프로그램	특 성
espresso	PLA 논리 최적화 프로그램
gs	포스트 스크립트 페이지-묘사 언어에 대한 해석 수행 프로그램
make	GNU make 프로그램
gcc	GNU C 컴파일러 버전 2.7.2
compress	자료 압축 프로그램
ls	디렉토리 나열 프로그램

〈표 2〉 벤치마크 프로그램의 수행 특성
〈Table 2〉 Execution characteristics of benchmark programs

베치마크 프로그램	작업 정도	명령 수	명 령 페이지	자 료 페이지	load 비 율	store 비 율
espresso	z5xp1	31,229,837	80	139	0.223	0.043
	mlp4	85,539,946	81	139	0.234	0.040
	cps	547,653,895	80	176	0.223	0.045
	largest	2,428,116,069	81	224	0.220	0.052
gs	small	26,897,998	95	241	0.164	0.054
	large	231,865,158	91	320	0.173	0.062
	manual	740,481,890	100	525	0.183	0.061
make	make	6,534,220	89	166	0.173	0.038
	gs	37,253,459	89	217	0.170	0.027
	perl	79,996,383	89	212	0.210	0.020
compress	compress	1626309734	59	207	0.218	0.087
ls	ls -lR	58,164,823	73	154	0.132	0.046
gcc	gcc -o	1,007,656	54	93	0.199	0.042
				평 균	0.190	0.040

* 명령, 자료 페이지 : 실행중에 참조된 서로 다른 페이지의 수

〈표 3〉 ORPC1의 예측 성공률 및 오퍼랜드 선취 비율
〈Table 3〉 accuracy of prediction and operand prefetching by ORPC1

프로그램	구 분	16	32	64	128	256	512	1024
espresso	예측정확도	0.880	0.901	0.927	0.928	0.918	0.915	0.912
	선취 비율	0.191	0.251	0.397	0.460	0.515	0.553	0.553
compress	예측정확도	0.999	0.999	0.999	0.999	0.998	0.998	0.998
	선취 비율	0.387	0.568	0.577	0.578	0.579	0.579	0.579
gcc	예측정확도	0.945	0.943	0.935	0.915	0.897	0.871	0.868
	선취 비율	0.089	0.144	0.164	0.207	0.233	0.245	0.249
gs	예측정확도	0.996	0.992	0.981	0.970	0.953	0.931	0.918
	선취 비율	0.137	0.148	0.286	0.406	0.462	0.496	0.500
ls	예측정확도	0.825	0.930	0.941	0.931	0.927	0.932	0.928
	선취 비율	0.035	0.147	0.195	0.253	0.300	0.419	0.491
make	예측정확도	0.983	0.894	0.796	0.810	0.825	0.828	0.825
	선취 비율	0.012	0.238	0.258	0.294	0.348	0.388	0.398
예측정확도 평균		0.938	0.943	0.930	0.925	0.920	0.913	0.908
선취비율 평균		0.142	0.249	0.313	0.366	0.406	0.447	0.461

* 예측정확도 : 예측성공 오퍼랜드 수/예측된 오퍼랜드 수
* 선취 비율 : 예측성공 오퍼랜드 수/전체 오퍼랜드 요청 수

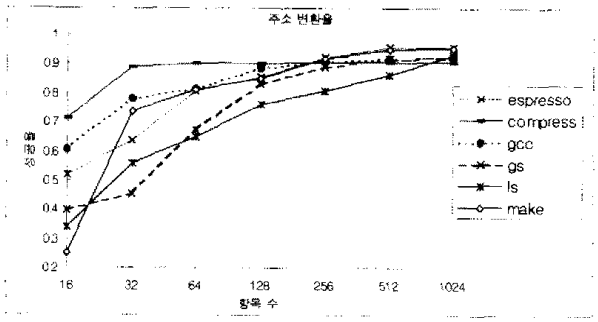
공한 확률이 높은 경우에만 예측하도록 카운터 값이 2 이상인 경우만 예측한 ORPC1의 예측에 대한 성공률과 전체 적재 오퍼랜드 수에 대한 예측 성공 오퍼랜드 수(선취 비율)를 〈표 3〉에 표시하였다. ORPC2에 대한 동일한 분석은 〈표 4〉에 제시된다.

