NUMA 시스템에서 소유권에 근거한 원격 캐시 교체 정책

(Keeping-ownership Cache Replacement Policies for Remote Access Caches of NUMA System)

신 숭 현 † 곽 종 욱 †† 장 성 태 ††† 전 주 식 ††††

(Soong Hyun Shin) (Jong Wook Kwak) (Seong Tae Jhang) (Chu Shik Jhon)

요 약 NUMA 시스템은 원격의 메모리에 반복적으로 접근하는 오버헤드를 피하기 위해 지역 노드 내에 원격 캐시를 둔다. 이러한 원격 캐시를 사용하여 원격 메모리로의 접근 지연 시간을 감소시키고 네트워크 상의 트래픽 양을 줄이지 못한다면 다중 프로세서 시스템의 성능 저하는 명백하다. 성능 상의 여러 기준 중에서 메모리 시스템과 관련해서는 캐시 교체 정책에 관한 연구가 계속되었고, 그 중 다중 프로세서 시스템에서의 캐시 교체 정책에 관한 연구도 이어졌다.

본 논문에서는 캐시의 공유 상태에 기반을 둔 교체 정책을 제안한다. 소유권이 없는 캐시 라인을 먼저 교체하고, 이를 통해 소유권이 옮겨지는 오버헤드를 피하여 메모리 지연 시간을 줄인다. 또한 소유권이 없는 캐시 라인에 지나친 피해가 없도록, "MRU를 사용한 소유권 유지 교체 정책(KOM)"과 "참조 비트를 사용한 소유권 유지 교체 정책(KORB)"를 제안하고, 이를 LRU, Pseudo LRU(PLRU)와 비교한다. KOM과 KORB는 PLRU에 비하여 수행 시간에서 25%, 13%씩 각각 향상을 보였다. 특히 KOM은 하드웨어 복잡도가 현저히 낮음에도 불구하고 LRU에 가까운 성능을 나타냈다.

키워드 : NUMA 시스템, 원격 캐시(RAC), 캐시 교체 정책, KOM, KORB, 소유권, 소유권 유지

Abstract NUMA systems have remote access caches(RAC) in each local node to reduce the overhead for repeated remote memory accesses. By this RAC, memory latency and network traffic can be reduced and the performance of the multiprocessor system can be improved. Until now, several cache replacement policies have been proposed in recent years, and there also is cache replacement policy for multiprocessor systems.

In this paper, we propose a cache replacement policy which is based on cache line coherence information. In this policy, the cache line that does not have an ownership is replaced first with respect to cache line that has an ownership. Like this way, the overhead to transfer ownership is avoided and the memory latency can be decreased. We also propose "Keeping-Ownership replacement policy with MRU (KOM)" and "Keeping-Ownership replacement policy with Reference Bit(KORB)" to reduce the frequent replacement penalty of the ownership-lacking cache line. We compare and analyze these with LRU and Pseudo LRU(PLRU). The simulation shows that KOM outperforms the PLRU by 25%, and KORB outperforms the PLRU by 13%. Although the hardware cost of KOM is very small, the performance of KOM is nearly equal to that of the LRU.

Key words : NUMA system, remote access cache(RAC), cache replacement policy, KOM, KORB, ownership, keeping-ownership

† 비 회 원 : 서울대학교 전기컴퓨터공학부
shordan@panda.snu.ac.kr
†† 비 회 원 : 서울대학교 전기컴퓨터공학부
leoniss@panda.snu.ac.kr
††† 종신회원 : 수원대학교 정보공학대학 컴퓨터학과 교수
stjhang@suwon.ac.kr
†††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
csjhon@riact.snu.ac.kr
논문접수 : 2003년 11월 9일
심사완료 : 2004년 5월 4일

1. 서 론

일반적으로 다중 프로세서 시스템 구조는 프로세서와 메모리의 구성 형식과 상호 연결 방식에 따라 두 가지로 나누어볼 수 있다. 프로세서 간 통신의 관점에서 봤을 때, 분산 메모리 구조(Distributed-memory Architecture)와 공유 메모리 구조(Shared-memory Archi-

tecture)로 나뉜다. 분산 메모리 구조는 노드들이 독립적인 프로세서와 메모리를 소유하고 이들은 서로 상호 연결망으로 연결되어, 노드 간 통신은 서로간의 명시적인 메시지 전달을 통해서 이루어지고, 각각의 노드마다 독자적인 운영체제를 수행하는 것이 일반적이다. 반면 공유 메모리 구조에서는 모든 프로세서들이 강결합성으로 연결되어 있고, 물리적으로 메모리를 공유하여 단일 시스템 이미지(Single System Image : SSI)를 제공하고 있다. 모든 프로세서들은 메모리의 모든 영역에 접근 가능하며, 프로세서와 메모리간의 통신은 공유 메모리의 공유변수나 메시지에 의해 이루어진다.

특히 공유 메모리 구조는 단일 주소 공간과 하드웨어적인 일관성을 유지하여 데이타 분할이나 동적 부하 조절 등의 문제 해결에 용이하고, 병렬 컴파일러, 운영체제, 다중 프로그래밍의 지원이 용이하므로 더 융통성 있고 강력한 컴퓨팅 환경을 제공한다.

NUMA(Non-Uniform Memory Access)구조는 공유 메모리 구조 중 하나다. NUMA는 동시 집중 접근을 줄여 확장성과 성능향상을 추구하기 위해 공유 메모리를 각각의 프로세서에 분산시킨다. 따라서 지역 공유 메모리로의 접근은 빠르게 할 수 있지만 원격 공유 메모리로의 접근 시에는 긴 지연시간을 갖게 된다. 이러한 이유로 NUMA 구조에서 각각의 노드는 원격 캐시를 두어 원격 메모리 접근 시 발생하는 지연시간을 숨긴다.

이 논문에서는 NUMA 구조의 새로운 원격 캐시 교체 정책을 소개한다. 소유권이 없는 캐시 라인은 교체 시 원격 노드 접근을 필요로 하지 않고, 따라서 긴 지연시간을 필요로 하는 원격 노드로의 접근을 피할 수 있다는 점에 착안하여 소유권이 없는 캐시 라인을 우선적으로 교체하는 "소유권 유지 캐시 교체 정책"이다. 소유권 없는 캐시 라인을 우선적으로 교체하여 발생하는 편향성을 줄이기 위해 MRU를 사용하거나, 참조 비트를 사용하여 수정한 소유권 유지 교체 정책을 Berkeley 프로토콜 하에서 LRU, PLRU 정책과 성능을 비교한다.

이하 본 논문은 아래와 같이 구성된다. 2장에서 연구 배경으로 NUMA 시스템과 원격 캐시에 관한 설명을 하고, 3장에서 캐시 교체 정책에 관련된 연구를 소개한다. 4장에서 소유권 유지 교체 정책을 설명하고 이를 수정한 MRU를 사용한 소유권 유지 교체 정책과 참조 비트를 사용한 소유권 유지 교체 정책을 각각 설명한다. 덧붙여 소유권 유지 교체 정책의 근거를 제시한다. 5장에서 실험 결과와 분석을 설명하고 6장에서 결론을 내린다.

2. 연구 배경(Background)

2.1 NUMA 시스템

본 논문에서 사용하고 있는 기본적인 시스템 모델은 계층적 버스에 기반 한 NUMA 시스템으로서[1], 시스템 전체 구조는 그림 1과 같다. 시스템은 메모리 주소를 공유하는 다중 프로세서 시스템으로, 각 연산 노드가 단방향(Uni-Direction) 지점 간 링크를 통해서 상호 연결되며, 이를 서로 다른 양 방향으로 구분하여 방향 분리 이중 링(Point-to-Point Dual Ring) 형태를 띈다. 이때 각각의 양 방향 링은 캐시의 일관성(Cache Coherence)을 보장하기 위한 스누핑(Snooping) 방식의 지원 전송 매체로 사용된다. 링을 통해서 전달될 패킷 중 방송용(Broadcasting) 패킷의 경우, 하나의 링은 시계 방향으로 전송을 하고 다른 하나의 링은 반시계 방향으로 전송한다. 특히 전송 데이타의 메모리 주소의 끝자리를 Odd/Even으로 구분하여 각각 서로 다른 해당 링을 할당하여 Broadcast를 하게 구성함으로써, 패킷의 전송 경로를 분산시켜 평균 응답 지연 시간을 줄인다. 단일 전송용(Uni-Casting) 패킷의 경우에는 두 가지 방향 중에서 가까운 방향을 결정하여 해당 링을 사용하여 전송한다. 지점 간 링크로 사용되는 방향 분리 이중 링은 IEEE 표준 SCI 링크로써, 16비트 데이타 폭에 1GByte/Sec(500Mhz)의 전송률을 가진다. 이와 같은 지점 간 링크는 방송 트랜잭션을 지원하므로 논리적으로 버스와 동일한 동작을 수행하게 된다. 또한 이와 같은 버스구조에 기반 한 시스템은 전술한 바와 같이 일반적으로 스누핑[2]에 의한 캐시 일관성 유지 방법을 사용한다.

