論文96-33B-2-5

페이지 정렬을 이용한 효과적인 동의어 문제 해결 기법에 관한 연구

(A Study on an Efficient Solution to the Synonym Problem using Page Alignment)

金 濟 成 *, 閔 相 烈 *, 田 尙 勳 **, 安 炳 喆 **,
鄭 德 均 ***, 金 宗 相 *

(Jesung Kim, Sang Lyul Min, Sanghoon Jeon, Byoungchul Ahn, Deog-Kyoon Jeong, and Chong Sang Kim)

요 약

본 논문은 가상 캐쉬에서의 동의어 문제를 효과적으로 해결하기 위한 새로운 기법을 제안한다. 제안된 기법에서는 최소화된 하드웨어가 정확한 실행을 보장해주며 성능 개선을 위해 소프트웨어가 이용된다. 이 기법의 핵심은 U-캐쉬라 명명한 작은 물리 캐쉬를 사용하는 것이다. U-캐쉬에는 정렬되지 않은 가상 페이지에 속하는 캐쉬 블록의 역변환 정보가 저장되는데 여기서 페이지가 정렬되었다 함은 가상 페이지 번호와 물리 페이지 번호의 하위 비트들이 일치함을 의미한다. 페이지 정렬은 간단한 소프트웨어 최적화 기법으로서 U-캐쉬 하드웨어를 효율적으로 사용할 수 있게끔 한다. 이와 같은 하드웨어/소프트웨어 복합 기법은 기존의 하드웨어적 기법에 비해 비용이 적게 드는 동시에 소프트웨어 기법에서의 복잡한 소프트웨어 수정과 이에 기인한 성능 저하를 피하는 장점을 가진다. ATUM 트레이스를 이용한 성능 평가 결과 제안된 기법은 95% 이상의 페이지가 정렬되었을 때 아주 작은 하드웨어를 가지고도 기존의 하드웨어적 기법에 필적하는 성능을 낼 수 있음이 입증되었다.

Abstract

This paper proposes a cost-effective solution to the synonym problem of virtual caches. In the proposed solution, a minimal hardware addition guarantees the correctness whereas the software counterpart helps improve the performance. The key to this proposed solution is an addition of a small physically-indexed cache called U-cache. The U-cache maintains the reverse translation information of the cache blocks that belong to unaligned virtual pages only, where aligned means that the lower bits of the virtual page number match those of the corresponding physical page number. The page alignment is a simple software optimization to improve the performance of the U-cache hardware. With the combination of both hardware and software, the proposed solution reduces the hardware costs and minimizes software modification and performance degradation. Performance evaluation based on ATUM traces shows that a U-cache, with only a few entries, performs almost as well as fully-configured hardware-based solution when more than 95% of the pages are aligned.

* 正會員, 서울大學校 工科大學 컴퓨터工學科
(Dept. of comp. Eng., Seoul Nat'l Univ.)
** 正會員, 嶺南大學校 電子工學科
(Dept. of comp. Eng., Yeungnam Univ.)
*** 正會員, 서울大學校 電子工學科
(Dept. of Elec. Eng., Seoul Nat'l Univ.)

※ 이 논문은 1994년도 한국학술진흥재단의 공모과제 연구비에 의해 연구되었음
接受日字:1995年5月6日, 수정완료일:1996年1月18日

Ⅰ. 서 론

최근 들어 처리기가 고속화됨에 따라 가상 캐쉬(virtual cache)의 중요성이 부각되고 있다[1,2]. 가상 캐쉬는 가상 주소로 직접 접근하는 캐쉬로, 가상-물리 주소 변환에 의한 지연이 없어 빠른 클록 속도를 가지는 고속의 처리기에 매우 유용하다. 이에 반해 물리 주소로 접근하는 물리 캐쉬(physical cache)는 캐쉬 접근에 앞서 가상 주소를 물리 주소로 변환하는 과정이 필요하여 클록 속도를 저하시키거나 파이프라인 단계 수를 증가시킨다

그러나 가상 캐쉬는 물리 캐쉬가 가지고 있지 않은 일관성 문제를 안고 있다[3]. 이는 여러 개의 가상 주소가 하나의 물리 주소를 공유할 경우 가상 캐쉬의 내부에 여러 개의 복사본이 존재할 수 있기 때문이다. 동의어 문제(synonym problem)로 알려진 이 일관성 문제는 하드웨어적으로[4,5,6], 소프트웨어적으로[7,8,9] 여러가지 해결책이 제시되어 있으나 하드웨어적 해결책은 비용이 많이 드는 단점이 있고 소프트웨어적 해결책은 구현이 복잡하며 성능이 저하되는 단점이 있다.

본 논문에서는 하드웨어적 방식과 소프트웨어적 방식을 결합한 새로운 해결책을 제안한다. 이 기법의 핵심은 U-캐쉬라 명명한 작은 크기의 물리 캐쉬를 사용하는 것이다. U-캐쉬는 정렬되지 않은 페이지에 속하는 캐쉬 블록들의 역변환 정보를 관리한다. 여기서 페이지가 정렬되었다 함은 가상 페이지와 물리 페이지의 하위 비트들이 일치함을 말한다. 정렬된 페이지에 속하는 캐쉬 블록은 물리 주소만으로 캐쉬 내에서의 위치를 파악할 수 있으므로 역변환 정보를 따로 저장하지 않는다. 이러한 방식은 캐쉬에 존재하는 모든 블록들에 대한 역변환 정보를 유지하는 기존의 기법들과 대조적이다.

ATUM 트레이스를 이용한 시뮬레이션 결과 본 논문에서 제안하는 기법은 95% 이상의 페이지가 정렬되었을 경우 아주 작은 크기의 U-캐쉬를 가지고도 기존의 하드웨어적 기법에 필적하는 성능을 나타냄이 입증되었다.

이러한 결과는 제안한 기법이 정확성을 보장하는 최소의 하드웨어와 성능을 최적화하는 간단한 소프트웨어를 적절히 결합시킨데 기인한다.

본 논문의 구성은 다음과 같다. 먼저 II장에서는 가상 캐쉬의 동의어 문제와 현재까지 알려진 해결책을 개관한다. 이어 III장에서는 제안한 기법에 대해 상세히 설명한다. IV장에서는 시뮬레이션 결과를 제시하며 끝으로 V장에서는 본 논문의 결론을 도출한다.

