다중 프로세서 시스템에서 프로세서 지역성을 이용한 원격 캐쉬 교체 정책

(Remote Cache Replacement Policy using Processor Locality in Multi-Processor System)

한 상 윤 † 곽 종 욱 † 장 성 태 †† 전 주 식 †††
(Sang Yoon Han) (Jong Wook Kwak) (Seong Tae Jhang) (Chu Shik Jhon)

요 약 컴퓨터 시스템에서의 메모리 접근 지연은 전체 시스템 성능에 큰 장애 요인 중 하나이다. 특히 분산 메모리 구조에서 지역 메모리와 원격 메모리의 접근 지연 시간은 큰 차이를 나타낸다. 원격 메모리 접근 지연으로 인한 성능 저하를 줄이고자 원격 메모리 영역만을 캐싱하는 원격 캐쉬가 제안되었으며, 원격 캐쉬는 프로세서 캐쉬와 더불어 다단계 캐쉬 형태로 구성된다.

일반적으로 상위 계층 캐쉬의 모든 내용을 하위 계층 캐쉬가 반드시 포함하는 다단계 캐쉬 내포성(MLI)을 지키는 다중 계층 메모리 구조에서 LRU 교체 정책을 사용할 경우, 하위 계층 캐쉬의 LRU 알고리즘에 따른 라인 교체로 인하여 상위 계층 캐쉬의 라인 교체가 일어날 때, 상위 계층 캐쉬로 요구된 라인 교체가 상위 계층 캐쉬 자체의 LRU 정보와 일치하지 않는 경우가 발생하며, 이로 인해 상위 캐쉬의 적중률이 저하되어 전체 시스템 성능이 저하된다. 본 논문은 원격 캐쉬를 추가시킨 분산 공유 메모리 구조 다중 프로세서 시스템의 성능 향상을 위해 LRU 캐쉬 교체 정책의 단점을 보완한 새로운 원격 캐쉬 교체 정책을 제안한다. 논문에서 제안하는 교체 정책은 LRU 정보에 부가하여 프로세서의 시간적 접근 지역성을 이용하여 교체할 캐쉬 라인을 선택하게 함으로써, 프로세서에서 자주 사용되는 원격 캐쉬 라인의 교체가 일어나지 않도록 하여 시스템의 성능 향상을 꾀한다. 시뮬레이션을 통한 성능비교 결과, 본 논문에서 제시한 원격 캐쉬 교체 정책은 기존의 LRU 교체 정책과 비교하여 평균 5%, 최대 10%의 무효화 및 캐쉬 접근 실패를 감소시켰고, 이 결과 전체 시스템의 성능은 평균 2.5%, 최대 3.5% 향상되었다.

키워드 : 다중 프로세서 시스템, 원격 캐쉬, LRU 캐쉬 교체 정책, 다중 계층 캐쉬 내포성, 원격 메모리 접근 패턴, 프로세서 지역성

Abstract The memory access latency of the system has been a primary factor of performance degradation in single-processor system and multi-processor system. The remote memory access latency takes a lot of overhead over the local memory access latency especially in the distributed shared-memory system. To resolve this problem, the multi-level cache architecture that contains a remote cache in the multi-processor system has been proposed.

In this paper, we propose a new cache replacement policy that improves the performance of the multi-processor system with the remote cache. If the multi-level cache keeps the multi-level inclusion(MLI) property and uses the LRU(Least Recently Used) cache replacement policy, the LRU information of the higher-level cache(a processor cache) would be different with that of the lower-level cache(a remote cache). In this situation, the replacement of a remote cache line can induce the exchange of a processor cache line that is used by the processor. It is a main factor of performance degradation in a whole system. To alleviate this disadvantage of the LRU replacement policy, the new policy analyses the processor's remote memory access pattern of each node and uses this information to reduce the number of invalidations of the useful cache line in the higher-level cache. The new

† 학생회원 : 서울대학교 전기컴퓨터공학부
leaderhsy@panda.snu.ac.kr
leoniss@panda.snu.ac.kr
†† 종신회원 : 수원대학교 컴퓨터공학부 교수
stjhang@suwon.ac.kr
††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
csjhon@riact.snu.ac.kr
논문접수 : 2005년 2월 25일
심사완료 : 2005년 8월 5일

replacement policy of the remote cache can improve the performance by 3.5% in maximum and 2.5% in average on SPLASH-2 benchmarks, compared to the general LRU cache replacement policy.

Key words : Multi-processor system, remote Cache, LRU cache replacement policy, Multi-level cache Inclusion property, Remote memory access pattern, Processor locality

1. 서 론

지금까지 연구되어 온 대규모 병렬 구조 다중 프로세서 시스템은 메모리의 결합 형태에 따라 분산 메모리 구조(Distributed-Memory Architecture)와 중앙 집중 메모리 구조(Centralized-Memory Architecture)로 나뉘는데 분산 메모리 구조의 경우 자신의 노드와 원격 노드의 메모리 접근 시간이 상이한 성질을 가지고 있다. 이 경우 분산 메모리 구조 다중 프로세서 시스템에서 원격 메모리 영역에 대한 접근 지연이 매우 길어 시스템 전체 성능을 좌우하기 때문에 이를 줄이기 위해 도입된 것이 원격 캐쉬(Remote Cache)이다[1]. 이는 한 노드 안에 원격 캐쉬라는 한 단계 아래의 캐쉬를 추가하여 원격 메모리 영역에 대해서만 캐싱하여 노드 내 프로세서들의 원격 메모리 접근 지연을 줄여주고 또한 노드들 사이에 발생하는 일관성 트랜잭션을 중개하여 처리해주기 때문에 노드 내의 지역 버스 대역폭을 증가시켜주는 역할을 한다. 그 결과 분산 메모리 구조 다중 프로세서 시스템은 원격 캐쉬를 도입함으로써 여타의 다중 프로세서 시스템에 비하여 높은 성능과 용이한 구현 정도를 가지게 되었다[3].

한편, 한 시스템 내에 다수의 프로세서를 사용하기 위해 여러 단계의 버스와 캐쉬를 계층적으로 구성한 시스템이 연구되었다. 다단계 내포성이 지켜지지 않을 경우 스누프 필터링에 따른 메모리 접근 필터링이 되지 않는데, 이 경우 모든 메모리 접근이 시스템 내의 모든 캐쉬의 내용을 찾아 봐야 일관성 유지가 가능하다. 이러한 캐쉬 일관성 유지는 복잡하고 큰 지연을 발생시키기 때문에 이를 해결하고자 다단계 내포성이 제시되었다[1]. 이는 하위 단계 캐쉬가 상위 단계 캐쉬의 모든 내용을 포함하고 있어야 하는 성질로써, 다단계 캐쉬 구조에서 메모리 접근에 대한 프로토콜을 단순 명료하게 설계 및 구현을 가능하게 하고 메모리 접근 경로의 단순화로 접근 지연을 줄일 수 있게 하였다. 보통 분산 메모리 구조 다중 프로세서 시스템의 경우, 각 프로세서는 자신의 전용 캐쉬를 가지는 형태로 구성되나, 각 노드의 내부에 원격 메모리로의 접근을 감소시키기 위한 원격 캐쉬를 추가함으로써, 프로세서 캐쉬와 원격 캐쉬간의 결합을 통한 다단계 캐쉬 구조로 구현하여 원격 메모리 접근을 지역화하여 지연을 줄이는데 기여하였다. 또한 캐쉬 사이의 일관성 유지를 위해 다단계 내포성을 적용하여 노드 외부의 요청에 원격 캐쉬가 응답함으로써 내부 트랜잭션을 줄이는 등 여러 가지 성능 향상의 효과를 가져왔다.