시스템 버스로 연결된 각각의 연산 노드의 구성은 다음과 같다. 지역 버스로 연결된 4개의 프로세서 모듈, 1차 캐시와 2차 캐시, 원격 캐시(RAC), 지역 메모리, 입출력 시스템, 지역 버스와 노드를 연결하는 노드 제어기(PLC), 노드와 전역 링을 연결하는 링 제어기(DRC)로 구성된다. 지역 버스는 요청 트랜잭션과 응답 트랜잭션이 분리될 수 있는 버스, 즉 분리 트랜잭션(Split Transaction)을 지원하는 버스이며 지역 버스 역시 스누핑 방식에 의한 캐시 일관성 유지 방법을 사용한다. 지역 메모리는 분산형 공유 메모리로 전체 시스템 메모리의 물리 주소 영역 중 일부를 구성하고 있다. 노드 제어기는 원격 노드로의 접근 지연 시간을 단축하기 위해 원격 캐시를 유지하고 있으며, 지역 버스에서 요청이 발생하면 원격 캐시나 전역 링크를 통한 원격 노드로부터의 데이타 제공을 담당한다. 또한 전역 링크에서 발생한 요청에 대해서는 링 제어기가 응답의 책임을 가진다. 이밖에 노드 제어기 및 링 제어기는 각각 지역 버스와 전역 링크에서 발생하는 모든 트랜잭션을 스누핑하여 메모리와 캐시 사이에 일관성 유지를 위한 디렉터리와 태그(Directory & Tag)를 가진다. 입출력 시스템은 프로세서에서 요구한 입출력 디바이스에 대한 읽기 및 쓰기를

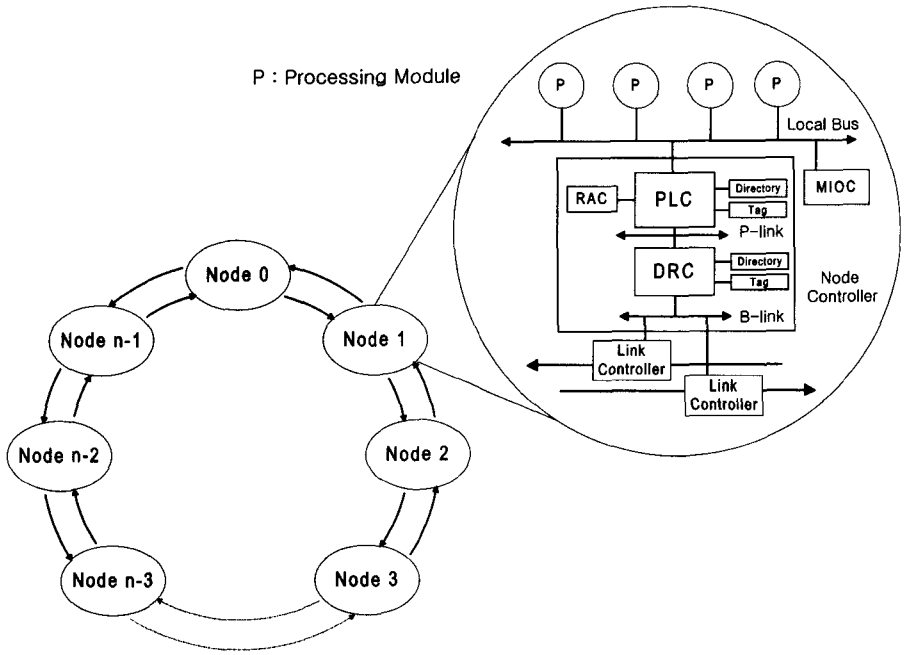


그림 1 방향 분리 이중 링 형태의 NUMA 시스템

수행한다. 모든 입출력 관련 트랜잭션은 지역 버스에 연결된 PCI 버스를 통해 처리하게 된다.

2.2 원격 캐시

그림 1과 같은 전형적인 NUMA 구조, 특히 하드웨어적으로 캐시의 일관성을 유지하는 CC-NUMA 구조에서는 UMA 구조에서 나타나는 메모리 병목현상의 단점을 극복하고자 메모리를 지역적으로 분산시켜 놓았다. 이렇게 분산된 여러 지역 메모리들이 모여서 하나의 전역 주소 공간을 이루게 된다. 그리고 메모리를 지역적으로 분산시켜 놓았다는 점에서는 COMA 구조가 NUMA 구조와 유사한 형태를 보인다. 하지만 COMA 구조에서는 분산된 각 지역 메모리(Attraction Memory : AM)를 캐시와 같이 필요에 따라 동적으로 이용한다. 이는 NUMA 구조에서의 단점인 원격 메모리로부터의 긴 접근 지연 시간을 줄일 수 있다는 점에서 장점이 된다. 하지만 참조 미스(Reference Miss)가 발생했을 경우 각 복사본에 대한 위치 추적과, 최후 복사본의 교체 시 이에 대한 확인 작업등 캐시 일관성을 유지하기가 어렵다. 그 외에도 주로 계층적 디렉터리 구조를 사용한다는 점에서 구현이 복잡하며, 동적 메모리 공간의 관리 및 할당의 오버헤드(Overhead)를 가지는 단점이 있다[3].

이와 같이 메모리가 지역적으로 집중되어 있는 UMA 구조의 단점과 제어상의 복잡도를 가지는 COMA 구조의 단점을 보완하면서, 두 시스템 형태의 장점만을 취한 것이 원격 캐시를 포함하는 NUMA 구조라 할 수 있다. 즉 원격 캐시는 여타의 시스템에 대해서 NUMA 구조가 성능상의 우위를 점하기 위한 필수 구성 요소로서, 노드 외부로 나가는 트랜잭션의 수를 줄이기 위해 원격 메모리 주소 영역에 대해 캐싱하는 역할을 한다. 따라서 본 논문에서도 이와 같은 원격 캐시를 채택하여, 원격 메모리 접근에 대한 응답 지연 시간 감소를 통해 성능상의 향상을 꾀하고자 한다.

2.3 관련 연구(Related Work)

특정 캐시의 성능을 결정하는 여러 요인들 중에서 가장 대표적인 것이 캐시 라인의 효율적인 교체정책(Cache Replacement Policy)이다. 특히 2.2절에서 전술한 바와 같이 NUMA 시스템의 성능 향상을 위한 필수 요소로서의 원격 캐시에 대한 효과적인 캐시 라인 교체 정책이야 말로 시스템 전체 성능 향상에 있어서 중대한 요소라 할 수 있다. 이와 같은 캐시의 교체정책에 대한 관련 연구는 그 숫자를 헤아리기 어려울 정도로 많다. 특히 최근에는 컴퓨터 메모리 시스템 일반적 관점에서의 캐시 교체정책에 대한 연구에서 한 걸음 더 나아가, 특정 시스템에 최적화된 형태의 캐시 교체정책이 제안되고 있는 추세이다. 가령 특정 네트워크와 인터넷 환경에서의 효과적인 웹 캐시(Web Cache)에 대한 논의[4,5], 혹은 모바일 환경이나 내장형 시스템(Embedded System)에서의 효율적인 캐시 교체정책에 대한 연구[6,7] 등이 그 대표적인 예라 할 것이다.

이와 같은 특정 환경이나 시스템에 적합한 캐시의 교

체 정책에 대한 연구 중, 다중 프로세서 시스템 환경 하에서의 캐시의 교체정책 역시 연구되었다[8,9]. 한편 단일 프로세서 시스템의 캐시 교체 정책은 대체로, 캐시의 미스 패널티가 같다는 가정 하에, 캐시 접근에 대한 전체 참조 실패 횟수를 줄이는 것을 목표로 하여왔다[10]. 그러나 메모리 계층 구조가 복잡해짐에 따라, 미스 패널티가 같다는 가정은 설득력을 잃고 있다. 그래서 단순히 반복 사용될 확률을 나타내는 지역성(Locality)만을 고려할 것이 아니라, 성능에 미치는 영향 정도(Criticality)를 함께 고려하는 교체 정책이 제안되었다[11]. 이런 경향은 특히 메모리 계층 구조가 복잡한 다중 프로세서 시스템에서 두드러진다. 그래서 각각의 캐시 라인의 교체 비용을 성능에 영향을 미치는 정도로 수치화하여 사용하는 GreedyDual[12] 정책이나 Cost-Sensitive Replacement Algorithms[9] 등이 제안되었다.

그러나 이와 같은 기존의 캐시 라인의 교체 비용을 고려한 교체 정책은 캐시 라인의 교체 시 들어가는 비용을 알아야 할 필요가 있을 뿐만 아니라, 하드웨어적으로 복잡도를 크게 한다. 이런 이유로, 본 논문에서는 하드웨어적인 복잡도를 낮추면서, 캐시 라인의 상태에 따라 교체 시 처리 비용이 다르다는 점에 착안한 소유권 유지 교체 정책을 제안한다. 특히 기존의 논문에서는 일반적인 공유 메모리 시스템에 기반 한 일차 캐시(L1 Cache) 혹은 이차 캐시(L2 Cache) 상에서의 연구가 진행되었으며, 또한 각각의 캐시들은 MESI 프로토콜을 사용하는 캐시 일관성 정책 하에서의 상태 기반 캐시 교체정책에 대한 연구였다. 하지만 본 논문에서는 특히 NUMA 형태인 이중 링 구조 스누핑 방식의 다중 프로세서 시스템에서, Berkeley 프로토콜을 사용하는 원격 캐시(L3 Cache)의 효과적인 교체정책을 제안하고자 한다.