Ⅱ. 연구 배경

캐쉬는 메모리의 일부를 저장하는 고속의 버퍼이며, 과거에는 대부분 물리 주소로써 접근되었다. 따라서 가상 주소를 지원하는 시스템에서는 캐쉬를 접근하기 전에 가상-물리 주소 변환을 수행하여야 했다. 주소 변환은 보통 TLB(Translation Lookaside Buffer) 접근에 의해 수행되는데 대개 이러한 TLB 접근에 의해 메모리 접근이 지연되었다. 가상 캐쉬는 이와 같은 물리 캐쉬에서의 지연을 없애고자 고안되었다. 일반적인 물리 캐쉬와 달리 가상 캐쉬는 가상 주소만으로 접근되기 때문에 TLB 접근없이 캐쉬를 접근할 수 있다. 이와 같은 특성은 매우 짧은 클록 시간을 가지는 고속의 처리기에 있어서 특히 유리하다.

그러나 가상 캐쉬는 물리 캐쉬가 가지지 않는 일관성 문제를 안고 있다. 동의어 문제(synonym problem)로 알려진 가상 캐쉬의 일관성 문제는 둘 이상의 가상 주소에 의해 접근되는 메모리 블록의 복사본이 가상 캐쉬 안에 여러 개 존재할 때 발생한다. 예를 들어 변수 A와 변수 B가 물리적 메모리 P를 공유한다면 가상 캐쉬에 블록 P에 대한 두 개의 복사본이 동시에 존재할 수 있다 (그림 1). 이들 복사본 중 한 곳에 쓰기 접근을 하면 두 개의 복사본의 값이 일치하지 않게 되어 메모리 참조의 일관성이 상실된다.

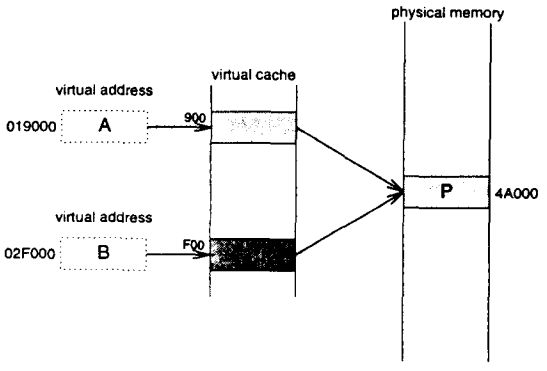

그림 1. 동의어 문제
Fig. 1. Synonym problem.

예를들어 가상 주소 공간이 16 메가바이트(2^{24} 바이

트), 물리 주소 공간이 1 메가바이트(2^{20} 바이트)라면 가상 주소는 6자리 16진수로, 물리 주소는 5자리 16진수로 표현된다. 변수 A의 가상 주소를 019000, 변수 B의 가상 주소를 02F000이라 하고 이 두 변수가 물리 주소 4A000을 공유한다고 하자. 가상 캐쉬의 크기가 64 킬로바이트(2^{16} 바이트), 블록 크기가 16 바이트(2^4 바이트), 연관도(associativity)가 1이라고 가정하면, 물리 주소 4A000은 변수 A의 접근에 의해 가상 캐쉬의 900번째 블록에 적재(load)되며 변수 B의 접근에 의해 F00번째 블록에 적재되어 특별한 처리를 하지 않으면 물리 주소 4A000에 대한 두 개의 복사본이 캐쉬 내부에 존재하게 된다. 이 상황에서 변수 A를 통해 900번째 블록에 어떤 값을 쓴다면 F00번째 블록은 잘못된 값을 가지게 된다. 이후에 변수 B를 통해 F00번째 블록을 읽게 되면 실행 결과가 바르지 않게 된다.

현재까지 동의어 문제의 해결책으로서 여러가지가 제시되었는데 이들은 크게 하드웨어적 기법과 소프트웨어적 기법으로 분류된다. 소프트웨어적 기법은 특별한 하드웨어를 필요로 하지 않고 컴파일러나 운영체제의 도움을 받아 동의어 문제를 해결한다. 가장 단순한 방법은 동의어 사용을 제한하는 것이다. 예를 들어 SPUR 시스템은 페이지 단위의 공유를 허용하지 않고 세그먼트 단위의 공유만을 허용하여 동의어 문제 발생을 방지한다[10]. 이보다 덜 제약적인 방법은 동의어들이 같은 캐쉬 블록에 사상될 경우에 사용을 허용하는 것이다[8]. 같은 블록에 사상되는 동의어는 캐쉬에 적재될 때 이전에 적재되어 있었던 동의어를 방출하여 캐쉬 내에 오직 하나의 복사본만이 존재하게 하기 때문에 동의어 문제의 발생 우려 없이 사용될 수 있다.

가상 메모리 보호 메카니즘을 이용하면 위와 같은 제약이 없이 동의어 문제를 해결할 수 있다[7]. 이러한 기법은 동의어 문제를 발생시킬 수 있는 동작을 예외 처리(exception handling)하여 운영 체제가 적절한 조치를 취하게끔 한다. 그러나 동의어 문제의 해결을 방지하기 위한 캐쉬 배출(flush), 삭제(purge) 등의 작업은 전체적인 성능을 저하시킬 수 있다. Wheeler 등은 이러한 성능 저하를 줄이기 위한 소프트웨어적 최적화 기법을 제시하였다[9].

일반적으로 소프트웨어적 기법은 부가적인 하드웨어 없이 운영체제나 컴파일러의 수정을 통해 구현할 수 있으므로 비용 면에서 유리하다. 그러나 동의어 문제 해결을 위해 운영체제, 컴파일러를 수정하는 것은 복잡한 작업이다. 더욱이 이들 수정은 하드웨어에 의존적이기 때문에 호환성 문제를 일으킬 수 있다.