〈표 4〉 ORPC2의 예측 성공률 및 오퍼랜드 선취율
〈Table 4〉 accuracy of prediction and operand prefetching by ORPC2

프로그램	구 분	16	32	64	128	256	512	1024
espresso	예측정확도	0.872	0.894	0.991	0.918	0.906	0.900	0.897
	선취 비율	0.212	0.280	0.428	0.483	0.530	0.559	0.559
compress	예측정확도	0.999	0.999	0.999	0.998	0.998	0.998	0.998
	선취 비율	0.415	0.569	0.577	0.579	0.579	0.579	0.579
gcc	예측정확도	0.941	0.939	0.930	0.899	0.887	0.862	0.859
	선취 비율	0.096	0.193	0.178	0.221	0.241	0.252	0.255
gs	예측정확도	0.986	0.977	0.970	0.960	0.942	0.921	0.908
	선취 비율	0.144	0.160	0.308	0.424	0.475	0.503	0.506
ls	예측정확도	0.830	0.928	0.938	0.927	0.923	0.927	0.922
	선취 비율	0.043	0.170	0.226	0.295	0.343	0.430	0.493
make	예측정확도	0.959	0.892	0.792	0.802	0.816	0.817	0.814
	선취 비율	0.013	0.254	0.277	0.310	0.363	0.397	0.404
예측정확도 평균		0.931	0.938	0.925	0.918	0.912	0.904	0.900
선취비율 평균		0.154	0.271	0.332	0.385	0.422	0.453	0.466

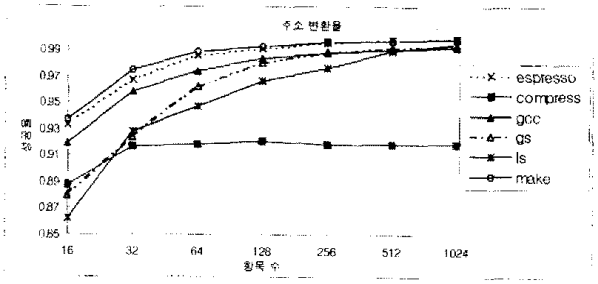
〈표 3〉에서 보듯이 카운터 값이 2이상인 경우만 예측하면, 512 항목의 경우에 예측이 성공할 확률은 최대(compress의 경우) 99.8%, 최소 (make의 경우) 82.8%, 평균은 91.3%으로 오퍼랜드 참조 예측의 필요성을 입증할 수 있다. 〈표 3〉에서 512 항목의 경우 적재 명령어의 평균 44.7%(선취 비율)에 대해 오퍼랜드 참조 지연은 감소하나, 8.7%(예측 실패율)는 파이프라인 수행을 정지시킨다. 〈표 4〉의 ORPC2는 〈표 3〉의 ORPC1에 비해 예측 정확도는 약간 낮지만 선취 비율은 반대로 약간 증가한다. 전체 적재 명령어 중에서 예측이 성공하여 오퍼랜드를 선취될 수 있는 명령어의 비율은 256 및 512 항목 수를 갖는 경우에 42.2%와 45.3% 이고, 이 비율만큼 오퍼랜드 참조 지연이 감소하고 자료 캐쉬 및 자료 TLB의 요구수가 감소하게 된다.

(그림 5)에서 보듯이 적재 명령어의 주소를 인덱스로 하여 ORPC3으로부터 오퍼랜드 주소 변환이 가능한 비율은 512항목에서 평균 91.7%이므로, 오퍼랜드 값 예측이 성공한 45.3%를 뺀 46.4%(91.7-45.3)의 오퍼랜드 참조에 대해 별도의 시간 지연 없이 주소 변환이 가능하다.



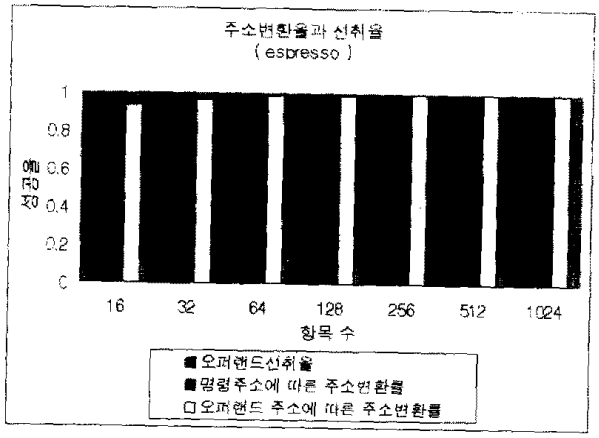
(그림 5) 명령어 주소에 의한 주소 변환 예측의 성공률
(Fig. 5) accuracy of predicting address translation

ORPC3 구조에서 오퍼랜드의 계산된 유효 주소를 인덱스로 하여 ORPC로부터 오퍼랜드 주소 변환 정보를 얻을 수 있는 비율은 (그림 6)과 같다. 512항목인 경우에는 주소 변환 비율이 98.1%이고 compress를 제외하면 99.5% 이상(이는 별도의 시뮬레이션을 통해 얻은 자료 TLB의 히트율과 유사하다)으로 오퍼랜드의 예측이 실패하더라도 ORPC3는 TLB의 기능을 제공한다.