3. 소유권 유지 교체 정책(Keep-Ownership Replacement Policy)

3.1 원격 캐시의 특성

NUMA 시스템은 원격의 정보를 지역 내 원격 캐시에 유지한다. 원격 메모리로의 접근 지연 시간을 줄이는 것이 첫 번째 목적이다. 일반적으로 프로세서의 수행 시간 중 메모리 접근 시간만을 선택하여 살펴보면 아래 수식과 같다[13].

$$MAT_{AVG} = HT_{L1} + MR_{L1} \times (HT_{L2} + MR_{L2} \times (HT_{RAC} + MR_{RAC} \times MP_{RAC}))$$

MATAVG : 평균 메모리 접근 시간
HT : Hit Time
MR : Miss Ratio
MP : Miss Penalty

그림 2 프로세서의 평균 메모리 접근 시간

메모리 접근 시간을 줄이기 위해서는 MR과 MP를 줄이는 것이 일반적이다. 원격 캐시는 이 중 MP를 낮추는 것이 성능향상에 더 효율적인 방법이다. L1, L2 캐시의 경우 원격 캐시와 달리 적중률이 매우 높으며 특히 L1 캐시는 평균적으로 90%를 넘는 적중률을 보인다. 따라서 원격 캐시에서 서비스되는 비율은 아주 낮다. 원격 캐시에 임의 교체 정책을 적용한 8개의 노드로 구성된 링 기반 CC-NUMA 시스템에서 캐시의 적중률을 살펴본 표 1을 보면 L1, L2캐시를 통틀어 평균 87% 메모리 접근이 적중하고 나머지 13%의 적중 실패 중 18% 정도가 원격 캐시에서 서비스되는 것을 알 수 있다.

이처럼 원격 캐시에서는 기본적으로 상위 레벨의 캐시에서 처리되고 남은 메모리 접근이라 경향성을 찾기 힘들다. 따라서 메모리 경향을 좇아 원격 캐시의 적중률을 높이기도 어렵다. 설령 적중률을 높인다 하여도 적용되는 비중이 일반적인 경우가 아닌 100% - L2까지의 캐시 적중률(A) 즉, 13% 정도로 효과가 적다. 따라서 MR을 낮추는 방법보다는 MP를 줄이는 방법이 더 효과가 있어 보인다.

기존의 연구 중 MP를 줄이는 방법으로 교체 시 더 짧은 시간이 소요되는 캐시 라인을 우선적으로 교체하는 방법이 있다[14]. 하지만 이 경우는 노드 간 거리가 짧은 캐시를 교체함에 있어서 상태에 따른 교체 작업 시간을 간과한 측면이 있다. 같은 거리의 노드에 존재하는 경우에도, 공유 상태(Shared)인 캐시와 소유권을 가진 상태(Modified, Modified_Shared)인 캐시는 교체 시간이 다르다. 링 구조의 NUMA 시스템에서 버클리 프

표 1 L1, L2 캐시의 평균 누적 적중률(%)

	[illegible] 캐시 적중률(A)	[illegible] 적중률(B)	순수 원격 캐시 적중률($\frac{100*(B-A)}{100-A}$)
Barnes Hut	81.07235	85.65746	24.2244019
FFT	92.59988	93.47285	11.7967006
LU	97.5949	98.65375	44.0251965
RADIX	77.81223	81.71081	17.570851
OCEAN	86.15056	81.71081	6.73788976
평균	87.045984	89.315718	17.5214698

표 2 상태별 교체 시간

상태	교체 시간
Invalid	0
Shared	0
Modified	O(n)
Modified_Shared	O(n)

로토콜을 사용할 경우 원격 캐시의 교체 시간은 노드의 수에 비례한다. 이 내용을 표 2에 정리하였다. 공유 상태인 캐시 라인은 침묵 교체(Silent Replacement) 즉, 아무 후속 조치 없이 캐시에서 제거하면 된다. 그러나 소유권을 가지고 있는 경우에는 캐시의 내용을 메모리에 반영하여야 한다. 따라서 추가적인 메모리 접근이 일어나므로 지연시간을 늘리고, 네트워크를 사용하여 성능 저하를 가져온다.

또한, 소유권을 가지고 있는 캐시 라인의 재사용빈도가 소유권이 없는 캐시 라인의 재사용빈도에 비하여 높다. 표 3, 4는 각각 일리노이 MESI를 사용하는 16개 다중 프로세서 시스템에서 LU와 RADIX 수행 중, 1000번의 메모리 접근 당 캐시 라인의 상태가 어떻게 변화하는지를 나타낸다[15]. 원격 캐시는 4-way로 하고 크기는 1MB로 한다. 표 3, 4에서 Invalid 상태는 NP (Not-Present)와 I로 구분된다. NP는 해당 라인이 교체되었음을 의미하고, I는 해당 라인이 무효화(Invalid)를 당했음을 의미한다.

표 3에 따르면, LU의 경우 1000번의 메모리 접근에 대하여 697.129번이 M 상태를 M 상태로 옮기는 메모리 접근으로, 메모리 접근의 70% 정도가 M 상태의 캐시 라인을 읽고 쓰는 작업임을 알 수 있다. 그에 비해 전체 캐시 영역을 소유권이 있는 경우와 없는 경우로

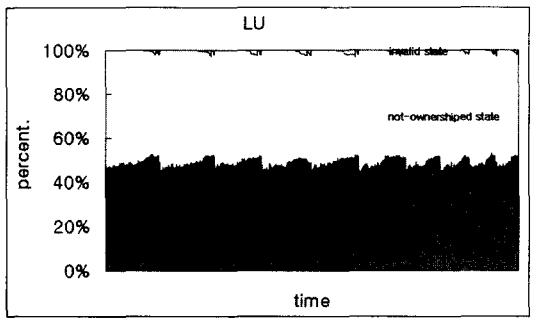


그림 3 LU에서 시간에 따른 원격 캐시 라인 상태의 비율

나누어 살펴본 그림 3에 따르면 전체 캐시 영역에서 소유권 있는 라인은 약 46%를 차지한다. 소유권을 가진 캐시 라인이 소유권이 없는 캐시 라인에 비해 그 수가 적음에도 불구하고 2 배를 넘는 메모리 접근이 이루어지는 것으로 보아 소유권을 가지고 있는 캐시 라인의 재사용빈도가 그렇지 않은 캐시 라인에 비해 더 높다는 것을 알 수 있다.

전술한 바와 같이 소유권이 없는 캐시 라인을 우선적으로 교체할 시에는 MP가 줄어드는 효과가 직접적으로 기대되고, 더불어 소유권 있는 캐시 라인의 재사용빈도가 높으므로 MR의 측면에서도 피해가 크지 않을 것으로 보인다.

또 다른 예로 RADIX를 들 수 있다. RADIX의 경우, 1000번의 메모리 접근에 대한 M 상태에서 M 상태로 옮기는 메모리 접근은 802.282로 80%에 이른다는 사실을 표 4를 통해 알 수 있다. 또한 그림 4에 따르면 전체 캐시 영역에서 소유권 있는 라인은 약 84%를 차지한다. 따라서 RADIX의 경우, 소유권을 가지고 있는 캐시 라인의 재사용빈도가 그렇지 않은 캐시 라인에 비해

표 3 LU에서 1000개 메모리 접근 당 상태 변화의 빈도

Application			To				
			NP	I	E	S	M
LU	From	NP	0	0	0.0000	0.6593	0.0011
		I	0.0000	0	0	0.0002	0.0003
		E	0.0000	0	0.4454	0.0004	0.2164
		S	0.0339	0.0001	0	302.702	0.0000
		M	0.0001	0.0007	0	0.2164	697.129

표 4 Radix에서 1000개 메모리 접근 당 상태 변화의 빈도

Application			To				
			NP	I	E	S	M
RADIX	From	NP	0	0	0.004746	3.524705	11.41111
		I	0.130988	0	0	1.108079	4.57868
		E	0.000759	0.002848	0.080301	0	0.00019
		S	0.029804	1.120988	0	178.1932	0.817818
		M	0.044232	11.53127	0	4.03157	802.282

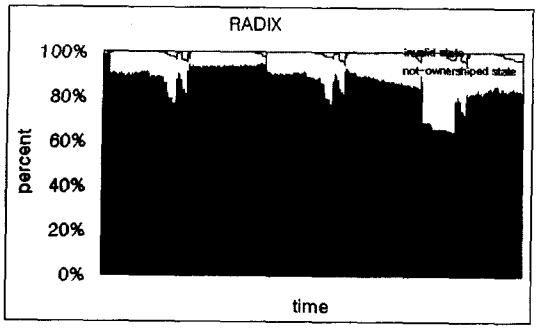


그림 4 RADIX에서 시간에 따른 원격 캐시 라인 상태의 비율

현저히 높다고 말할 수는 없다. 그러나 표 4를 보면, E, S 상태에서 M 상태로 옮겨지는 메모리 접근이 0.818008로 0.08% 정도이므로 E, S상태에서 M 상태로 옮겨지는 경우가 드물다는 사실을 알 수 있고, M 상태에서 I 상태로 옮겨지는 메모리 접근이 11.53127로 1.15% 정도라는 데서 M 상태에서 다른 노드의 필요에 의해 소유권을 빼앗길 확률도 1.15% 정도에 머무른다는 점을 확인할 수 있다. 즉, 이와 같이 캐시 라인의 소유권이 좀처럼 이동하지 않는 경우, 자주 사용되는 노드에 해당 캐시의 소유권이 머물도록 하여 소유권 이동으로 인한 작업 부하를 피할 수 있다.