이처럼 분산 메모리 구조 다중 프로세서 시스템에서의 원격 캐쉬는 한 노드 내의 프로세서들이 요구하는 원격 메모리 영역을 최대한 오래 유지함으로써 원격 메모리 접근 지연을 줄이도록 하여야 한다. 하지만 깊은 단계의 다단계 캐쉬 구조에서 내포성과 LRU 교체 정책은 캐쉬의 이용 목적인 시공간적 지역성을 최대한 지켜주지 못하는 원인으로 작용하기도 한다. 그 이유는 본 논문의 뒷부분에서 상세히 설명하겠지만 다단계 내포성이 지켜지는 가운데 야기되는 상위 단계의 캐쉬와 하위 단계 캐쉬 사이의 LRU 정보의 차이 때문이다. 본 논문은 이러한 LRU 캐쉬 교체 정책과 다중 계층 내포성의 결합으로 인해 발생하는 비효율적인 캐쉬 라인의 교체를 줄이고자 특정 정보를 이용한 새로운 캐쉬 교체 정책을 제시하기로 한다. 즉 LRU 교체 정책과 다단계 내포성이 주는 장점을 최대한 살리면서, 또한 이러한 특성으로 시스템 성능 저하의 원인이 되는 요인을 최대한 줄여 다중 프로세서 시스템의 성능 향상을 추구한 논문이다.

본 논문은 2장에서 관련 연구에 대해 서술하고, 3장에서는 새로이 고안한 원격 메모리 접근 지역성을 이용한 원격 캐쉬 교체 정책을 제시하고 있다. 그리고 4장에서는 실험 환경 및 실험 결과, 끝으로 5장의 결론으로 구성되어 있다.

2. 관련 연구

2.1 다중 프로세서 시스템 및 원격 캐쉬

본 논문에서는 새로이 고안된 원격 캐쉬 교체 정책을 구현하고자 그림 1과 같이 계층적 버스에 기반 한 분산 메모리 구조 다중 프로세서 시스템을 사용하였다[4].

그림 1과 같은 분산 메모리 구조 다중 프로세서 시스템, 특히 하드웨어적으로 캐쉬의 일관성을 유지하는 캐쉬 일관성 다중 프로세서 구조에서는 UMA 구조에서 나타나는 메모리 병목현상의 단점을 극복하고자 메모리를 지역적으로 분산시켜 놓았다. 이렇게 분산된 여러 지역 메모리들이 모여서 하나의 전역 주소 공간을 이루게 된다. 이처럼 메모리가 지역적으로 집중되어 있는 UMA 구조의 단점을 극복하며, 제어상의 용이성을 취한 것이

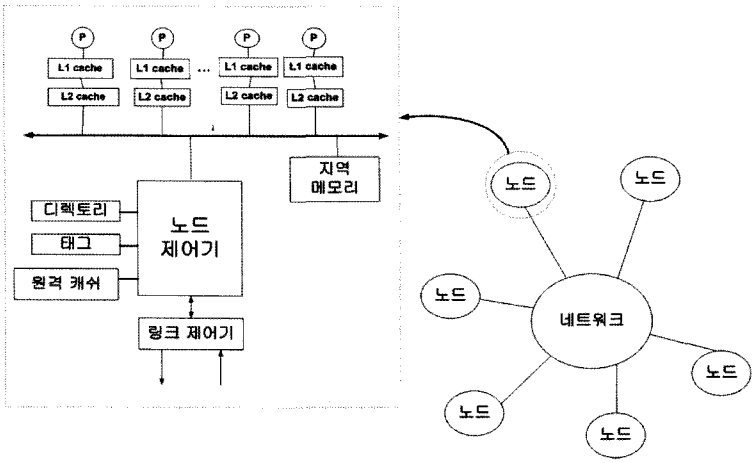


그림 1 NUMA 구조 형태의 다중 프로세서 시스템

원격 캐쉬를 포함하는 분산 메모리 구조 다중 프로세서 시스템이라 할 수 있다. 즉 원격 캐쉬는 여타의 시스템에 대해서 분산 메모리 구조 다중 프로세서 시스템이 성능상의 우위를 점하기 위한 필수 구성 요소로서, 노드 외부로 나가는 트랜잭션의 수를 줄이기 위해 원격 메모리 주소 영역에 대해 캐싱하는 역할을 한다. 그리고 그림 1에서 노드들을 연결하는 네트워크 종류가 분산 메모리 구조 시스템의 확장성을 다르게 보장한다. 캐쉬 일관성을 유지하기 위한 방법으로 스누핑 방식과 디렉토리 방식이 있다. 높은 확장성을 보장하기 위해 최근 두 단계 디렉토리 구조 cc-NUMA 다중 프로세서가 연구되었다[2]. 따라서 본 논문에서는 다단계 캐쉬 구조의 원격 캐쉬를 채택하여, 원격 메모리 접근에 대한 응답 지연 시간 감소를 통해 성능상의 향상을 꾀하고자 한다.

2.2 다단계 캐쉬 및 다단계 캐쉬 내포성

프로세서 클럭 속도가 급격하게 빨라지는 반면 메모리 접근 속도는 이를 제대로 지원하지 못하게 되자 지역성(Locality)을 이용한 캐쉬에 대한 연구가 많은 논문들을 통해 이루어졌다[1]. 초기에는 단일 캐쉬로 구성되었으나 프로세서와 메모리의 속도 차이가 더 증가함에 따라 새로운 구조의 캐쉬가 요구되었다. 이로 인해 제안된 것이 하드웨어는 작을수록 빠르다는 원리를 이용해 프로세서의 속도를 받쳐줄 작고 접근 속도가 빠른 캐쉬를 프로세서에 결합하고, 속도는 느리지만 메모리에 대한 접근을 최대한 줄일 수 있도록 크기를 늘린 캐쉬를 메모리에 가깝게 구성한 다중 계층 캐쉬 개념이 고안되었다[5]. 본 논문에서는 다중 계층 캐쉬에서 프로세서에 가장 근접한 캐쉬를 레벨 1 캐쉬, 다음 단계 캐쉬를 레벨 2 캐쉬, 레벨 3 캐쉬의 역할을 하는 원격 캐쉬는 원격 캐쉬라고 하기로 한다. 다중 프로세서 시스템에서 크기가 큰 하위 단계 캐쉬는 각 프로세서마다의 메모리 접근 수를 줄이고 메모리로부터의 데이타 적재를 줄여 시스템의 성능을 크게 향상 시킬 수 있다. 분산 메모리 구조 다중 프로세서 시스템과 같이 공유 메모리가 지역과 원격에 분산된 경우 원격 메모리에 대한 접근 지연이 수백 클럭에 달하므로 이를 줄이기 위한 원격 캐쉬의 필요성은 더욱 중요하다고 하겠다. 이러한 다단계 캐쉬가 시스템의 정확성을 해치지 않으면서 일관성 문제를 정확히 해결하고, 또한 하위 단계 캐쉬가 메모리, I/O 또는 다른 프로세서의 트랙픽에 의한 상위 단계 캐쉬로의 접근하는 것을 줄여 프로세서의 캐쉬 이용률을 높이는 다단계 내포성(MLI)이 제시 되었다[6].