(그림 6) 오퍼랜드 주소에 의한 주소 변환 예측의 성공률
(Fig. 6) accuracy of predicting address translation

(그림 7)에는 espresso에 대해 ORPC3의 항목 수에 따른 오퍼랜드 값 및 주소 변환(명령어 주소 및 오퍼랜드 주소에 의한)에 대한 예측의 정확도를 보여준다. 512항목의 경우에는 예측에 의해 55.9%의 오퍼랜드 참조에 대해 참조 지연과 자료 캐쉬의 참조를 줄일 수 있고, 99.7%까지 주소 변환을 수행하는 것이 가능하다.



(그림 7) espresso의 예측 정확도
(Fig. 7) Prediction accuracy of espresso

ORPC 운영 구조의 선택은 단순한 수치 비교가 아닌 프로세서의 파이프라인 처리의 운영 구조 및 작업부하에 의한 성능 분석, 구현 비용, 요구 대역폭, 파이프라인 수행에 대한 영향 등을 고려하여 선정해야한다. ORPC의 세 가지 운영 구조에 대한 구현 비용은 각 항목의 구성을 비교하면 ORPC1 = ORPC2 < ORPC3의 순서이고, 대역폭 요구는 주소 변환의 요구를 비교하면 ORPC1 = ORPC2 < ORPC3의 순서어다.

5. 결 론

본 논문에서는 계층적 메모리 구조의 자료 캐쉬만으로는 쉽게 지원되지 않는 오퍼랜드 참조의 여러 성능적 문제점을 해결하기 위하여, 예측 기법을 사용하여 적재 명령어의 오퍼랜드 참조의 효율성을 극대화할 수 있는 오퍼랜드 참조 예측 캐쉬의 세 가지 운영 구조(ORPC1, ORPC2, ORPC3)를 제안하고, 다음과 같은 이점을 제시하였다.

1. 오퍼랜드 참조 시간의 감소
2. 예측의 실패로 인한 성능 손실의 극소화
3. 자료 캐쉬의 대역폭 요구의 감소
4. 자료 TLB에 대한 대체 기능 제공

각 운영 구조에 대해 6개 벤치마크 프로그램에 대한 trace-driven 시뮬레이션을 통해 오퍼랜드 및 주소 변환 예측의 정확도가 제시되었으며, 이로부터 ORPC를 이용한 효율적인 오퍼랜드 참조를 입증하였다. 제시된 ORPC 운영 구조의 선택은 작업부하에 의한 성능 분석, 구현 비용, 요구 대역폭, 파이프라인 수행에 대한 영향 등을 고려하여야 한다.

본 연구에서는 각 운영 구조에 따른 예측의 성공률 분석에 중점을 주어, 실제 파이프라인 구조에서의 수행시에 발생하는 영향을 고려하지 않은 낙관적 분석 결과를 제시하였다. 특히, 쓰기 수행에 의한 영향 및 out of order 실행에 의한 실제 파이프라인 처리에서의 영향은 별도의 연구 과제로서 연구되고있다. 멀티프로세서에서의 ORPC의 일관성 유지 및 멀티프로세서에서의 ORPC에 의한 성능 개선의 분석도 향후 연구 과제이다.

참 고 문 헌

[1] T. Austin, "Hardware and Software Mechanisms for Reducing Load Latency," Ph. D. dissertation, University of Wisconsin, 1996.

[2] Tien-Fu Chen and Jean-Loup Baer, "A Performance Study of Software and Hardware Data Prefetching Schemes," ISCA, Vol. 22, No.2, pp.223-232, 1994.

[3] Bob Cmelik and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling," Sigmetrics, ACM, pp.138-137, 1994.

[4] H. Cragon, Memory Systems and Pipelined Processors, Jones and Bartlett Publishers, 1996.

[5] M. Golden and Trevor Mudge, "Hardware Support for Hiding Cache Latency," CSE-TR-152-93, available at www.cs.umich.edu, U. of Michigan, 1993.

[6] Hennessy and Patterson, Computer Architecture - A Quantitative Approach, Morgan Kaufmann, 1996.