3.2 캐시 일관성 프로토콜

다중 프로세서 시스템에서 캐시를 사용하여 메모리 접근 지연을 줄일 때, 캐시 일관성을 유지하여야 한다. 본 논문에서는 캐시 일관성을 유지하기 위한 프로토콜은 버클리 프로토콜을 가정한다. 버클리 프로토콜은 MS 상태를 두어, 캐시 라인이 공유될 때에도 소유권을 메모리로 옮기는 횟수를 줄인다. 일리노이 프로토콜의 경우 캐시 라인이 공유될 때에는 무조건 소유권이 메모리로 옮겨지므로, 추가적인 지연 시간이 발생한다. 버클리 프로토콜은 I, S, MS, M의 네 가지 상태로 이루어진다. 각 상태에 대한 설명은 다음과 같다.

① I (Invalid) : 현재 원격 캐시에 해당 주소에 대한 유효한 데이타가 없음을 의미한다.

② S (Shared) : 현재 원격 캐시에 해당 주소의 유효한 데이타가 존재하며 자기 노드의 프로세서는 읽기 권한만 있는 상태를 나타낸다.

③ MS (Modified Shared) : 현재 원격 캐시에 해당 주소의 유효한 데이타가 존재하며 자기 노드의 프로세서는 읽기 권한만 있으나 다른 노드의 데이타 요청에 응답해야 할 책임이 있다. 이 때의 데이타는 홈의 메모리와 일치하지 않으며 원격 캐시 안의 데이타가 가장 최신의 데이타이다.

④ M (Modified) : 현재 원격 캐시의 해당 주소의 데이타가 존재하며 자기 노드의 프로세서는 읽기와 쓰기 권한이 모두 있으며 다른 노드의 데이타 요청에 응답해야 할 책임이 있다. 이 때의 데이타는 홈의 메모리와 일치하지 않으며 가장 최신의 데이타는 자기 노드의 프로세서나 원격 캐시 안에 존재한다.

상태 전이도는 그림 5와 같다.

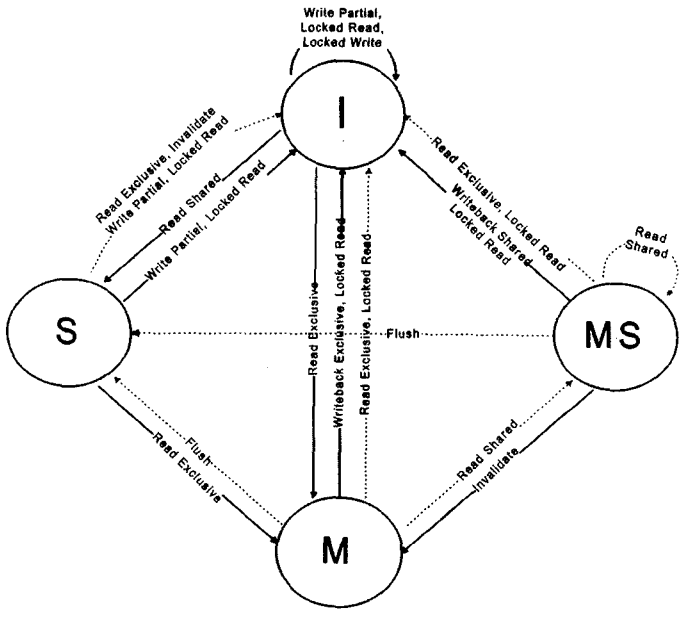


그림 5 캐시 상태 전이도

그림 5에서 실선으로 표시된 트랜잭션은 현재 캐시가 위치한 노드에서 들어온 트랜잭션들이고 점선으로 표시된 트랜잭션은 외부 노드에서 들어온 트랜잭션이다.

3.3 소유권 유지 교체 정책

본 논문에서 원격 캐시는 4-way 연관 사상(Set-Associativity)을 가정한다. '소유권 유지 교체 정책'은 소유권을 가지고 있는 라인을 최대한 캐시에 유지하는 것으로 다음과 같이 동작한다. 캐시 라인의 상태를 관찰하여 우선적으로 I 상태를 교체한다. 모든 I 상태 라인이 교체된 후의 교체 요구 시 S 상태인 라인 중 하나를 임의로 선택한다. I, S 상태가 전혀 없는 상황에서는 M, MS 상태인 라인 중 하나를 임의로 선택하여 교체한다. 이 정책은 가능한 M, MS 상태의 캐시라인은 계속 유지하는데 목적이 있다. 같은 상태인 라인 중 하나를 선택할 시에 LRU, LFU 등을 보조적으로 사용할 수도 있지만, 추가적인 메모리 사용을 가능한 피하고 회로를 간단히 하기 위하여 임의 선택 방식을 따른다. 알고리즘은 그림 6과 같다.

그림 7은 4-way 연관 사상 캐시에서 소유권 유지 교체 정책의 한 예이다. M 상태인 0번 라인은 교체되지 않고 계속 유지되고, 나머지 라인은 임의로 교체된다.

그러나 이러한 경우는 캐시가 M 상태로 포화되어 S 상태의 라인이 계속적으로 교체되는 상황이 발생할 수 있다. 그림 8은 E, F가 M 상태로 캐시에 들어오면서 4개 중 3 개가 M 상태가 된 경우이다. 이후, G, H 라인이 들어오면 계속적으로 캐시라인 2번만 교체된다. 나머지 라인이 M 상태이고 한 라인만 S 상태인 경우, 캐시가 마치 원래 크기의 25%에 불과한 직접 사상(Directed Map)인 것처럼 동작하게 된다. 이런 시나리오에서는 캐시는 가용 공간의 일부만 사용하게 되어 낭비가 심하게 된다.

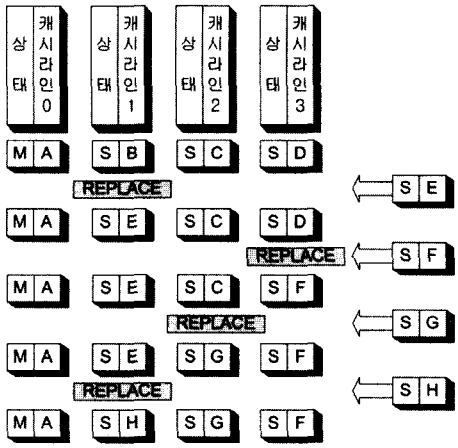


그림 7 소유권 유지 교체 정책의 예

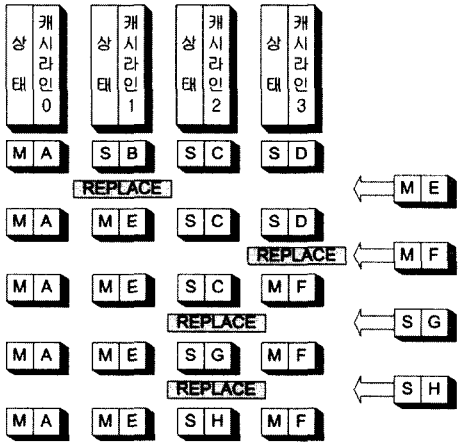


그림 8 소유권 유지 교체 정책의 문제점

```
Keeping_Ownership(){
    selected_way = 0;
    for (i=0; i<remote_cache_associativity; i++)
        // Invalid 상태가 있으면 그 라인을 교체하고 종료한다.
        if(cache_line[i].state == INVALID){
            selected_way = i;
            GOTO END;
        }
    if(SHARED 상태 라인이 존재)  // SHARED 상태 라인이 존재하면
                                 // 그 중에서 임의로 선택한다.
        selected_way = randomly_select_from_candidate_state
        _shared();
    else  // 없다면, 나머지 중 임의로 선택한다.
        selected_way = randomly_select_from_candidate_state
        _owner();
    }
END:
    return selected_way;
}
```

그림 6 소유권 유지 교체 정책의 알고리즘

이 문제를 해결하기 위해 참조 비트를 두거나, 가장 최근에 사용한 라인을 기록[16]하는 방법 등이 제시될 수 있다.

3.3.1 참조 비트를 사용한 소유권 유지 교체 정책

이 절에서는 공유 상태 라인만의 교체를 피하기 위하여 참조비트를 사용한 소유권 유지 교체 정책(Keeping-Ownership replacement policy with Reference Bit, KORB)을 제안한다. 이 정책은 기본적으로 소유권 유지 교체 정책을 따르면서, 참조 비트는 각 라인에 하나씩 두어, 최근에 참조되었음을 나타낸다. 교체는 참조 비트가 리세트된 라인에서만 이루어진다. 참조 비트는 새 라인이 캐시에 들어올 때 세트되고, 메모리 접근이 일어나 캐시에서 적중될 때도 세트된다. 만약 네 개의 캐시 라인의 참조 비트가 모두 세트되면 가장 마지막에

참조된 라인을 남기고 나머지는 리세트한다. 참조 비트가 세트되면 S 상태에서도 교체되지 않고, 반대로 참조 비트가 리세트되면 M이나 MS 상태에서도 교체가 가능하다. 이를 통해 사용되지 않고 자리를 차지하는 M, MS 상태의 캐시 라인을 교체하고, 또 빈번히 사용되는 S 상태의 캐시 라인이 캐시에 유지된다. 알고리즘은 그림 9와 같다.