프로세서 내부의 캐쉬로 레벨 1 캐쉬, 레벨 2 캐쉬가 있고 이들과 앞 절에서 언급한 하위 계층 캐쉬로 원격 캐쉬와의 계층적인 관계는 다중 계층 캐쉬 구조를 이룬다. 이러한 한 노드 내부의 다중 계층 캐쉬 구조에서 내부 요청과 외부 요청에 대한 효율적인 반응과 캐쉬 간의 일관성을 유지하기 위하여 여러 가지 프로토콜이 정의될 수 있다. 그 중 하나가 다단계 내포성인데, 이 성질이 본 논문의 시스템 구조에서 캐쉬 일관성 유지를 위해 사용된다.

다단계 내포성(MLI)이란 캐쉬가 다단계 캐쉬 구조를 이루고 있을 경우, 임의의 하위 단계 캐쉬에는 상위 단계 캐쉬의 모든 내용을 포함하고 있어야하는 성질을 말한다. 메모리 단계에 따른 집합 내포성을 식으로 나타내면 식 (1)과 같다.

$$M1 \subset M2 \subset M3 \subset \ldots \subset Mn$$

(Mi⊂Mj이고 i<j일 경우, i번째 캐쉬는 j번째 캐쉬보다 상위캐쉬)

식 (1) 내포성 성질 성립 조건

이를 유지함으로써 프로세서에서 멀리 떨어진 메모리와 I/O에서 발생하는 일관성 트랜잭션을 하위 단계 캐쉬가 필터링(filtering)함으로 상위 레벨로의 트랜잭션 흐름을 줄여 프로세서의 상위 단계 캐쉬의 이용 효율성을 증가 시켜준다. 이러한 다단계 캐쉬 내포성(MLI)을 유지하기 위해서는 다단계 캐쉬 구성에 있어서 여러 가지 제약이 있다. 가령 각 계층의 캐쉬마다 집합 연관성(Set Associativity) 및 캐쉬 블록 크기 등을 적절히 구성하여야 다단계 캐쉬 내포성을 지킬 수 있다[6].

이와 같은 다중 계층 내포성을 유지하고 그 제약 조건을 만족시키기 위해서는 유용한 상위 계층 캐쉬 라인의 무효화가 빈번하게 발생할 가능성이 있다. 이러한 단점을 해결하고자 내포성 성질을 깨뜨리는 다중 계층 캐쉬로도 설계하기도 하고[7], 내포성의 조건을 엄밀하게 유지하는 것이 아니라 제약 조건을 약하게 지킴으로써 효율적인 캐쉬 구조를 구성하는 약내포성에 대한 연구도 있었다[5].

2.3 캐쉬 교체 정책

캐쉬 교체 정책이란 캐쉬가 집합 연관성(Set Associativity)을 가지고 있을 때, 적중 실패시 비어있는 캐쉬 라인이 없을 경우 하나의 집합에 존재하는 여러 캐쉬 라인 중 어떤 라인을 교체할 것인지를 결정하는 정책을 말한다. 캐쉬의 접근 시간을 최대한 줄이는 것이 중요한 상위 레벨 캐쉬는 직접 연관 사상을 이용하고, 캐쉬의 접근 성공률을 높이기 위한 하위 레벨 캐쉬에서는 집합 연관성을 적절히 높여 사용함으로써 캐쉬의 이용률을 높이도록 설계한다. 일반적인 캐쉬 교체 정책으로 LRU(Least Recently Used), LFU(Least Frequently Used), 임의교체(Random)등이 있는데 LRU 및 LRU 응용 정책이 주로 사용된다[8-10]. 때로는 응용 프로그램의 메모리 참조 특성에 따라 LRU와 LFU를 동적으로 달리 적용하는 LRFU 캐쉬 교체 정책도 제안되었다[11]. 한 집합에 포함된 캐쉬 라인 중 임의의 라인을 선택하여 교체 대상으로 삼는 임의 교체 정책에서의 단점을 보완하기 위해서 LRU 교체 정책에서는 응용 프로그램의 캐쉬 접근 패턴을 분석하여 캐쉬 라인을 교체한다. LRU 교체 정책은 가장 최근에 사용되지 않은 블록을 기록하여 그 블록을 교체한다. 구현 방법은 매번 캐쉬에 대한 접근이 이루어질 때마다 해당 집합의 모든 유효한 캐쉬 라인에 대해 LRU 비트를 갱신한다. 집합 내에서 최근에 접근된 순서를 나타내는 LRU 비트는 캐쉬 라인 교체가 일어날 때, 가장 최근에 사용되지 않은 블록을 구분해 주는 정보로 사용되는데, 이 라인이 바로 교체 라인으로 선택된다. 이러한 LRU 교체 정책은 집합 연관성이 커짐에 따라 LRU 비트 갱신에 대한 복잡성이 증가하는 단점이 있지만 여타 교체 정책에 비교하여 뛰어난 성능을 보여준다.

3. 프로세서 지역성을 이용한 원격 캐쉬 교체 정책

본 절에서는 앞서 언급한 다중 계층 캐쉬 구조에서 다중 계층 내포성과 하위 레벨 캐쉬의 LRU 교체 정책으로 인한 프로세서 내부 캐쉬(레벨 1, 레벨 2)의 유용한 라인의 무효화를 줄여 레벨 1 캐쉬 및 레벨 2 캐쉬의 접근 성공률(Hit Ratio)을 높이고 전반적인 다중 프로세서 시스템의 성능 향상에 기여하고자 한다. 3.1절에서는 상용화된 시스템에서 이용되는 LRU 교체 정책 사용시 시스템 성능에 끼치는 영향에 대해 서술하고, 3.2절에서는 프로세서들의 동적인 원격 메모리 접근 패턴을 이용한 새로이 제안된 교체 정책을 구현하는 방안에 대해 제시한다. 본 논문에서 제안하는 교체 정책을 uPL 캐쉬 교체 정책(Cache Replacement Policy using Processor Locality)이라 한다.