하드웨어적 기법은 소프트웨어의 도움없이 동의어 문제를 투명하게 해결한다. 이들 기법은 대개 물리 주소를 가상 주소로 역변환하는 캐쉬를 부가하여 가상 캐쉬에 새로운 블록을 적재할 때 이미 캐쉬에 있는 동의어를 검색하여 캐쉬에서 제거한다(동의어 배척: antil-aliasing). 예를 들어 Goodman은 물리 태그 캐쉬라 명명한 다중 처리기 캐쉬의 중복 태그 메모리를 물리 주소로 접근되도록 구성하여 동의어를 배척하게 하는 기법을 제안하였다[6]. Goodman의 물리 태그 캐쉬를 이용한 동의어 배척 기법의 원리를 간단히 설명하면 다음과 같다(그림 2).

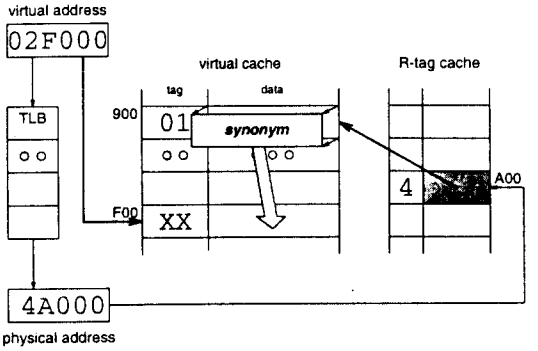

그림 2. 물리 태그 캐쉬를 이용한 방법
Fig. 2. R-tag cache approach.

- ○ 가상 캐쉬에 적재된 모든 블록은 물리 태그 캐쉬에 대응 항을 가진다. 물리 태그 캐쉬의 각 항은 그에 대응하는 가상 캐쉬 블록의 위치를 나타내는 포인터를 가진다.
- ○ 가상 캐쉬에서 접근성공(hit)이 발생하면 처리기는 바로 데이타를 사용할 수 있다.
- ○ 가상 캐쉬에서 접근실패(miss)가 발생하면 TLB 접근을 통해 얻은 물리 주소를 이용하여 물리 태그 캐쉬를 접근한다.
  - 물리 태그 캐쉬의 접근이 성공하면 이는 가상 캐쉬에 또다른 가상 주소로써 접근되는 블록, 즉 동의어가 존재한다는 것을 의미한다. 따라서 물리 태그 캐쉬 접근으로 얻은 포인터가 가리키는 가상 캐쉬 블록, 즉 동의어를 새로운 블록으로 이동시킨다. 그리고 물리 태그 캐쉬의 포인터를 수정하여 이동된 블록을 가리키도록 한다.

- 물리 태그 캐쉬의 접근이 실패하면 이는 가상 캐쉬 내에 또다른 가상 주소로써 접근되는 블록, 즉 동의어가 존재하지 않는다는 것을 의미한다. 따라서 메모리로부터 새로운 블록을 읽어들이고 물리 태그 캐쉬의 한 항을 할당하여 새로 읽어들인 가상 캐쉬 블록을 가리키도록 포인터를 설정한다. 이들 포인터는 가상 캐쉬 내의 유효 블록의 수만큼 필요하므로, 물리 태그 캐쉬를 가상 캐쉬의 블록 수와 동일한 항을 가지는 완전연관 캐쉬로 구현할 경우 항상 빈 항을 찾을 수 있다.

예를 들어 그림 2는 가상 주소 019000과 02F000이 물리 주소 4A000을 공유하는 동의어일 경우 가상 주소 019000에 대한 복사본이 이미 적재되어 있는 상태에서 가상 주소 02F000을 접근하는 상황을 보이고 있다. 이 주소에 대해 접근실패가 발생하였을 때 물리 주소 4A000을 이용하여 물리 태그 캐쉬를 접근하면 동의어가 900번째 블록에 존재한다는 사실을 검출할 수 있다. 따라서 이 블록을 F00번째 블록으로 이동시키므로써 두 개의 복사본이 적재되는 것을 방지할 수 있다.

일반적으로 하드웨어적 기법은 소프트웨어적 기법에서 야기된 캐쉬의 배출, 삭제 등에 의한 성능 저하가 없다는 장점이 있으나 이는 값비싼 하드웨어에 의해서 가능하다. 본 논문에서는 물리 태그 캐쉬의 하드웨어 비용을 줄이기 위한 기법을 제안하고 이를 간단한 소프트웨어적 기법과 결합하여 효율적이고 경제적인 동의어 문제에 대한 해결책을 제시하고자 한다.

Ⅲ. 페이지 정렬과 U-캐쉬를 결합한 동의어 처리 기법

1. 페이지 정렬

앞 장에서 설명하였듯이 물리 태그 캐쉬는 가상 캐쉬의 모든 블록에 대한 포인터를 저장하기 위해 가상 캐쉬의 블록 수 만큼의 항을 가지는 완전 연관 캐쉬로 구성되어야 한다. 하드웨어 비용을 줄이기 위한 한가지 방법은 완전 연관 캐쉬 대신 집합 연관 캐쉬나 직접 사상 캐쉬를 물리 태그 캐쉬로서 사용하는 것이다. 그러나 이러한 경우에는 물리 태그 캐쉬 안에서의 같은 항을 사용하는 블록들이 접근될 때 이들에 대한 포인터를 모두 저장하지 못하는 상황이 발생할 수 있다. 이 상황에서는 부득이 물리 태그 캐쉬의 한 항을 선택해서 교체해야 한다. 이 때 주의할 점은 방출되는 포인터가 가리키는 블록 역시 동의어 문제의 발생을 방지하기 위해 함께 방출되어야 한다는 것이다. 예로써 그림 3은 직접사상 물리 태그 캐쉬를 사용할 때 가상 주소 019000에 대한 블록을 적재하기 위해서 03으로 태그된 블록을 교체(replace)하고 포인터를 저장하기 위하여 0B로 태그된 블록까지 방출하는 상황을 보이고 있다. 만일 여기서 0B로 태그된 블록을 방출하지 않으면 이 블록의 동의어가 캐쉬에 적재될 가능성이 있어 동의어 문제를 야기시킨다. 따라서 하나의 새로운 블록을 적재하기 위해 새로운 블록에 의해 교체될 블록 뿐만 아니라 물리 태그 캐쉬에서 방출되는 포인터가 가리키던 블록까지 총 두 개의 블록을 방출해야 하는 현상이 발생한다. 더욱이 후자의 블록은 방출된 후 다른 새로운 블록으로 채워지기까지 이용되지 않는 상태로 남아 있게 된다. 이와 같은 현상을 연쇄 방출(paired eviction)이라 명명하였는데, 이는 가상 캐쉬 안에 유효 블록이 많을 수록 빈번하게 발생하여 캐쉬 안의 유효 블록의 수를 감소시킨다[11]. 캐쉬의 이용률 저하는 캐쉬의 크기가 실제보다 작은 것과 같은 효과를 가져와 접근실패율을 상승시킨다.