그림 10은 참조 비트를 사용한 소유권 유지 교체 정책의 한 예로, 그림 8의 시나리오를 따랐다. M 상태인 E가 들어올 때, 참조 비트가 리세트된 S 상태의 캐시 라인 1, 2중 하나를 선택하였다. 캐시 라인 3은 참조 비트가 세트되어 선택되지 않는다. 또한 S 상태인 G 라인

```
KORBO{
        selected_way;
        for (i=0; i<remote_cache_associativity; i++)
                if(cache_line[i].state == INVALID){
                        selected_way = i;
                        GOTO END;
                }
        if(리세트 중 S 라인 존재)// 리세트된 라인 중 SHARED 상태 라인이 존재하면
                                // 그 중에서 임의로 선택한다.
                selected_way = randomly_select_from_candidate_reset_shared();
        else    // 나머지 리세트된 라인 중 M, MS 상태 라인을 임의로 선택한다.
                selected_way = randomly_select_from_candidate_reset_owner();
        }
END :
        cache_line[selected_way].reference = SET;
// 만약 모든 캐시 라인이 세트되면 최근 선택한 라인을 제외하고 모두 리세트한다.
        if(all reference bit of cache_line = SET)
                all reference bit of cache_line except selected_way = RESET;
        return selected_way;
}
```

그림 9 참조 비트를 사용한 소유권 유지 교체 정책의 알고리즘

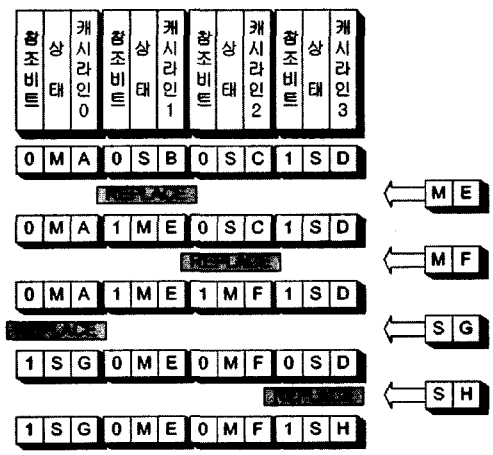


그림 10 참조 비트를 사용한 소유권 유지 교체 정책

이 들어올 때, 만약 참조 비트가 없었다면, 캐시 라인 3이 교체되고 이후, H 역시 캐시 라인 3에 교체되지만, 캐시 라인 3의 D는 참조 비트가 세트되어 교체되지 않고 대신 캐시 라인 0의 A가 교체된다. S 상태인 G는 M 상태인 A를 교체하여 캐시에 들어간다. 이 때, 전체 캐시 라인의 참조 비트가 세트되므로 라인 0의 G를 제외한 나머지의 참조 비트는 리세트된다. 참조 비트는 n-way의 경우 1에서 n-1 개까지 세트될 수 있다.

3.3.2 MRU를 사용한 소유권 유지 교체 정책

전술한 바와 같이 참조 비트를 사용할 시에는 소유권이 없는 라인만이 교체되는 단점을 줄일 수는 있지만, 그 결과 소유권을 가지고 있는 라인의 교체 비율이 순수한 소유권 유지 교체 정책에 비해 상대적으로 높아졌다. 이 절에서는 소유권을 유지하는 비율을 높이기 위해서 MRU(Most Recently Used) 라인을 기록하여, 그 라인은 제외하고 소유권 유지 교체 정책을 적용하는 방법을 제안하고자 한다[16].

MRU를 사용한 소유권 유지 교체 정책(Keeping-Ownership replacement policy with MRU, KOM)은 참조 비트를 사용한 소유권 유지 교체 정책과 대부분 같다. 다만 참조 비트를 각 라인마다 두지 않고 가장 최근에 사용한 한 라인만을 기록한다. 즉, MRU로 지정되지 않은 라인에서만 소유권 유지 교체 정책을 적용하여 교체될 라인을 선택한다. MRU는 n-way에서 $\log n$의 추가 메모리가 필요하므로 4-way의 경우 참조 비트 사용 시 4 비트가 필요한데 비해 절반인 2 비트로 충분하다. 알고리즘은 그림 11과 같다.

```
mru; // most recently used
KOM0{
        selected_way = 0;
        for (i=0; i<remote_cache_associativity; i++)
                if(cache_line[i].state == INVALID){
                        selected_way = i;
                        GOTO END;
                }
        if(MRU가 아닌 라인 중 SHARED 상태 라인이 존재)
                        // MRU를 제외한 SHARED 상태의 라인이 존재하면
                        // 그 중에서 임의로 선택한다.
                selected_way = randomly_select_from_candidate_shared_except_MRU();
        else{
                // MRU를 제외한 나머지 중 임의로 선택한다.
                selected_way = randomly_select_from_candidate_owner_except_MRU();
        }
END:
        mru = selected_way;
        return selected_way;
}
```

그림 11 MRU를 사용한 소유권 유지 교체 정책의 알고리즘

위의 알고리즘의 예가 그림 12에 나타나 있다. 특히 S 상태의 H가 캐시에 들어올 때, S 상태인 2번 캐시 라인에 들어와야 하나, MRU가 2로 되어 있으므로 M 상태인 0, 1, 3번 캐시 라인 중 하나를 임의 선택하여 들어오고, 그 위치를 MRU에 기록하였다.

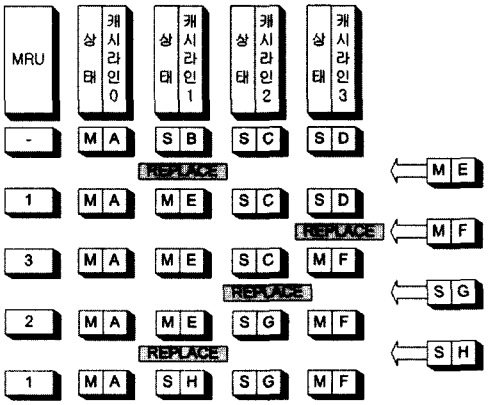


그림 12 MRU를 사용하는 소유권 유지 교체 정책

4. 성능 평가(Performance Evaluation)

4.1 모의실험 환경

본 논문에서 제안한 소유권 유지 교체 기법을 평가하기 위해서 MINT를 사용하여 모의실험을 수행하였다. MINT는 프로그램 구동 방식의 시뮬레이터로서 이를 이용한 모의실험은 시뮬레이터에서 직접 실행 파일을 읽어 프로그램을 수행하기 때문에, 입력 프로그램의 수정 없이 적용이 가능하며, 수행 과정에서의 시스템 상황에 따른 변화도 재현된다[18].

모의실험 환경은 500Mhz 클럭의 CPU 4개로 이루어진 노드 8개를 가정하였다. 이 CPU들은 100Mhz의 지역 버스로 연결되어 있고, 지점 간 링크는 500Mhz로 동작하고, 16비트 전송이 가능하여 1GB/s의 대역폭을 가진다. 원격 캐시는 4-way 연관 사상이라 가정한다. 여러 시스템 인자 값은 표 5와 같다.

표 5 모의실험 인자

인자	값
프로세서 수	32
한 노드내의 프로세서 수	4
노드 수	8
프로세서 캐시의 크기	16KB
원격 캐시의 크기	64, 128, 256, 512, 1024KB
프로세서 클럭 속도	500Mhz
노드 간 링 연결 클럭 속도	500Mhz
프로세서 캐시 접근 시간	2 Processor Clock
버스 요청 명령 전송 시간	9 Bus Clock
버스의 단위 블록 데이타 전송 시간	18 Bus Clock
노드 간 요청 명령 전송 시간	100 Ring Clock
노드 간 블록 데이타 전송 시간	400 Ring Clock

4.2 벤치마크 프로그램

벤치마크용 프로그램은 SPLASH-2[19]에서 제공하는 프로그램 중 FFT, Radix, LU, Barnes, Ocean을 사용하였다. FFT는 Fast Fourier Transform를 수행하는 프로그램으로 전치 행렬을 구하는 과정에서 모든 프로세서들 간에 상호적인 통신을 발생시킨다. LU는 행렬의 LU 변환을 수행하는 프로그램이고, Radix는 기수 정렬을 수행하는 프로그램이다. 그리고 Barnes는 Barnes-Hut Hierarchical N-body Problem 방식을 사용하여 각 입자 사이의 상호 작용을 구하는 프로그램이다. 마지막으로 Ocean은 red-black Guass-Seidel multigrid equation solver를 사용하여 해수의 움직임을 모델링하는 프로그램이다. 프로그램의 인자들은 표 6과 같다.

표 6 벤치마크 입력 크기

응용 프로그램	인자
FFT	512K Complex Doubles
RADIX	1M Integers, Radix 1024
LU	512×512 Matrix, 16×16 Blocks
BARNES	8K Particles(Bodies)
OCEAN	258×258 Ocean

4.3 비교할 교체 정책(Comparisons)

모의실험을 통해 비교할 대상은 LRU 교체 정책, Pseudo LRU(PLRU) 교체 정책[17], 참조 비트를 사용한 소유권 유지 교체정책(Keeping-Ownership with Reference Bit, KORB), 그리고 MRU를 사용한 소유권 유지 교체정책(Keeping-Ownership with Mru, KOM)이다.