3.1 LRU 캐쉬 교체 정책시 왜곡된 LRU 정보

LRU 캐쉬 교체 정책은 단일 프로세서에서 단일 단계의 캐쉬에서 사용되도록 고안되었다. 따라서 LRU 교체 정책은 다단계 캐쉬 내포성을 유지하는 다단계 캐쉬에서 정확한 메모리 접근의 LRU 정보를 가질 수 없다. 즉 각 단계의 캐쉬가 프로세서로부터 직접 발생한 모든 메모리 접근 내용을 알고 라인 교체를 수행할 때 가장 효율적인 교체가 이루어진다. 하지만 다단계 캐쉬에서 다단계 캐쉬 내포성을 유지할 시, 상위 단계에 대한 프로세서 접근 정보가 하위 단계 캐쉬에는 전달되지 않으므로 각 캐쉬 간에 LRU 정보가 같지 않을 수 있다. 즉 캐쉬의 단계가 증가하고 나중 쓰기(Write-Back) 정책을 사용함에 따라 프로세서로부터 멀리 떨어진 하위 단계 캐쉬는 상위 단계 캐쉬의 LRU 정보와 상이한 LRU 정보를 유지하고, 각 단계 캐쉬는 자신의 지역 LRU 정보에 따라서 교체를 수행하기 때문이다. 이는 곧 성능 향상의 여지가 남아있다는 것을 의미한다. 지역 LRU 정보란 다중 프로세서로 이루어진 다단계 캐쉬 구조에서 상위 캐쉬는 직접 자신을 접근하는 프로세서에 의해 캐쉬 교체를 하는데, 이 때 그 프로세서의 접근에 의한 LRU 정보만을 말한다. 그 노드 내의 모든 프로세서들의 접근 시간에 따른 전역 LRU 정보는 지역 LRU 정보와 다를 수 있다.

이를 설명하기 위해 다음과 같은 노드 내의 간단한 메모리 접근 패턴을 예로 들기로 한다. 그림 2에서 보는 것과 같이 레벨 2 캐쉬와 원격 캐쉬에서 LRU 교체 정책을 사용할 경우 상이한 LRU 정보를 가지는 것을 볼 수 있다. 그림 2는 레벨 1캐쉬는 직접 연관, 레벨 2 캐쉬와 원격 캐쉬는 4-way 집합 연관성을 가정한 것이다.

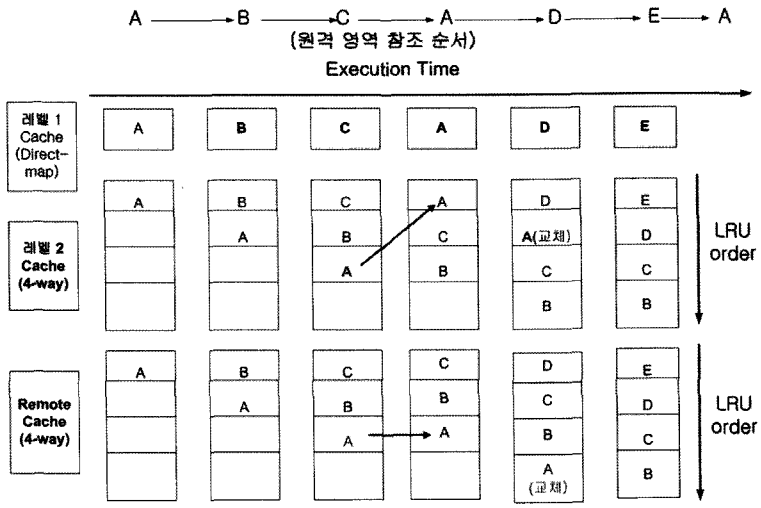


그림 2 LRU에 의한 캐쉬 교체 시나리오

그림 2를 보면 다중 캐쉬 간에 상이한 LRU 정보로 인해 상위 레벨 캐쉬에서는 시간적 지역성을 가지고 있는 캐쉬 라인 A가 하위 레벨 캐쉬에서 교체 대상이 되어 상위 레벨 캐쉬로의 연쇄적 무효화를 발생시키는 것을 보여준다. 그림 2에서 마지막 원격 메모리 접근 주소인 A는 레벨 2 캐쉬에서의 LRU 정책이었다면 여전히 교체되지 않고 접근 실패가 발생하지 않았을 것을 다단계 내포성 유지를 위해 원격 캐쉬의 왜곡된 LRU 정보에 의해서 교체됨으로써 레벨 2 캐쉬의 캐쉬 적중률을 떨어뜨리는 원인이 된다. 이는 곧 다중 프로세서 시스템의 전체 시스템 성능 저하의 원인이 되기도 한다. 그림 1에서 보는 것과 같이 다중 프로세서 시스템의 경우 원격 캐쉬를 한 노드 내의 여러 프로세서가 공유하므로, 원격 캐쉬에서의 라인 교체는 다단계 내포성 유지를 위해 상위 프로세서 캐쉬들에게 유효한 캐쉬 라인에 대한 무효화를 강제로 발생시킬 경우가 더 많을 것이다. 관련 연구 논문에 의하면 다단계 내포성을 이용하는 다단계 캐쉬 구조에서 LRU 교체 정책은 프로그램의 특성에 따라 왜곡된 LRU 정보가 아닌 완벽한 정보를 이용해 캐쉬 교체 정책을 폈을 때에 비해 11~23% 성능 향상의 여지가 남아있다고 한다[9].

한편, 원격 캐쉬를 가진 분산 메모리 구조 다중 프로세서 시스템에서 원격 캐쉬는 전체 시스템의 성능에 중요한 영향을 미친다. 원격 메모리 접근 지연은 지역 메모리 접근 지연에 비해 수백 배에 달하므로 원격 캐쉬의 이용률이 높아질수록 전체 성능은 높아질 것임에 틀림없다. 원격 캐쉬는 프로세서와 바로 근접하는 상위 단계 캐쉬와는 달리 빠른 접근 속도를 요구하지 않는다. 하지만 다중 프로세서 시스템에서 수행되는 프로그램들은 대부분 처리해야 될 데이타의 양이 대단히 크기 때문에 원격 캐쉬도 효율적으로 이용되지 못한다면 다중 프로세서 시스템의 성능을 크게 향상시킬 수 없다.