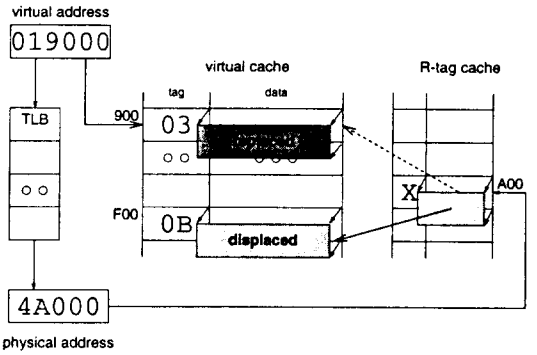

그림 3. 연쇄 방출의 예
Fig. 3. Example of paired eviction.

이러한 상황은 그림 4와 같이 적재할 블록의 포인터가 교체할 블록의 포인터의 위치에 놓이도록 하므로써 회피할 수 있다. 즉 교체할 블록과 적재할 블록의 물리 태그 캐쉬 인덱스가 일치시킨다면 앞에서 설명한 연쇄 방출을 막을 수 있다. 이러한 성질을 이용하면 연쇄 방출에 의한 성능 저하 없이 완전 연관 캐쉬를 구현하기 위한 비용을 줄일 수 있다.

이러한 조건을 수학적으로 표현하기 위해 몇가지 정의를 하겠다. 먼저 가상 주소 공간 V에서의 관계(re-

lation) R_V를 다음과 같이 정의한다.

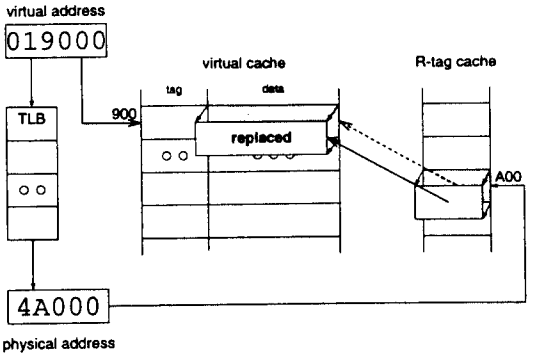

그림 4. 연쇄 방출의 회피
Fig. 4. Avoidance of paired eviction.

$$R_V = \{\langle u, v\rangle \mid u/p \equiv v/p \pmod{c/p}\}$$

(단 c는 캐쉬 크기, p는 페이지 크기)

즉 가상 주소 u, v에 대해 $uR_V v$가 성립함은 u, v의 페이지 번호 $u/p, v/p$의 하위 $\log_2 \frac{c}{p}$ 비트가 일치함을 의미한다. 이와 같이 정의한 관계 R_V는 동치 관계(equivalence relation)이므로 (즉 반사적, 대칭적, 추이적이므로) 가상 주소 공간 V를 분할(partition)하여 동치류(equivalence class)를 생성한다. 이들 동치류의 집합을 V/R_V로 표시하기로 한다. 예를 들어 캐쉬 크기가 32 킬로바이트, 페이지 크기가 4 킬로바이트이면 가상 주소 공간은 8개의 동치류로 분할된다.

가상-물리 사상 함수를 f_x라 할 때 가상 주소 공간에서의 관계 R_P를 다음과 같이 정의한다.

$$R_P = \{\langle u, v\rangle \mid f_x(u/p) \equiv f_x(v/p) \pmod{c/p}\}$$

관계 R_P 역시 V를 분할하여 동치류를 생성한다. 이들 동치류의 집합은 V/R_P로 표시하기로 한다.

이제 교체할 블록과 적재할 블록의 물리 태그 캐쉬 인덱스가 일치할 조건은 다음과 같이 표시된다.

임의의 $u, v \in V$에 대하여 $uR_V v$이면 $uR_P v$이다.

이 조건을 만족시키기 위해 V/R_V에서 V/R_P로의 임의의 전단사 함수 f를 정의하고 이 관계가 만족되도록 가상-물리 사상 함수 f_x를 유지하면 연쇄 방출을 예방할 수 있다. 가능한 함수 f의 종류는 총 $\frac{c}{p}!$ 개이다. 그림 5-(a)는 그 중 한 예를 보이고 있다. 이들 중 가장 간단한 함수는 그림 5-(b)와 같은 항등 함수이다. 이러한 관계를 만족하는 가상-물리 사상 함수 f_x는 다음과 같은 성질을 가진다.

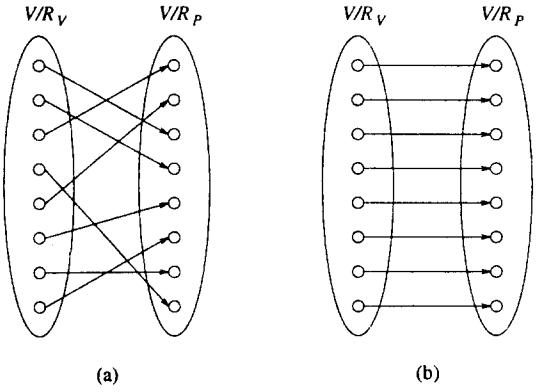

그림 5. V/R_V에서 V/R_P로의 함수 f의 예
Fig. 5. Examples of functions from V/R_V to V/R_P.

임의의 $v \in V$에 대해 $f_x(v/p) \equiv v/p \pmod{c/p}$가 성립된다.

다시 말해 가상 페이지에 물리 페이지를 할당할 때 페이지 번호의 하위 $\log_2 \frac{c}{p}$들이 같도록 하면 위의 관계가 만족된다. 이와 같은 조건이 만족되도록 가상-물리 사상 함수를 유지하는 것을 페이지 정렬이라 정의한다.