LRU 교체 정책은 가까운 시점에 사용된 라인일수록 나중에 교체하는 정책이다. LRU 교체 정책은 사용 메모리가 크고 상태 변이 회로가 복잡하다. 캐시가 4-way인 경우, 접근 순서에 따라 24(4!) 가지의 상태를 가지므로 상태를 포시하기 위해 5비트가 필요하다. 캐시 라인을 A, B, C, D라 나타낸다면, 사용된 순서에 따라,

ABCD, ABDC, ACBD, ADBC, ACDB, ADCB,
BACD, BADC, CABD, DABC, CADB, DACB,
BCAD, BDAC, CBAD, DBAC, CDAB, DCAB,
BCDA, BDCA, CBDA, DBCA, CDBA, DCBA

의 상태가 존재하게 된다. 만약 현재 상태가 ABCD인데, D가 접근한다면, 순서는 DABC로 변한다. LRU 교체 정책은 추가로 필요한 메모리가 많다는 단점보다 상태 전이 회로가 복잡하고 크다는 단점이 더 나쁜 영향을 차지한다. 24가지 상태가 A, B, C, D의 4가지 접근

에 따라 변하는 회로는 굉장히 크고 느리다. 이런 이유로, 순수한 LRU를 사용하는 예는 거의 찾을 수 없다. 순수한 LRU를 사용하는 대신 NLU(Not Last Used)라 하여 가장 최근에 접근한 라인을 제외한 나머지에서 임의로 추출하는 방법이나 아래에서 설명할 PLRU 등의 교체 정책을 대신 사용한다.

PLRU 교체 정책은 LRU 교체 정책의 일종으로, n-way의 캐시의 경우 $n-1$ 비트가 필요하다. 그림 10은 PLRU에서 교체될 라인을 선택하는 알고리즘을 나타낸다. 각 비트는 가장 최근에 일어난 접근이 좌측인지 우측인지를 기록한다. 좌측에 접근이 일어나면 0으로, 우측에 일어나면 1로 기록한다. 교체할 시에는 그림 10에서와 같이 기록된 값이 0이면 우측에서, 아니면 좌측에서 교체될 라인을 선택한다. 예를 들어 A way의 접근이 있고, 뒤이어 C way의 접근이 있었다고 하자. 우선 A의 메모리 접근에 AB/CD 비트와 A/B 비트는 0이 되고, C/D 비트는 계속 값을 유지한다. 뒤 이은 C의 접근에 AB/CD 비트는 1이 되고 C/D 비트는 0이 된다. 이 때 라인의 교체가 일어난다면, AB/CD 비트가 1이므로 좌측인 AB에서 선택하고, A와 B 중의 선택은 A/B 비트가 0이므로 우측인 B가 선택된다.

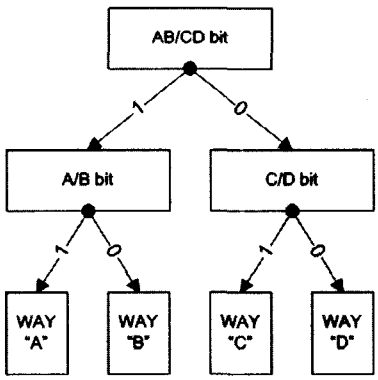


그림 13 Pseudo LRU

PLRU는 LRU의 60%인 3비트면 충분하고, LRU와 달리 각 비트의 전이 회로가 간단하다. 반면 LRU 교체 정책은 선택된 라인에 따라 상태가 복잡하게 변하여 하드웨어 복잡도가 높다. 즉, PLRU 교체 정책은 LRU 특성을 포함하면서도 상태 천이 회로가 간단하다는 장점이 있다. KORB, KOM 등의 소유권 유지 교체 정책 역시 하드웨어 복잡도가 낮다는 점에서 PLRU와 공통점이 있다. 따라서 순수 LRU는 실제 사용되는 예가 거의 없다는 점, 소유권 유지 교체 정책과 PLRU의 하드웨어 비용이 비슷하다는 점 등의 이유로 비교 기준을 PLRU로 정한다. LRU 교체 정책에서 상태 천이 시 오버헤드를 실제보다 크게 가정하여 모의실험 한다면, 소유권 유지 교체 정책들이 상대적인 이득을 얻게 된다. 따라서 LRU 교체 시 캐시 라인의 상태 천이 오버헤드는 없다고 가정한다.

LRU, PLRU, KORB, KOM의 4가지 교체 정책에 대하여 FFT, RADIX, LU, BARNES, OCEAN에서의 수행 시간 성능 향상을 원격 캐시를 64, 128, 256, 512, 1024KB로 하여 비교한다. PLRU에 비교한 수행 시간 그리고 LRU에 비교한 캐시 적중률을 살펴본다.

4.4 모의실험 결과와 분석

그림 14를 보면 MRU를 사용한 소유권 유지 교체 정책이 하드웨어 비용이 적음에도 불구하고 LRU에 근접한 성능을 보여주고 있다. PLRU는 캐시 적중률을 LRU처럼 높이지도 못하고, 소유권 유지 교체 정책처럼 비용이 큰 캐시 교체를 줄이지 않아서 미스 패널티가 높기 때문에 PLRU의 성능이 다른 정책에 비해 낮다.

표 7을 보면 소유권 유지 교체 정책은 PLRU에 비하여 참조 비트를 사용한 경우(KORB) 약 13%, MRU를 사용한 경우(KOM) 약 25% 성능 향상을 보였다. 순수 LRU의 경우 약 28% 성능 향상을 보이는데 이는 MRU

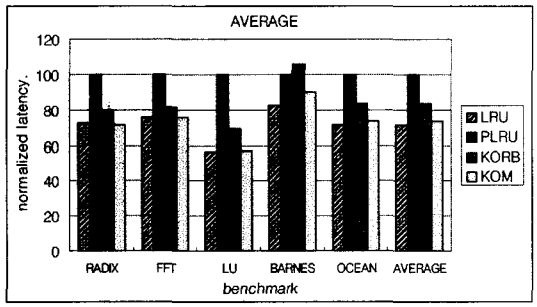


그림 14 PLRU를 기준으로 표준화한 벤치마크별 평균 수행 시간

표 7 PLRU를 기준으로 한 벤치마크별 평균 수행 시간의 비율(%)

	RADIX	FFT	LU	BARNES	OCEAN	AVERAGE
LRU	72.44568	75.3477	56.3748	82.28427	71.30064	71.55062
PLRU	100	100	100	100	100	100
KORB	78.55344	81.29382	69.3232	105.5207	99.17906	86.77404
KOM	71.17216	75.46591	56.83426	89.67899	79.67596	74.56546

표 8 LRU를 기준으로 한 벤치마크별 캐시 적중률(%)

	RADIX	FFT	LU	BARNES	OCEAN	AVERAGE
LRU	100	100	100	100	100	100
KORB	63.65819	86.73319	75.00042	47.97614	68.59821	68.39323
KOM	100.5947	98.7871	95.53229	71.82957	84.03677	90.15609

를 사용한 소유권 유지 교체 정책(KOM)과 상당히 유사하다. LRU에서 상태 천이 오버헤드를 고려하지 않았다는 점을 고려한다면 더욱 그렇다. 특히 MRU를 사용한 소유권 유지 교체 정책은 4-way의 캐시 한 세트에 2 비트 메모리를 추가하는 반면, LRU는 5 비트를 추가해야 할 뿐 아니라 메모리 접근으로 인한 상태 천이 회로의 복잡도가 높으므로 하드웨어적 비용의 측면에서 MRU를 사용한 소유권 유지 교체 정책이 우위를 점한다고 할 수 있다.

표 8에 따르면, KOM의 캐시 적중률이 LRU에 비해 평균 10% 낮다. 그러나 표 7에서처럼 수행 시간은 LRU에 비해 평균 2.8% 낮다. 캐시 적중률은 떨어졌지만, 소유권을 가진 캐시 라인의 캐시 적중률은 높아져 수행 시간의 단축을 가져왔기 때문이다.

이 후, 실험 결과를 벤치마크 당 캐시 크기를 달리하여 다시 살펴본다. 그림 15, 17, 19, 21, 23은 PLRU를 기준으로 표준화한 벤치마크 별 수행 시간을 나타내고, 그림 16, 18, 20, 22, 24는 LRU를 기준으로 표준화한 캐시 적중률을 나타낸다.

그림 15를 보면, 캐시 크기가 커짐에 따라 전반적으로 성능 향상이 나타남을 알 수 있다. 특히, KOM은 성능이 LRU와 비슷하다. KORB의 경우 캐시가 증가하면서 성능향상 효과가 더 크다는 것을 알 수 있다. 또한, 그림 16에서도 확인할 수 있는 바와 같이 캐시 적중률의 증가가 매우 크다. 이 이유는 KORB가 KOM에 비해 캐시를 효율적으로 사용하지 못하기 때문이다. 그러나 KORB에서 소유권 유지 교체 정책의 특성을 더욱 분명히 발견할 수 있다. KORB는 RADIX에서 평균적으로 LRU에 비해 캐시 적중률이 64%에 불과하지만 수행 시간에서 손해는 6% 정도이다. 즉, 캐시 적중률에서 손해를 입지만, 소유권을 옮기는 작업을 최소로 줄여서 수행 시간을 PLRU에 비해 단축할 수 있었다.

RADIX는 읽기 위주로 잦은 이동이 발생하는 프로그램이다. RADIX에서 캐시의 상태를 살펴보면, 주로 M 상태의 라인으로 이루어져 있다. S 상태도 드물고, MS 상태는 구간의 반복(iteration)이 끝날 때마다 잠시 나타난다. M 상태의 라인의 이동이 잦지만 지역 노드에서 반복적으로 사용될 가능성이 크므로, 외부 노드의 무효화 요구에 의해 캐시에서 내보내질 때까지 캐시에 남겨두는 편이 좋다. 이 성질이 소유권 유지 교체 정책과 맞아 성능상의 이점을 보인다. RADIX에서는 KOM의 수행 시간이나 캐시 적중률이 LRU보다 평균적으로 높은 것으로 나타난다.