3.2 uPL 캐쉬 교체 정책

다중 프로세서 시스템에서 메모리 참조 패턴을 보면 프로세서 지역성(Processor Locality)이 있다. 프로세서 지역성이란 공유 메모리의 특정 데이타에 대해 하나의 프로세서가 다른 프로세서의 접근 없이 일정시간 동안 독자적으로 접근하는 메모리 참조 경향을 말한다[12,13]. 특히 이러한 경향은 쓰기 연산에 있어서 자주 나타나는데, 이 쓰기 연산이 프로세서 지역성을 가지는지의 여부에 따라 다음과 같이 다른 일관성 프로토콜이 사용된다. 즉 프로세서 지역성이 있는 응용 프로그램들은 한번 참조하면 오랜 시간동안 참조하고 있기 때문에 무효화 기반(Invalidate-Based) 프로토콜이 적당하고, 이러한 경향이 없는 응용 프로그램들은 공유 메모리 데이타를 빠른 시간 내에 다른 프로세서에 의해 접근이 일어나기 때문에 업데이트 기반(Update-Based) 프로토콜이 적당하다.

프로세서 지역성을 이용한 캐쉬 교체 정책(uPL 캐쉬 교체정책)이란 여러 프로세서 및 프로세서 캐쉬들에게 공유되며 낮은 단계의 캐쉬로 동작하는 원격 캐쉬에서 프로세서 지역성을 도출한 교체 정책을 적용함으로써, 첫째 원격 캐쉬의 적중률을 향상시키고, 둘째 다단계 캐쉬 내포성으로 인한 상위 단계 프로세서 캐쉬의 무효화 및 이로 인한 상위 단계 캐쉬 실패를 최소화시킴으로써 원격 메모리 접근 횟수를 줄이는 데 효과적인 방법이다.

앞서 언급하였듯이 분산 메모리 구조 다중 프로세서 시스템의 전체 시스템 구조는 간단히 나타내면 그림 1

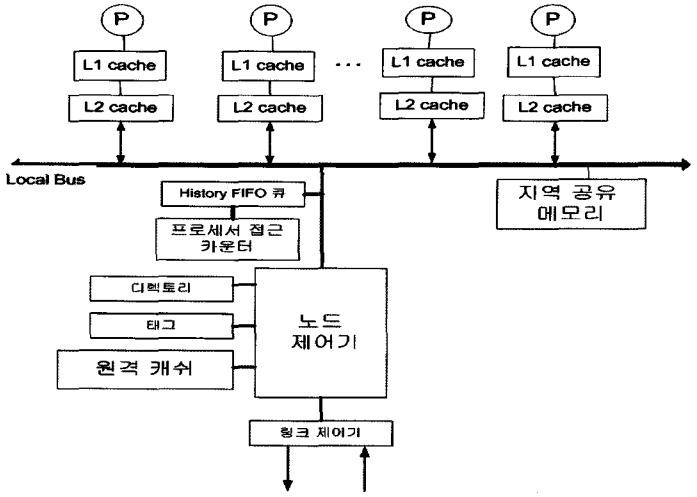


그림 3 uPL 캐쉬 교체 정책을 위한 하드웨어 구조

과 같다. 하지만 uPL 캐쉬 교체 정책을 적용하기 위해 추가된 하드웨어 구조를 포함한 노드 형태는 그림 3과 같다. 하나의 노드 내에 여러 개의 프로세서가 있고, 각 노드마다 지역 메모리를 가지고 있다. 이 그림에서 보이는 원격 캐쉬는 노드 내에 있는 모든 프로세서에 의해 공유되어 사용된다. uPL 캐쉬 교체 정책은 프로그램을 실행하는 도중 캐쉬 교체가 필요한 시점에 프로세서들의 원격 메모리 접근 패턴을 분석한다. 노드 내의 모든 프로세서 중에서 어느 한 프로세서의 원격 메모리 접근 횟수가 빈번하여 원격 캐쉬 이용률이 높을 경우, 이 프로세서에게 우선권(Priority)을 주어 LRU에 의해서 쫓겨나지 않게 하여 상위 단계의 캐쉬 적중률을 높이기 위한 정책이다.

uPL 캐쉬 교체 정책 알고리즘의 이해를 돕기 위해 프로그램 실행시 원격 캐쉬와 프로세서들의 레벨 2 캐쉬의 내용 변화를 도시한 그림 4와 그림 5를 설명하면 다음과 같다. 그림 4는 LRU 캐쉬 교체 정책을 실시하였을 경우의 원격 캐쉬와 레벨 2 캐쉬의 내용 변화를 보이고, 그림 5는 본 논문의 프로세서 지역성을 이용한 교체 정책이 사용될 경우 원격 캐쉬와 레벨 2 캐쉬의 내용 변화를 도시한 예이다. 그림 4와 그림 5에서 한 노드 내에 존재하는 4개의 프로세서를 P_0, P_1, P_2, P_3라고 하자. 레벨 1 캐쉬의 집합 연관성은 다이렉트-맵이고, 원격 캐쉬와 레벨 2 캐쉬의 집합 연관성은 4-way로 구성되어 있다. 프로그램이 실행되는 도중 원격 캐쉬에서 충돌 실패가 발생하여 교체될 캐쉬 라인을 찾고 있다. 그림 4와 그림 5에서 원격 메모리 라인 A에 대한 접근이 레벨 2 캐쉬와 원격 캐쉬간에 상이한 LRU 정보를 보이고 있다. 그림 4에서 P_0 프로세서가 요구하는 원격 메모리 라인 B를 위한 공간을 할당하기 위해 원격 캐쉬의 LRU 라인인 A 라인을 교체 대상으로 선택해야 한다.

이후 P_0 프로세서에서 A 라인은 다시 요청하는데 이미 원격 캐쉬에서 쫓겨났고, 내포성을 유지하기 위해 P_0 프로세서의 레벨 2 캐쉬에서도 쫓겨났기 때문에 원격 메모리로부터 다시 읽어 와야 한다. 이처럼 원격 캐쉬의 LRU 라인인 A 라인 교체는 노드 내 프로세서 P0에게는 다시 사용될 수 있는 라인임에도 불구하고 무효화가 되어 P_0 프로세서 상위 단계 캐쉬의 적중 실패율을 높이는 원인이 된다.

반면에 LRU 캐쉬 교체 정책과는 달리 본 논문이 제시하는 교체 정책에서는 그림 5와 같이 이루어진다. 충돌 미스가 발생하여 캐쉬 교체 시점에 P_0 프로세서가 나머지 세 개의 프로세서 P_1, P_2, P_3보다 원격 캐쉬에 대한 접근이 빈번하다고 가정하자. 여기서 P_0 프로세서가 원격 캐쉬에 대해 접근이 빈번하다는 의미는 어느 과거시점에서 가까운 과거동안 나머지 프로세서에 비해 원격 메모리에 대한 접근이 많았다는 것을 뜻한다. 프로세서의 원격 메모리 접근 빈번성에 대한 정보 추출은 앞으로 상세히 설명될 추가된 하드웨어 구조로 이루어진다. 이는 곧 그 시점에서 원격 캐쉬에 접근이 많은 프로세서 P_0는 원격 메모리에 대한 참조가 많고 이렇게 하여 불린 메모리 영역은 시공간적 지역성이 있어 상위 단계 캐쉬에서는 사용되어 LRU 정보가 갱신되었으나, 가장 하위 단계에 있는 원격 캐쉬에는 그 후 프로세서의 메모리 접근 정보가 갱신되지 않아 LRU 라인으로 왜곡되어 교체 대상이 되는 것으로 생각할 수 있다. 그림 5에서처럼 프로세서 P_0가 원격 메모리 라인 B 라인을 요청할 때, 원격 캐쉬의 LRU 라인인 A 라인을 바로