[7] M. Lipasti and et al., "Value Locality and Load Value Prediction," Procs. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.138-147, 1996.

[8] James Smith and Gurindar Sohi, "The Microarchitecture of Superscalar Processors," available at http://rouge.engr.wisc.eduu/ece

LR 파서를 위한 효율적인 점진적 파싱

안 희 학†

요 약

본 논문에서는 실제 사용에 있어서 시간과 기억 장소를 상당히 요구하는 기존의 점진적 파싱 알고리즘들을 조사하여, 이들보다 효율적인 점진적 LR 파싱 알고리즘을 제안한다.

문법 기호를 포함하는 확장형 LR 파싱표를 본 논문에서 제안한 점진적 LR 파싱 알고리즘에 적용한다.

여러 문장의 경우에 본 점진적 LR 파싱 알고리즘을 이용하여 파싱 단계와 기억 장소를 감소시켰다. 본 알고리즘은 복잡하고 큰 문법의 경우에 더욱 효과적이다.

An Efficient Incremental Parsing for LR Parsers

Heui-Hak Ahn†

ABSTRACT

In this paper, we review the conventional incremental parsing algorithms which are too expensive in time and memory space to be of practical use, and we propose an incremental LR parsing algorithm which is more efficient than the previous ones.

We apply an extended LR parsing table including grammar symbols to our incremental LR parsing algorithm.

We show that the parsing steps and memory spaces in our incremental LR parsing algorithm are reduced in several sentences. Our algorithm is more effective in the case of complex and large grammars.

1. Introduction

Incremental parsing was abundantly investigated in the late 1970s and early 1980s[2-8]. The most designers of incremental parsers have jointly pursued the two goals of fast parsing and maximal reuse.

A number of methods and algorithms have been proposed for incremental parsing and for the construction of incremental parsers[2-5, 8-9]. In order to perform incremental parsing, the result of a preceding parse sequence must be utilized. The computation complexity of an incremental parsing algorithm largely depends on the data structure used to save parse sequence.

Ghezzi and Mandrioli[2,3] examined the concept of incremental parsing and presented two detailed algorithms which augment a general shift-reduce parser and an LR(0) parser to support incrementality. Celentano [4] derived an incremental parsing algorithm.

The basic algorithm requires the complete parse sequence to be saved for future rean-

※ 이 논문은 1998년도 관동대학교 학술연구비 지원에 의한 결과임.

† 정 회 원 : 관동대학교 전자계산공학과 교수

논문접수 : 1998년 3월 2일, 심사완료 : 1998년 6월 3일

/faculty/ smith_james.html.

[9] Sun microsystem, "shade user's manual," Sun microsystem, http://sw.sun.com/shade.

[10] David L. Weaver and Tom Germond, The SPARC Architecture Manual, Prentice Hall, 1994.

[11] L. Widigen, E. Sowadsky and K. McGrath, "Eliminating Operand Read Latency," Computer Architecture News, Vol.24, No.5, pp. 18-22, 1996.

[12] Benjamin Zorn and Dirk Grunwald, "Evaluating Models of Memory Allocation," ACM Transsactions on Modeling and Computer Simulation, Vol.4, No.1, January, pp.107-131, 1994.

[13] 김 홍준, 안 효범, 조 경산, "확장성을 고려한 계층적 시스템 성능 모델 및 시뮬레이션," 한국시뮬레이션학회 논문지, 제 4권 제 2호, pp.1-16, 1995.

[14] 주 영상, 조 경산, "분기 예측과 이중 경로 전략을 결합한 파이프라인 구조에 관한 연구," 정보 처리 논문지, 제3권 제1호, pp.181-190, 1996.

김 홍 준

1989년 단국대학교 전자계산학과 졸업(학사)

1993년 단국대학교 전산통계학과 졸업(석사)

1993년~현재 단국대학교 전산통계학과 박사과정

관심분야 : 컴퓨터 구조론, 성능평가, 시뮬레이션, 컴퓨터 통신

조 경 산

1979년 서울대학교 전자공학과 졸업 (학사)

1981년 한국과학원 전기 및 전자공학과 졸업(공학석사)

1988년 텍사스대학교(오스틴) 전기전산공학과 졸업(Ph.D.)

1988년~1990년 삼성전자 컴퓨터 부문 책임 연구원

1990년~현재 단국대학교 전산통계학과 부교수

관심분야 : 시스템 구조론 및 성능 분석, 컴퓨터 통신, 시뮬레이션