그림 17, 18을 보면 RADIX와 유사한 양상을 보임을 알 수 있다. 캐시가 증가함에 따라 수행 시간이 단축되

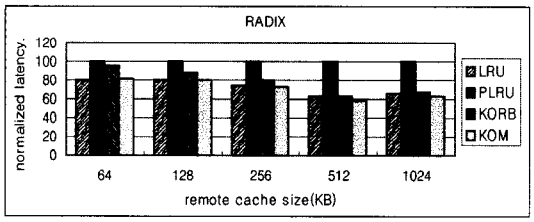


그림 15 RADIX에서 PLRU를 기준으로 표준화한 수행 시간

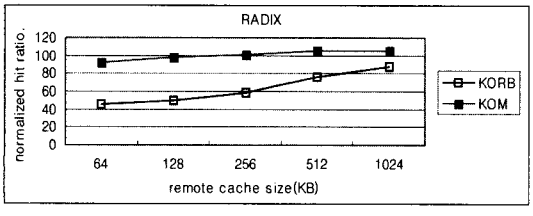


그림 16 RADIX에서 LRU를 기준으로 표준화한 캐시 적중률

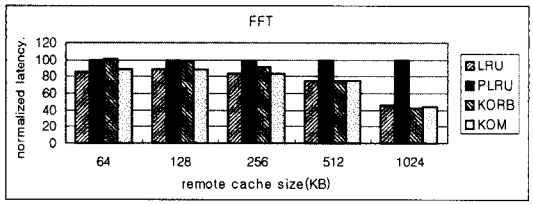


그림 17 FFT에서 PLRU를 기준으로 표준화한 수행 시간

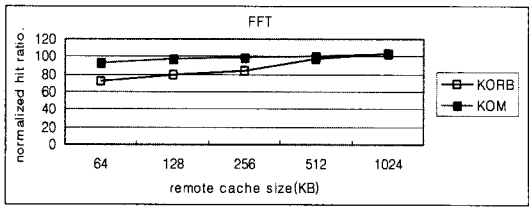


그림 18 FFT에서 LRU를 기준으로 표준화한 캐시 적중률

고 캐시 적중률이 높아진다. FFT는 데이타가 한 프로세서에서 다른 프로세서로 이동(Migration)하는 경우가 많으며, 프로세서는 그 데이타에 대해 읽기, 쓰기 연산을 병행하여 수행한다. 따라서 M, MS 상태로 캐시에 적재될 확률이 높으며 반복적으로 사용될 가능성이 크기 때문에 캐시 적중률이 높고, 그래서 수행 시간이 단축되었다.

수행 시간이나 캐시 적중률에서 LRU와 소유권 유지 교체 정책이 차이가 거의 없는데, LRU적인 면에서 보아도 M, MS 상태의 라인이 반복적으로 사용되기 때문이다. 즉, 소유권을 가진 라인이 반복적으로 옮겨짐으로 소유권을 가진 라인의 접근 빈도가 높고 따라서 LRU 정책을 따를 때에도 M의 교체 확률이 낮다. 결국 LRU와 소유권 유지 교체 정책은 비슷한 양상을 나타낸다. 성능 면에서도 KOM과 LRU는 차이가 거의 없다.

LU는 Working Set이 크다는 특성이 있다. 그러한 이유로 캐시가 커지면 성능 향상의 효과가 다른 벤치마크보다 크다. 뿐만 아니라 LU는 공간 지역성(Spatial Locality)이 높다. 따라서 그림 19를 보면 캐시가 커지면서 수행 시간이 급격히 줄어드는 것을 알 수 있다. LU는 배리어(Barrier)를 사용한 동기화에 많은 시간을 소모한다[19]. 따라서 어느 정도 이상의 메모리 접근 시간은 줄일 수 없고 캐시 미스율을 어느 정도 이상 낮출 수 없다. 그 결과 그림 20에서처럼 캐시 적중률이 더 이상 높아지지 못한다. 전반적으로 KOM의 성능이 캐시 크기에 따라 향상됨을 확인할 수 있다.

BARNES는 소유권 유지 교체 정책의 성능이 가장 낮은 벤치마크다. BARNES를 제외하고 전반적으로 캐시 크기가 커지면 KOM의 캐시 적중률이 LRU에 근접하거나 오히려 LRU를 능가하는데, 그림 21에서처럼 유독 BARNES에서는 캐시 크기가 커지면서 캐시 적중률이 LRU에 더욱 미치지 못하게 된다. 이는 BARNES가 읽기 위주인데다가, 여러 프로세서가 한 데이타를 공유하고 있는 경우가 많기 때문이다. 기본적으로 읽기 목적인 S 상태를 우선 교체하는 소유권 유지 교체 정책의 특성상 BARNES에서는 성능 향상의 효과가 클 수 없다. 그림 22에서처럼 소유권 유지 교체 정책은 캐시 적중률이 캐시 크기가 증가함에 따라 LRU에 미치지 못한다.

OCEAN은 전체 데이타 세트를 고정된 크기의 하부 그리드(sub-grid)로 나누고, 일정한 시간 간격으로 각 노드의 하부그리드에 대한 연산을 수행하기 위해 배리어(barrier)를 다른 벤치마크에 비해 많이 사용한다. OCEAN은 지역성이 큰 벤치마크이지만, 노드 사이의 데이타 공유율도 높다. 그림 23을 보면 LRU의 수행 시간이 64KB의 경우 PLRU의 51% 정도지만, 1024KB의 경우 71%까지 길어진다. OCEAN의 높은 지역성 때문에, 시간 지역성을 최대한 이용하는 LRU의 성능이 캐시 크기가 작은 경우 높게 나타나지만, 캐시 크기가 증가하면서 높은 데이타 공유율로 의해 수행 시간이 길어

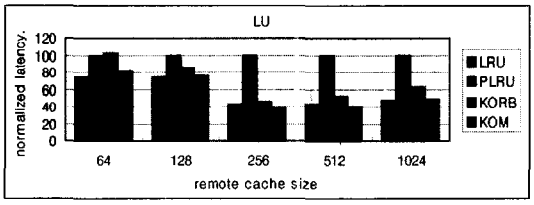


그림 19 LU에서 PLRU를 기준으로 표준화한 수행 시간

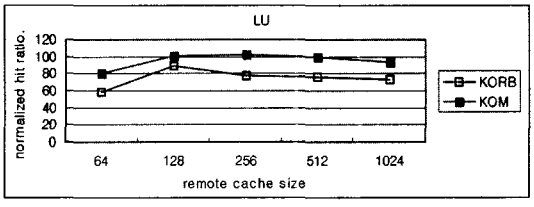


그림 20 LU에서 LRU를 기준으로 표준화한 캐시 적중률

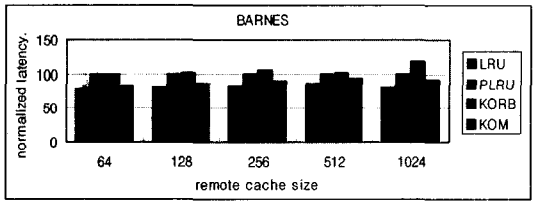


그림 21 BARNES에서 PLRU를 기준으로 표준화한 수행 시간

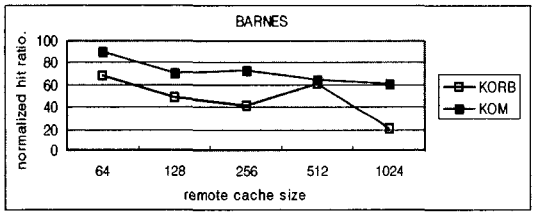


그림 22 BARNES에서 LRU를 기준으로 표준화한 캐시 적중률

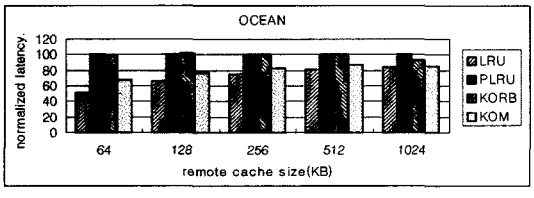


그림 23 OCEAN에서 PLRU를 기준으로 표준화한 수행 시간

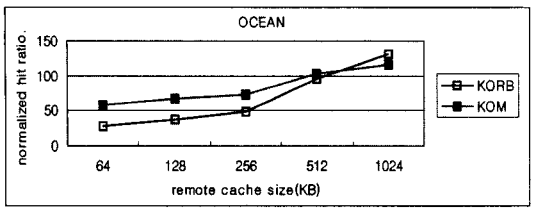


그림 24 OCEAN에서 LRU를 기준으로 표준화한 캐시 적중률

지고, 그 결과 KOM과의 성능 차가 15%에서 8%로 떨어진다. 이는 데이타 공유가 많은 경우, 캐시 라인이 소유권을 잃은 후, 소유권을 회복하는 작업 부하는 더 크기 때문이다. 그림 24를 보면, 캐시 크기가 커짐에 따라 KOM, KORB의 캐시 적중률이 높아짐을 알 수 있다. 이는 캐시가 작을 때는 M, MS 상태의 캐시 라인을 유지하는 일이 지역성을 활용하지 못하여 악영향을 미쳤으나, 캐시 크기가 커지면서 소유권이 좀처럼 옮겨지지 않는 OCEAN의 특성으로 인해 M, MS 상태의 캐시 라인을 유지하여 얻는 이득이 더 커졌기 때문이다.