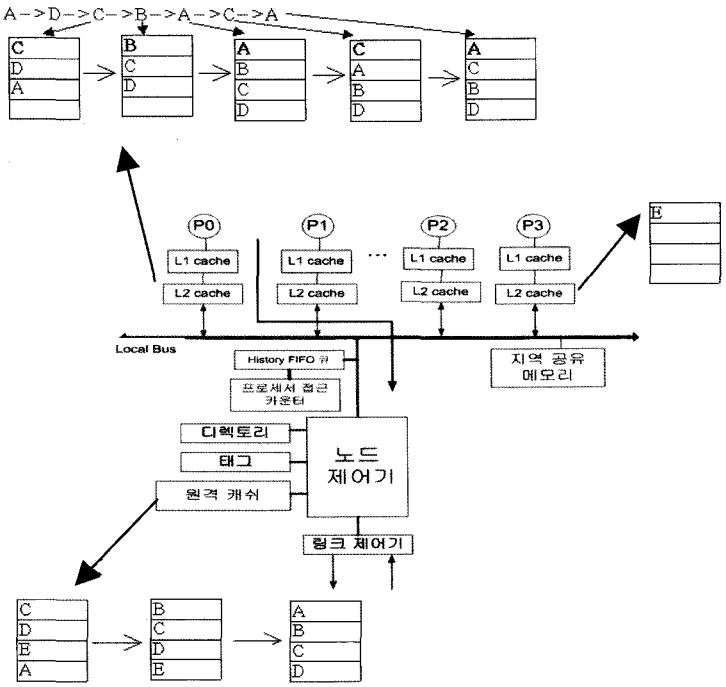


그림 4 LRU 캐쉬 교체 정책시 원격 캐쉬와 레벨 2캐쉬 상태 변화(메모리 접근 순서 :A(P0)->E(P3)-> D(P0)->C(P0)->B(P0)->A(P0)->C(P0)->A(P0))

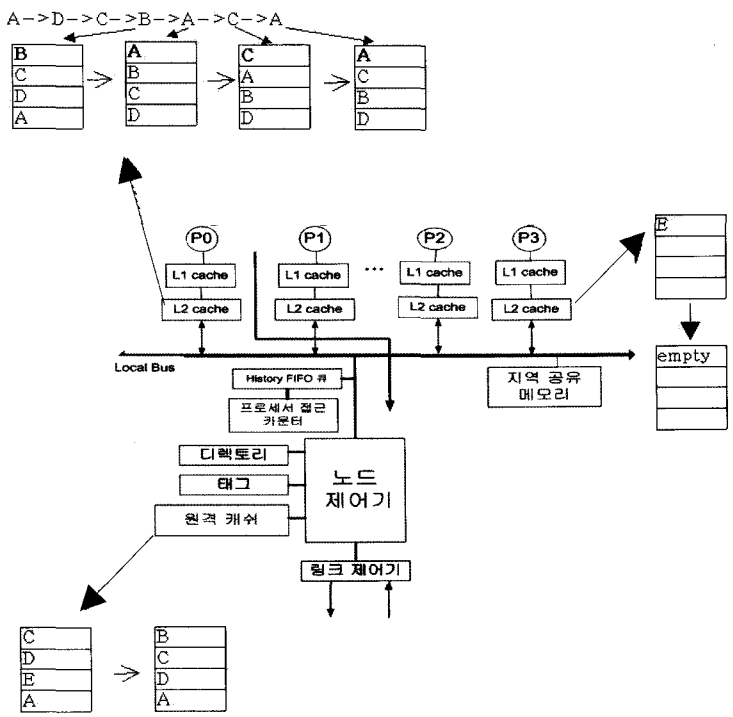


그림 5 uPL 캐쉬 교체 정책시 원격 캐쉬와 레벨 2 캐쉬 상태 변화(메모리 접근 순서 :A(P0)->E(P3)-> D(P0)->C(P0)->B(P0)->A(P0)->C(P0)->A(P0))

교체하는 것이 아니라 uPL 캐쉬 교체 정책에 의해 왜곡된 LRU 교체 라인보다는 원격 메모리 참조 경향이 낮은 프로세서인 P_3 프로세서의 참조 라인인 E 라인을 교체하여 P_0 프로세서에서 다시 사용되는 A 라인을 무효화시키지 않는다. 이처럼 한 프로세서의 지역성을 이용한 노드 내의 원격 캐쉬 교체 정책은 프로세서 캐쉬에서의 전체 무효화 수와 모든 다단계 캐쉬의 적중 실패율을 감소시킴으로써 전체적인 시스템의 성능을 향상시킬 수 있다.

이와 같은 uPL 캐쉬 교체 정책을 구현하기 위해서는 두 가지 정보가 필요하다. 첫째로 원격 메모리 전체 라인에 대한 프로세서 참조 정보가 필요하다. 이를 위해서 그림 6처럼 원격 캐쉬의 각 라인에 한 노드 내의 프로세서 수만큼의 디렉토리(Directory)를 두어 각 라인에 대한 프로세서의 참조 여부를 나타낸다. 이는 프로세서가 읽기 또는 쓰기 요청을 지역 버스에 내보낼 때 어느 프로세서가 요청하는지를 나타내는 프로세서 아이디(Processor ID)를 함께 내보내는데 이를 보고 각 라인의 프로세서 공유 비트에 프로세서 아이디에 맞도록 설정하면 된다. 그림 6의 유효 비트는 해당 캐쉬 라인의 유효 여부를 의미하고, 태그 비트는 같은 집합 내의 해당 라인을 확인하기 위해 사용되는 주소 값이며, LRU 정보 비트는 한 집합의 관련 라인들의 LRU 정보를 나타내고, 데이타 비트는 프로세서에 의해서 사용되는 실제 의미 있는 데이타이다.