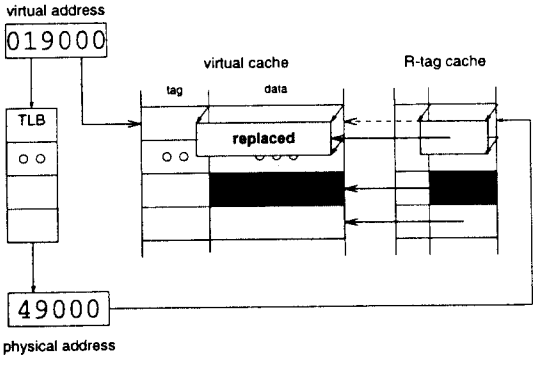

그림 6. 페이지 정렬에 의한 연쇄 방출의 방지
Fig. 6. Prevention of paired eviction by page alignment.

모든 페이지들이 정렬되었다면 물리 태그 캐쉬의 포인터들은 그림 6과 같이 같은 인덱스를 가지는 가상 캐쉬 블록들을 가리키게 될 것이다. 따라서 직접사상 캐쉬를 물리 태그 캐쉬로서 사용하더라도 완전 연관

캐쉬를 사용했을 경우와 동일한 효과를 얻을 수 있다.

페이지 정렬은 여러 연구에서 각기 다른 목적으로 사용되어 왔으며 page coloring[12]이라고 불리우기도 하였다. 예를 들어 Kessler와 Hill은 대용량의 물리 캐쉬에 있어서 자주 사용되는 페이지들이 같은 캐쉬 영역에 사상되었을 때 빈번히 발생하는 접근실패를 줄이기 위하여 이 기법을 사용하였다[13]. 다른 예로 Taylor는 TLB를 단순화시킨 TLB slice를 제안하고 이의 성능을 개선하기 위한 방편으로 페이지 정렬을 이용하였다[12].

그러나 페이지 정렬은 물리 메모리의 연관도를 줄이게 된다. 즉 페이지 할당에 제약을 주지 않으면 물리 메모리는 완전 연관이지만 예를 들어 캐쉬 크기가 64 킬로바이트이고 페이지 크기가 4 킬로바이트이면 페이지 정렬은 가상, 물리 페이지 번호의 하위 4 비트 ($\log_2 \frac{64K}{4K}$)가 같을 것을 요구하기 때문에 연관도는 16이 된다. 하지만 일반적으로 연관도가 8을 넘어서면 성능에 거의 영향을 미치지 않기 때문에 연관도의 감소에 의한 성능 차이는 무시할 수 있다[14,15]. 더욱이 페이지 정렬은 정확한 동작을 위한 필수 조건이 아닌 성능 개선을 위한 사항이므로 페이지 선택을 융통성있게 하여 성능 저하를 최소화할 수 있다. 이러한 특성은 모든 페이지가 정렬되어야 한다는 프로그래밍 상의 제약을 피할 수 있다는 데 의미가 크다. 또한 동의어들이 같은 캐쉬 블록에 할당되도록 가상 주소를 관리하는 복잡한 소프트웨어적 동의어 해결 기법에 비해 소프트웨어 수정 및 검증 작업이 용이하다.

2. U-캐쉬

그림 6을 관찰하면 정렬된 페이지들에 대한 물리 태그 캐쉬의 포인터들은 불필요하게 중복된 정보임을 알 수 있다. 그러므로 경청(snooping)이 필요치 않은 단일 처리기에서라면 물리 태그 캐쉬의 제거를 고려할 수도 있을 것이다. 하지만 정렬되지 않은 페이지들이 존재할 수 있으므로 이에 대한 특별한 처리가 필요하다. 본 연구에서는 정렬되지 않은 페이지의 블록들에 대한 포인터를 U-캐쉬라 명명한 작은 크기의 물리 태그 캐쉬에 저장하여 동의어 문제를 해결하는 기법을 제안한다 (그림 7). 이 기법의 핵심은 페이지들을 정렬하여 가상 캐쉬 블록의 위치를 물리 주소를 통하여 파악하되 정렬되지 않은 페이지에 속하는 블록의 위치는 U-캐쉬로부터 얻게 하는 것이다. 이 기법은 기존의 물리 태그 캐쉬 기법에 비해 적은 비용의 하드웨어로 동의어 문제를 해결한다는 장점을 가진다.

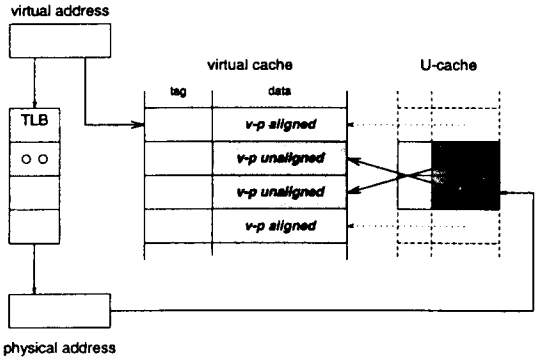

그림 7. U-캐쉬 구조
Fig. 7. U-cache organization.

직접사상 가상 캐쉬에서 U-캐쉬를 이용한 동의어 처리는 다음과 같이 이루어진다.