앞서의 실험 결과를 통해, KOM의 성능이 KORB의 성능에 비해 평균적으로 높다는 사실을 알 수 있다. 이러한 결과를 설명하기 위해, KOM과 KORB의 특성을 아래에서 비교하여 살펴본다. KORB의 참조 비트는 한 라인만 세트된 경우 MRU와 차이가 없다. 그러나 진행 과정에서 2, 3 개의 참조 비트가 세트될 수 있다. 세트된 2, 3개의 참조 비트 중 MRU를 제외한 나머지는 사실 LRU적 특성이 떨어지고 교체될 라인을 선택할 때 선택의 폭만 줄이는 결과를 가지고 와서 전반적으로 성능이 KOM에 미치지 못한다. KORB의 경우 LRU 특성이 KOM에 비해 좋지 않고 KOM에 비해 S 상태의 라인이 캐시에 머무를 가능성을 좀 더 주고 있다. FFT에서 캐시 라인은 빈번히 이동하고 설령 캐시가 커진다고 해도 여러 프로세서에서 순차적으로 접근하는 메모리 영역이 많아 외부 노드에 의한 무효화 요구가 많다. 그래서 그림 18을 보면 FFT에서 캐시 사이즈가 커짐에 따라 KORB의 캐시 적중률은 높아진 반면, PLRU와 LRU는 무효화 요구를 피하지 못하여 캐시 적중률의 향상이 크지 않아 비슷한 값을 갖게 된다. 그림 17을 보면 KORB의 수행시간이 캐시가 커짐에 따라 짧아짐을 알 수 있는데, 이는 캐시 적중률의 향상에 기인한다. BARNES에서는 LRU에 비교하여 캐시 적중률이 평균 52% 정도에 미치며, 그림 22를 보면 캐시가 커질수록 적중률은 더 낮아진다. BARNES의 경우 읽기 위주이고 공유하는 빈도가 높다. 그 결과 S 상태의 캐시 라인이 대부분이다. 소유권 유지 교체 정책은 기본적으로 S 상태의 캐시 중 하나를 임의로 선택하는 성질이 있다. 그 결과 대부분이 S 상태이므로 단순한 임의 선택과 같은 특성을 나타내게 된다. KOM의 경우 MRU를 사용하여 다소나마 LRU 특성을 가지고 있지만, KORB의 경우 그렇지 않아 PLRU보다 나쁜 성능을 나타낸다.

5. 결론(Conclusions)

다중 프로세서 시스템에서 원격 캐시는 성능 향상을 위해 반드시 필요한 하드웨어이다. 특히 캐시 일관성을 유지하는 시스템에서 캐시 라인의 상태에 따른 교체 시 필요한 작업양은 캐시 라인의 상태에 따라 다르다. 특히, NUMA 시스템에서는 그 차이가 확연하다. 원격 캐시에서 S 상태의 캐시 라인의 교체는 특별한 처리 작업이 필요하지 않지만, M과 MS 상태의 경우에는 소유권 이전 등의 처리 작업이 필요하다.

따라서 본 논문에서는 처리 작업이 필요한 소유권을 가진 상태의 캐시 라인을 더 오래 유지시키고 소유권을 가지지 않은 상태의 캐시 라인을 우선적으로 교체하는 정책을 제안하였다. 그리고 버클리 프로토콜(Berkeley Protocol) 하에서, 참조 비트를 사용하는 소유권 유지 교체 정책과 MRU를 사용하는 소유권 유지 교체 정책을 LRU, 그리고 PLRU와 비교하였다. PLRU를 기준으로 MRU를 사용하는 소유권 유지 교체 정책은 수행 시간에서 25%, 참조 비트를 사용한 소유권 유지 교체 정책은 13% 성능 향상을 보였다. 특히 MRU를 사용한 소유권 유지 교체 정책은 LRU에 비해 하드웨어 복잡도가 극히 낮음에도 수행 시간에서 2.8%에 불과한 성능 차이를 보였다.

향후, 소유권 유지 교체 정책의 면밀한 분석을 위하여 노드 간 패킷량의 감소율을 조사할 필요가 있다. 특히 M과 MS 상태가 유지됨에 따라 메모리에 되쓰기(Write Back)를 줄일 수 있고, 패킷 전달에 따른 네트워크 사용 절약을 살펴보고 그 효과를 검증하여야 한다. 또한 벤치마크 별로 성능상의 큰 차이를 보이는 점을 보안하기 위해 정밀한 탐색이 필요할 것이다.

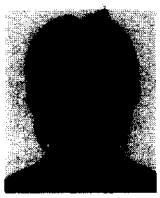
참 고 문 헌

[1] 김형호, "지점간 링크를 이용한 스누핑 버스의 설계 및 성능 분석", 서울대학교 석사학위 논문, 1996.

[2] D.J. Lilja. "Cache coherence in large-scale shared-memory multiprocessors : Issues and comparisons," ACM Computing Surveys, 25(3):303-338, Sept. 1993.

[3] Per Stenstrom, Truman Joe, and Anoop Gupta, "Comparative Performanc Evaluation of Cache-Coherent NUMA and COMA Architectures," In the 19th Int'l Symp. on Computer Architecture, pages 80-91, 1992.

[4] Namgi Kim, Sungkee Jean, Jinsoo Kim; Hyunsoo Yoon, "Cache Replacement Schemes for Data-Driven Label Switching Networks," IEEE Workshop on High Performance Switching and Routing, Page(s): 223-227, 2001.
[5] Yeung, K.H. Ng, K.W., "An Optimal Cache Replacement Algorithm for Internet Systems," 22nd Annual Conference on Local Computer Networks, Page(s): 189-194, 1997.
[6] Liangzhong Yin, Guohong Cao, Ying Cai, "A Generalized Target-Driven Cache Replacement Policy for Mobile Environments," 2003 Symposium on Applications and the Internet, Page(s): 14-21, 2003.
[7] Jain, P., Devadas, S., Engels, D., Rudolph, L., "Software-Assisted Cache Replacement Mechanism for Embedded Systems," IEEE/ACM International Conference on Computer Aided Design, Page(s): 119-126, 2001.
[8] Mounes-Toussi, F., Lilja, D.J., "The Effect of Using State-Based Priority Information in a Shared-Memory Multiprocessor Cache Replacement Policy," International Conference on Parallel Processing, Page(s): 217-224, 1998.
[9] Jaeheon Jeong, Michel Dubois, "Cost-Sensitive Cache Replacement Algorithms," In Proceedings of the 9th International Symposium on High-Performance Computer Architecture, pp.327- 337, February 2003.
[10] A.J.Smith, "Cache Memories," ACM Computing Surveys, vol. 3, pp. 473-530, September 1982.
[11] S.T.Srivivasan, R.D. Ju, A.R. Lebeck and C.Wilkerson, "Locality vs. Criticality," In Proceedings of the 28th International Symposium on Computer Architecture, pp. 132-143, July 2001.
[12] N.Young, "The k-server Dual and Loose Competitiveness for Paging," Algorithmica, vol.11, no.6, pp.525-541, June 1994.
[13] J.L. Hennessy and D.A. Patternson., "Computer Architecture: A Quantitative Approach," Second Edition, Morgan Kaufmann Publishers, 1996.
[14] 신정헌, "이중 링 스누핑 버스 시스템에서 성능 향상을 위한 원격 캐시 교체 정책 연구", 서울대학교 석사 학위 논문, 2001.
[15] David E. Culler and Jaswinder Pal Singh with Anoop Gupta, Parallel Computer Architecture : A Hardware/Software Approach, Morgan Kaufmann Publishers, Inc, Page(s): 306-311, 1998.
[16] Kimming So and Rudolph N.Rechtschaffen, "Cache operations by MRU change," IEEE Transactions on Computers, Vol.37(6):700-709, Jun 1988.
[17] Jim Handy, "The Cache Memory Book," Academic Press, Inc, Page(s): 49-60, 1993.
[18] J. E. Veenstra and R. J. Fowler. MINT: a front end for efficient simulation of shared-memory multiprocessors. In Proc. 2nd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pages 201-207, 1994.
[19] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. Methodological considerations and characterization of the SPLASH-2 parallel application suite. In Proc. 22th Annual International Symposium on Computer Architecture, 1995.



신 승 현

2000년 서울대학교 전기컴퓨터공학부(학사). 2002년 서울대학교 전기컴퓨터공학부(석사). 2002년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 computer architecture, parallel computer architecture, embedded system

곽 종 욱

1998년 경북대학교 컴퓨터공학과 공학사 2001년 서울대학교 대학원 컴퓨터공학과 석사. 2002년~현재 서울대학교 대학원 전기컴퓨터공학부 박사과정. 관심분야는 컴퓨터 구조, 고성능 병렬 처리 시스템, 내장형 시스템

장 성 태

1986년 서울대학교 전자계산기공학과 공학사. 1988년 서울대학교 대학원 컴퓨터공학과 석사. 1994년 서울대학교 대학원 컴퓨터공학과 박사. 1994년~현재 수원대학교 정보공학대학 컴퓨터학과 부교수 관심분야는 다중 프로세서 시스템, 병렬처리, 캐쉬 구조, 메모리 모델

전 주 식

1975년 서울대학교 응용수학과 학사. 1977년 한국과학기술원 전산학과 석사. 1983년 미국 Univ. of Utah 박사. 1983년~1985년 Univ. of Iowa 조교수. 1985년~현재 서울대학교 전기컴퓨터공학부 교수 1994년~1999년 서울대학교 컴퓨터신기술공동연구소 소장. 관심분야는 컴퓨터 구조, 병렬처리, VLSI/CAD