그림 6에서 한 집합이 4-way 연관성인 원격 캐쉬의 프로세서 접근에 따른 캐쉬 내용을 보여준다. 각 유효 비트는 두 가지 상태를 가지며, "참조"일 때 1, "비참조"일 때 0을 나타낸다. 처음에는 0 상태로 시작하여, 그 라인에 대한 요청이 있을 경우는 상태를 1로 바꿔 후에 교체 될 라인을 선택할 때 이용된다. 캐쉬 간 전송이 일어날 때에도 트랜잭션이 지역 버스를 통해 이루어지므로 스누핑을 통해 알 수 있다. 이러한 경우에도 프로세서 공유 상태를 1로 설정한다. 그림 6의 예에서처럼 P_1이 태그 D인 라인을 접근하여 사용할 경우, 프로세서 공유 비트가 P_1 프로세서만 읽어갔다는 의미를 나타내는 0100으로 설정된다. 만약 캐쉬가 교체되거나 일관성 트랜잭션에 의해 무효화가 발생하면 그 라인의 프로세서 공유 비트들은 0으로 설정된다. 그리고 요청된 메모리 라인이 노드 내 다른 프로세서 캐쉬에 의해 서비스되지 않고 원격 캐쉬에서 적중이 발생하면 이는 지역 내의 다른 캐쉬가 참조하고 있지 않음을 나타내므로 프로세서 공유 비트들을 모두 0으로 설정하고 현재 요구하는 프로세서의 공유 비트만 1로 설정한다. 여기서 각각의 프로세서 공유 비트가 0일 경우는 반드시 해당 프로세서가 이 라인을 참조하고 있지 않음을 나타내지만,

유효 비트	태그 비트	프로세서 공유 비트<4>	LRU 정보 비트	데이타 비트
1	A	P0 (1000)	1	
1	B	P1, P3(0101)	2	
1	C	P2(0010)	3	
1	D	P1(0100)	4(LRU)	

그림 6 원격 캐쉬 구조(4-way의 한 집합)

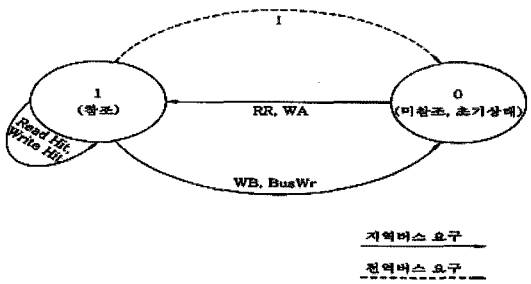


그림 7 원격 캐쉬 프로세서 공유 비트의 상태 전이도

1의 표시는 해당 프로세서가 읽어가기는 했지만 현재 프로세서가 이 라인을 가지고 있는지에 대한 여부는 알 수 없다. 그 이유는 프로세서 캐쉬에서의 교체로 인해 그 라인을 프로세서 캐쉬에서는 가지고 있지 않지만 원격 캐쉬에는 현재 프로세서 캐쉬에서 교체될 때 그 사실을 알 수 없기 때문이다. 따라서 1의 표시는 반드시 프로세서 캐쉬가 참조하고 있음을 나타내지는 않는다. 이러한 각 프로세서 별 공유 비트 상태 전이도를 그림 7에서 보여준다. 프로세서가 읽기 요구(RR)나 쓰기 할당(WA)을 할 때 원격 캐쉬에 그 라인이 적재되면 해당 프로세서의 공유 비트는 1로 설정된다. 즉 프로세서에서 읽기(RR), 쓰기(WA), 캐쉬간 전송에 따른 시스템 버스 요구(BusRd)가 오면 그 메모리 라인에 대해 해당 프로세서 공유 비트가 1로 설정된다. 이렇게 1로 설정된 해당 공유 비트는 해당 프로세서의 읽기 히트(Read Hit), 쓰기 히트(Write Hit)가 발생하면 1을 유지한다. 하지만 시스템 버스에서 타 프로세서의 되쓰기(WB), 쓰기 요구(BusWr), 전역 상호 연결망에서의 무효화(I) 등의 요구가 오면 프로세서 공유 비트는 0으로 설정된다.

둘째로 uPL 캐쉬 교체 정책을 위해 필요한 정보는 교체되는 시점에서 가까운 과거에서 그 시점까지 노드 내 프로세서들의 원격 메모리 참조 패턴(Pattern History)을 알아야 한다. 이 정보를 얻기 위해서는 동적인 패턴 분석이 필요한데 그 방법은 다음과 같다. 그림 3과 그림 8에서처럼 적당한 크기의 요청 기록 FIFO (Request History First-In First-Out) 큐를 두어 지역 버스에서 발생하는 원격 메모리 참조 프로세서들의 프로세서 아이디를 저장하면 된다. 히스토리 FIFO 큐의

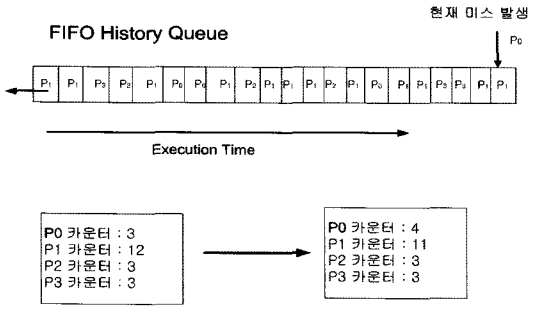


그림 8 History FIFO 큐의 구조 및 동작 예

동작 예를 보이는 그림 8에서처럼 큐가 가득 찼을 경우, FIFO 구조이기 때문에 오래된 프로세서 아이디는 큐에서 제거되고, 새로이 지역 버스에 발생하는 원격 메모리 참조 프로세서 정보가 큐에 저장된다. 이러한 정보는 동적으로 버스에 새로운 메모리 요청이 발생할 때마다 병렬적으로 변경된다. 그리고 그림 3에 나타낸 프로세서 접근 카운터라는 작은 크기의 프로세서별 카운터가 있는데, 이는 히스토리 FIFO 큐에서 각 프로세서 아이디의 수를 세는 카운터이다. 최대 하나의 프로세서만 접근하였을 경우 FIFO 큐 크기만큼 셀 수 있는 비트면 된다. 큐의 크기가 N일 경우 $2^K \geq N$을 만족시키는 가장 작은 수 K 크기의 비트가 카운터 크기이다. 이러한 카운터 변화를 도시한 그림 8에서처럼, P_0 프로세서가 접근하여 히스토리 FIFO 큐에 들어오면 P_0 카운터가 1 증가하여 3에서 4가 되고, 가장 오래되어 히스토리 FIFO 큐에서 쫓겨나는 프로세서 아이디가 P_1이라면 P_1의 카운터가 1 감소하여 12에서 11이 된다.