- 캐쉬의 접근성공 시에는 기존의 물리 태그 캐쉬를 사용할 때와 마찬가지로 동의어 문제에 대한 처리가 필요치 않다.
- 캐쉬의 접근실패 시에는 블록을 적재하기 전에 U-캐쉬를 접근한다. 이 때 접근성공 여부에 따라 다음과 같이 처리된다. 이들 작업은 메모리를 접근하는 동안의 처리기 운휴 시간에 이루어지므로 성능에 영향을 주지 않는다.
  - U-캐쉬의 접근이 성공하면 이는 캐쉬에 (정렬되지 않은 페이지에 속하는) 동의어가 존재함을 의미한다. 동의어의 위치는 U-캐쉬에 저장된 포인터로 알아낼 수 있다. 이 동의어 블록을 요구된 블록으로 이동시켜 캐쉬 접근실패를 처리한다. 그리고 새로운 위치를 가리키도록 포인터를 갱신한다.
  - U-캐쉬의 접근이 실패하면 이는 적재할 블록의 동의어가 존재하지 않거나 존재하더라도 정렬된 페이지에 속함을 의미한다. 만일 적재할 블록이 정렬된 페이지에 속한다면 동의어와 같은 캐쉬 블록에 사상되므로 특별한 처리가 필요치 않다 (직접사상 캐쉬를 가정). 적재할 블록이 정렬되지 않은 페이지에 속한다면 (정렬된 페이지에 속하는) 동의어 블록을 찾아 삭제해야 한다. 그런데 정렬된 페이지에 속하는 블록은 가상 주소와 물리 주소의 하위 비트들이 일치하므로 물리

주소로부터 캐쉬 내에서의 위치를 알아낼 수 있다. 따라서 그 위치의 블록을 캐쉬에서 삭제하여 가능한 동의어를 방출한다. 두 경우에 대해 적재할 블록이 정렬되지 않은 페이지에 속하면 물리 태그 캐쉬를 사용할 때와 마찬가지로 U-캐쉬에서 한 항을 할당하여 포인터를 저장한다. 빈 항이 없는 경우에는 임의로 한 항을 선택하여 교체한다. 적재할 블록이 정렬된 페이지에 속할 때에는 U-캐쉬에 포인터를 저장할 필요가 없다.

본 논문에서 제안하는 기법의 특징은 여타 논문에서 제안한 방식과 달리 페이지들을 정렬시키는 것이 정확한 동작을 위한 필요조건이 아니라는 것이다. 즉 실행의 정확성은 하드웨어에 의해 보장되며 소프트웨어는 하드웨어가 효율적으로 사용되도록 돕는 역할을 한다. 따라서 같은 캐쉬 블록에 사상되지 않은 동의어와 같이 페이지 정렬이 불가능한 경우가 있더라도 실행의 정확성에는 영향이 없다.

3. 집합 연관 캐쉬에서의 U-캐쉬

집합 연관 캐쉬에 있어서도 가상 캐쉬 접근 성공시나 U-캐쉬 접근성공시에는 직접사상 캐쉬에서와 동일하게 처리한다. 그러나 가상 캐쉬와 U-캐쉬 접근이 모두 실패하였을 때는 상황이 좀더 복잡하여 가상 인덱스-물리 태그 캐쉬의 경우에만 효율적인 처리가 가능하다. 가상 인덱스-물리 태그 캐쉬를 가정하면 가상 캐쉬 및 U-캐쉬의 접근실패시 다음과 같이 처리하므로써 동의어 문제를 해결할 수 있다.

직접사상 캐쉬에서와 마찬가지로 U-캐쉬 접근실패는 동의어가 존재하더라도 정렬된 페이지에 속함을 의미한다. 적재할 블록이 정렬된 페이지에 속할 경우에는 가상 캐쉬 접근실패가 동의어의 부재를 함축한다(물리 태그를 사용하므로). 적재할 블록이 정렬되지 않은 페이지에 속할 경우에는 (정렬된 페이지에 속하는) 동의어가 존재할 가능성이 있으므로 이를 검색하여 제거해야 한다. 동의어의 위치는 현재 접근하고자 하는 블록의 물리 주소로부터 알아낼 수 있으며, 존재 여부는 가상 캐쉬의 물리 태그를 검색하여 결정할 수 있다. 즉 접근실패한 블록의 물리 주소를 이용하여 캐쉬를 재접근하므로써 동의어 문제의 발생을 예방할 수 있다.

4. 다중처리기 캐쉬에서의 U-캐쉬

최근에는 고성능 컴퓨터 시스템의 캐쉬를 두 계층으로 구성하여 첫번째 계층을 고속으로 설계하고 두번째 계층은 접근실패를 줄이는데 주안점을 두는 구조를 많이 채택하고 있다. 특히 다중처리기에 있어서는 다계층 포함성(multi-level inclusion property)을 유지시켜 두번째 계층의 캐쉬가 다중 캐쉬 일관성까지 해결하도록 할 수 있다[16].

본 논문에서 제안하는 U-캐쉬는 두번째 계층의 캐쉬를 장착하고 두 캐쉬 사이의 포함성을 유지하여 다중처리기 시스템에 응용할 수 있다. 여기서 두번째 계층의 캐쉬는 V-R 캐쉬 방식[5]과 같이 물리 캐쉬로 구성하여 동의어 문제를 해결하고 버스 접속을 단순화시키는 것이 바람직하다. 그리고 첫번째 계층의 가상 캐쉬 동의어 문제는 단일처리기에서와 같이 U-캐쉬를 이용하여 처리할 수 있다.

이와 같은 가상-물리 2계층 캐쉬 구조에서 캐쉬간의 포함성은 물리 캐쉬에서 블록을 방출할 때 가상 캐쉬에 존재하는 복사본을 함께 방출하므로써 유지할 수 있다[5]. 이를 위하여 물리 캐쉬의 각 블록에 가상 캐쉬 복사본의 존재 여부를 나타내는 한 비트(inclusion bit)를 추가한다. 이 inclusion bit는 가상 캐쉬에 복사본이 적재될 때 1로 설정되고 복사본이 방출될 때 0으로 설정된다. 즉 물리 캐쉬에서 한 블록이 다른 블록에 의해 교체되거나 버스 트랜잭션에 의해 방출될 경우 그 블록의 inclusion bit를 검사하여 1일 경우 가상 캐쉬의 복사본을 함께 방출시킨다. 복사본의 위치는 U-캐쉬에 의해 다음과 같이 결정된다.

- ○ 물리 캐쉬에서 방출되는 블록의 물리 주소로 U-캐쉬를 접근한다.
  - U-캐쉬 접근이 성공하면 이로부터 포인터를 얻어내 위치를 알아낸다.
  - U-캐쉬 접근이 실패하면 이는 주어진 물리 주소의 블록이 정렬되었음을 의미한다. 따라서 물리 주소로부터 가상 캐쉬 안에서의 블록의 위치를 알아낼 수 있다.

이와 같은 2계층 캐쉬에서는 U-캐쉬가 캐쉬내 일관성, 즉 동의어 문제를, 두번째 계층 캐쉬가 캐쉬간 일관성, 즉 다중 캐쉬 일관성을 각각 전담하여 처리하게 된다. 이러한 성질은 계층간의 접속을 단순화시켜 하드웨어 구현을 용이하게 한다.

Ⅳ. 성능 평가

본 연구에서 제안한 U-캐쉬의 성능은 ATUM 트레

이스[17]에 의한 시뮬레이션을 통하여 평가되었다. ATUM 트레이스는 마이크로 코드에 의해 생성된 가상 주소 트레이스로 운영 체제의 참조까지 포함하고 있어 시뮬레이션 결과의 신뢰성을 높여 준다. 본 시뮬레이션에서는 13개의 트레이스를 결합하여 총 480만 참조를 시뮬레이터 입력으로 사용하였다.

각 시뮬레이션에서 페이지 크기는 4 킬로바이트, 블록 크기는 32 바이트로 가정하였다. 특별히 언급하지 않은 경우에는 각 캐쉬들은 직접사상 캐쉬이다.

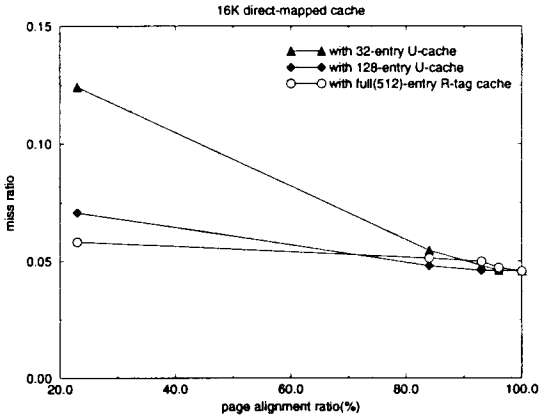

그림 8. U-캐쉬와 물리 태그 캐쉬의 성능 비교
Fig. 8. Performance evaluation of U-cache and R-tag cache.

그림 8은 U-캐쉬를 사용하는 가상 캐쉬와 물리 태그 캐쉬를 사용하는 가상 캐쉬를 시뮬레이션한 결과를 보인다. 이는 정렬된 페이지의 비율을 변화시켜가며 항의 갯수가 32개 및 128개인 U-캐쉬와 항의 갯수가 512개인 물리 태그 캐쉬의 접근실패율을 측정한 것이다. 가장 왼쪽의 점은 페이지 정렬을 고려하지 않았을 때의 접근실패율을 나타낸다. 이 경우 페이지 정렬 비율이 25%인 것은 캐쉬 크기가 페이지 크기의 4배이므로 전체 페이지의 $\frac{1}{4}$ 정도가 우연히 정렬되기 때문이다. 이 지점에서는 U-캐쉬의 성능이 물리 태그 캐쉬에 비해 상당히 떨어진다. 이는 정렬되지 않은 페이지에 속하는 블록이 최대한 U-캐쉬 항의 수만큼만 캐쉬에 적재될 수 있기 때문이다. 예를 들어 U-캐쉬의 항이 32개이면 캐쉬는 정렬되지 않은 페이지로부터 많아야 32개의 블록만이 캐쉬에 적재될 수 있다. 그러나 정렬된 페이지의 비율이 커짐에 따라 U-캐쉬 방식의 접근실패율은 급격히 개선된다. 그리고 모든 페이지가 정렬되었을 경우에는 앞 장에서 설명하였듯이 연쇄 방출이 전혀 발생하지 않기 때문에 접근실패율이 동일하게 나타난다. 한가지 흥미로운 현상은 95% 이상의 페이지가 정렬되면 상대적으로 적은 하드웨어 자원을 사용하는 U-캐쉬의 성능이 더 많은 하드웨어 자원을 필요로 하는 물리 태그 캐쉬 방식을 능가한다는 것이다. 이는 물리 태그 캐쉬가 모든 가상 캐쉬 블록의 포인터를 저장해야 하는데 반해 U-캐쉬 방식에서는 정렬된 블록의 포인터를 저장할 필요가 없기 때문이다.

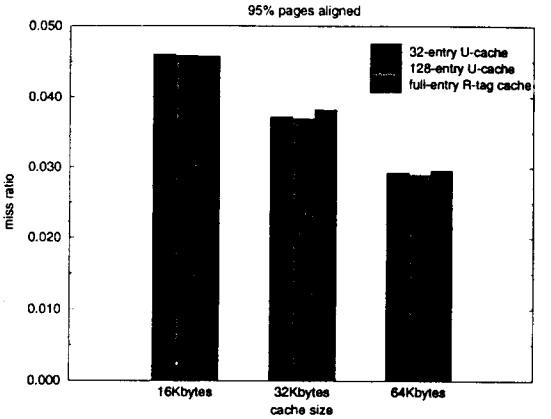

그림 9. U-캐쉬와 물리 태그 캐쉬의 성능 비교 (95% 페이지 정렬)
Fig. 9. Performance evaluation of U-cache and R-tag cache(95% of pages aligned.

그림 9는 95%의 페이지가 정렬되었다고 가정할 경우 16, 32, 64 킬로바이트의 가상 캐쉬에서 U-캐쉬를 사용할 때의 접근실패율을 나타낸다. 이 결과에서도 U-캐쉬는 아주 적은 갯수의 항을 가지고도 물리 태그 캐쉬에 필적하는 성능을 보인다. 더욱이 U-캐쉬의 크기는 가상 캐쉬의 크기에 상관없이 고정할 수 있기 때문에 가상 캐쉬의 크기에 비례하는 수의 항을 요구하는 물리 태그 캐쉬에 비해 더욱 유리하다.

이상의 시뮬레이션 결과에 의해 정렬된 페이지의 비율이 충분히 클 경우 U-캐쉬는 적은 하드웨어 자원을 가지고도 물리 태그 캐쉬에 필적하는 성능을 가진다는 점을 입증하였다.

V. 결 론

최근 들어 처리기가 고속화되면서 접근시간이 짧은 가상 캐쉬의 중요성이 부각되고 있다. 따라서 가상 캐쉬의 동의어 문제에 대한 효과적인 해결 방법을 모색

하는 것이 고속의 처리기 설계에 있어서 또하나의 관건이다.

동의어 문제에 대한 기존의 해결책은 특수한 하드웨어한 이용하는 기법과 소프트웨어를 기반으로 하는 기법으로 분류되는데 이들은 각각 비용면에서, 또 성능면에서 단점을 가지고 있다. 본 논문에서는 하드웨어 기법과 소프트웨어 기법을 결합한 새로운 해결책을 제시하였다. 하드웨어의 측면에서는 기존의 하드웨어를 최소화한 U-캐쉬를 고안하였으며 그 성능을 소프트웨어적으로 최적화하기 위한 페이지 정렬 기법을 아울러 제안하였다.

본 논문에서 제안한 기법의 성능은 ATUM 트레이스를 이용하여 분석하였다. 시뮬레이션 결과 U-캐쉬는 매우 적은 하드웨어 자원을 필요로 하면서도 물리 태그 캐쉬에 필적하는 성능을 가진다는 점이 입증된다. 이러한 가격 대 성능비의 개선은 최적화된 하드웨어와 성능 향상을 위한 소프트웨어를 적절히 결합한 데에서 기인한다.

참 고 문 헌

[1] T. Asprey, G. S. Averill, E. DeLano, R. Mason, B. Weiner, and J. Yetter, "Performance features of the PA7100 microprocessor", *Micro*, vol. 13, no. 3, pp. 22-35, June 1993.

[2] MIPS Computer Systems, *MIPS R4000 microprocessor user's manual*, Integrated Device Technology, 1991.

[3] A. J. Smith, "Cache memories", *ACM Computing Surveys*, vol. 14, no. 3, pp. 473-530, Sept. 1982.

[4] V. Knapp and J.-L. Baer, "Virtually addressed caches for multiprogramming and multiprocessing environments", in *Proceedings of the 18th Annual Hawaii International Conference on System Sciences*, 1985, pp. 477-486.

[5] W.-H. Wang, J.-L. Baer, and H. M. Levy, "Organization and performance of a two-level virtual-real cache hierarchy", in *Proceedings of the 16th Annual International Symposium on Computer Architecture*, 1989, pp. 140-148.

[6] J. R. Goodman, "Coherency for multiprocessor virtual address caches", in *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 1987, pp. 72-81.

[7] C. Chao, M. Mackey, and B. Sears, "Mach on a virtually addressed cache architecture", in *Proceedings of the First Mach USENIX Workshop*, 1990, pp. 31-51.

[8] R. Cheng, "Virtual address cache in Unix", in *Proceedings of the 1987 Summer USENIX Conference*, 1987, pp. 217-224.

[9] B. Wheeler and B. N. Bershad, "Consistency management for virtually indexed caches", in *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992, pp. 124-136.

[10] M. D. Hill et al., "Design decisions in SPUR", *Computer*, vol. 19, no. 11, pp. 8-22, Nov. 1986.

[11] S. L. Min, J. Kim, C. S. Kim, H. Shin, and D.-K. Jeong, "V-P cache: A storage efficient virtual cache organization", *Microprocessors and Microsystems*, vol. 17, no. 9, pp. 537-546, Nov. 1993.

[12] G. Taylor, P. Davies, and M. Farmwald, "The TLB slice - A low-cost high-speed address translation mechanism", in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, 1990, pp. 335-363.

[13] R. E. Kessler and M. D. Hill, "Page placement algorithms for large real-indexed caches", *ACM Transactions on Computer Systems*, vol. 10, no. 4, Nov. 1992.

[14] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[15] T.-C. Chiueh and R. H. Katz, "Eliminating the address translation bottleneck for physical address cache", in *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992, pp. 137-148.

[16] J.-L. Baer and W.-H. Wang, "On the inclusion properties for multi-level cache hierarchies", in *Proceedings of the 15th Annual International Symposium on Computer Architecture*, 1988, pp. 73-80.

[17] A. Agarwal, R. L. Sites, and M. Horowitz, "ATUM: A new technique for capturing address traces using microcode", in *Proceedings of the 13th Annual International Symposium on Computer Architecture*, 1986, pp. 119-127.

저 자 소 개

金 濟 成(正會員)

1991년 서울대학교 컴퓨터공학과(학사). 1993년 서울대학교 컴퓨터공학과(석사). 1993년-현재 서울대학교 컴퓨터공학과 박사과정

閔 相 烈(正會員)

1983년 서울대학교 전자계산기공학과(학사). 1985년 서울대학교 전자계산기공학과(석사). 1989년 워싱턴 주립대학교 전산학(박사). 1989년-1990년 IBM T. J. Watson Research Center 객원 연구원 1990년-1992년 부산대학교 컴퓨터공학과 조교수. 1992년-현재 서울대학교 컴퓨터공학과 조교수

田 尙 勳(正會員)

1994년 영남대학교 전산공학과 학사

安 炳 喆(正會員)

1976년 영남대학교 전자공학과 학사. 1978년 ~ 1984년 국방과학연구소 연구원. 1986년 오레건 주립대학교 전기 및 컴퓨터공학과 석사. 1989년 오레건 주립대학교 전기 및 컴퓨터공학과 박사. 1989년 ~ 1992년 삼성전자 수석 연구원. 1992년 ~ 현재 영남대학교 전산공학과 부교수

鄭 德 均(正會員)

1981년 서울대학교 전자공학과(학사). 1984년 서울대학교 전자공학과(석사). 1989년 U. C. Berkeley 전기 및 컴퓨터공학(석사). 1989년 ~ 1991년 Texas Instruments VLSI Design Lab. 연구원 1991년 ~ 현재 서울대학교 공과대학 전자공학과 조교수

金 宗 相(正會員)

1960년 서울대학교 전자공학과(학사). 1965년 서울대학교 전자공학과(석사). 1975년 서울대학교 전자공학과(박사). 1979년 ~ 현재 서울대학교 공과대학 컴퓨터공학과 교수. 1986년 ~ 1988년 한국정보과학회 회장. 1992년 ~ 현재 서울대학교 컴퓨터신기술연구소 소장. 1993년 ~ 1994년 IEEE Seoul Section 의장. 1995년 IEEE Korea Council 의장