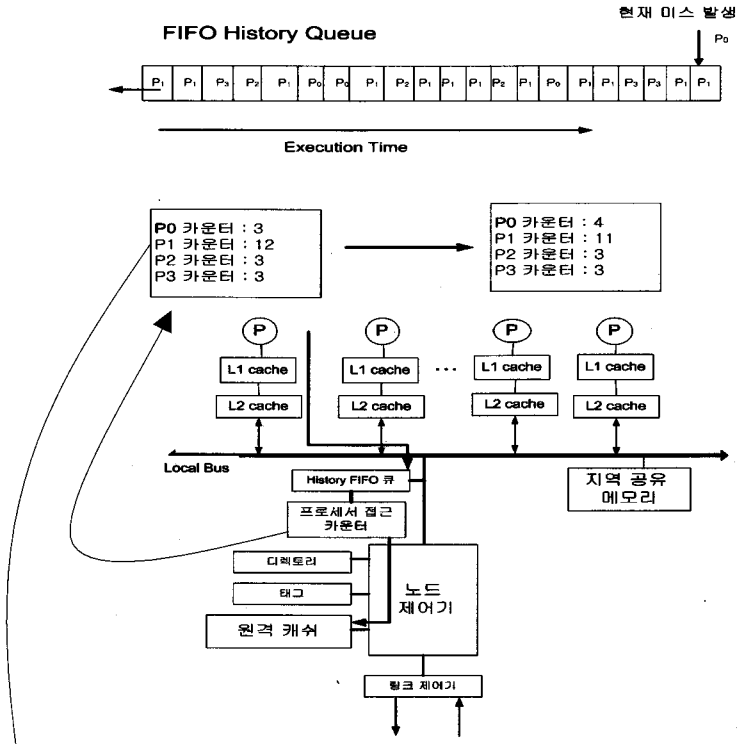
이제는 앞서 언급한 두 가지 정보를 가지고 uPL 캐쉬 교체 정책이 실제로 어떻게 동작하는지에 대해 설명한다. 그림 8은 히스토리 FIFO 큐의 동작 예를 보여준다. 원격 캐쉬에서 적중 실패가 발생하여 교체 대상을 선택해야 한다고 하자. 그림 6에 의하면 하나의 4-way 집합 연관성에서 일반적인 LRU에 의해 마지막 라인인 태그 D가 교체 대상이 된다. 하지만 uPL 캐쉬 교체 정책의 경우 그림 8에서 동적으로 수집된 그 시점의 프로세서당 원격 캐쉬 접근 기록을 보고 일반적인 LRU를 선택할 것인지, 특정 프로세서에게 우선권을 줄 것인지를 결정한 후 최종 교체 라인을 선택하게 된다. 그림 9는 uPL 캐쉬 교체 정책이 실행될 때 노드 내에서의 한 동작 예를 보여준다. 그림 9에서 P_1의 접근 횟수가 히스토리 FIFO 큐의 최대 크기 21 중에 12번으로 P_0, P_2, P_3에 비해 그 기간 동안 많은 접근이 이루어졌음을 알 수 있다. 여기서 P_1이 참조하고 있는 원격 캐쉬 라인에 대해 한 번 더 기회를 줄 것인가의 결정은 특정 임계값(Threshold)을 두어 그 값을 넘겼을 경우 우선권을 준다. 임계값은 시스템에 따라 다르게 결정되어질 수 있다. 가령 임계값을 11로 정할 경우, 21개의 큐 크기 중에 P_1이 11번 접근하였으므로 4개의 프로세서 중에 원격 캐쉬에 대한 접근이 다른 프로세서의 평균 원격 메모리 접근 수의 3배 정도 되는 값이므로, 교체 라인이 P_1 프로세서가 참조한 라인이라면 한 번 더 기회를 주는 것을 뜻한다. 그림 9에서 LRU에 의해 선택된 교체 라인 D의 프로세서 공유 비트를 관찰 후 P_1이 참조하고 있는 경우, 현재 P_1의 원격 캐쉬 접근 수가 임계값을 넘었으므로 P_1에게 우선권을 주어 P_1을 포함하지 않는 다음 LRU 라인인 C를 선택한다. 여기에서 LRU인 태그 D가 P_1 프로세서의 참조로 인해 캐쉬에 좀 더 오래 남을 수 있는 특혜를 받았으므로 다음 번 교체 시에는 재차 그런 특혜를 받는 것을 막기 위해 이 라인의 프로세서 공유 비트의 P_1을 나타내는 프로세서 공유 비트를 0으로 설정한다. 이는 적절한 길이의 시간적 지역성을 유지해주면서 LRU를 따르기 위한 원리이다.

다음 LRU 라인인 태그 C는 P_2가 참조한 라인이고, 현재 P_2의 원격 캐쉬 접근 수가 3이므로 이를 교체 대상으로 선택한다. 그리고 여기서 모든 집합 내의 라인에 대해 이를 적용하면 가장 최근의 라인이 교체되어 성능 저하의 원인이 될 수 있다. 따라서 MRU인 태그 A는 교체 후보에서 제외한다. 보통 MRU 라인은 시간적 지역성이 크기 때문에 제외하는 것이 LRU 교체 정책의 장점을 살려주기 때문이다. 만약 집합 내의 모든 라인이 P_1에 의해 참조되는 것이라면 기존 LRU 교체 정책에 의해 교체 대상이 선택된다. History FIFO 큐와 프로세서 접근 카운터의 추가는 하드웨어 복잡성은 증가하였지만, 각 유닛의 작업은 버스의 데이타 흐름과 병렬적으로 수행되기 때문에 지연시간의 증가는 없다. 하지만 원격 캐쉬의 교체 라인 선택 시 곧바로 LRU 라인을 선택하지 않고 추가적인 작업이 수행되므로 지연시간이 길어질 수 있다. 하지만 이 uPL 교체 정책은 특정 프로세서의 접근 빈도율이 높을 때 수행되고, 이로 인해 원격 캐쉬의 적중률 미스를 줄일 수 있기 때문에 이득이 있다. 이처럼 uPL 캐쉬 교체 정책은 프로세서들의 원격 메모리 접근 지역성을 이용한 것으로서 임의 교체보다 실제 프로세서의 원격 캐쉬 접근 경향을 바탕으로 LRU 캐쉬 교체 정책을 효율적으로 이용함으로써 상위 단계 캐쉬의 불필요한 무효화를 줄여 적중률을 높이고, 이를 바탕으로 전체 시스템의 성능 향상을 가져올 수 있다.

4. 성능 평가(Performance Evaluation)

4.1 모의실험 환경

본 논문에서 제안한 접근 지역성을 이용한 원격 캐쉬 교체 정책을 평가하기 위해서 MINT 시뮬레이터를 사용



유효 비트	태그 비트	프로세서 공유 비트<4>	LRU 정보 비트	데이터 비트
1	A	P0 (1000)	1	
1	B	P1, P3(0101)	2	
1	C	P2(0010)	3	
1	D	P1(0100) ->0000	4(LRU)	

그림 9 uPL 캐쉬 교체 정책이 적용된 동작 예

하여 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[14].

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 캐쉬는 4-way 집합 연관성이라 가정한다. 여러 시스템 인자 값은 표 1과 같다.

4.2 벤치마크 프로그램

벤치마크용 프로그램은 SPLASH-2에서 제공하는 프로그램 중 FFT, Radix, LU, Barnes를 사용하였다[15].

표 1 모의실험 인자

인자	값
프로세서 수	32
한 노드내의 프로세서 수	4
노드 수	8
프로세서 캐쉬의 크기	16KB
원격 캐쉬의 크기	64, 128, 256, 512KB
프로세서 클럭 속도	500Mhz
노드 간 링 연결 클럭 속도	500Mhz
프로세서 캐쉬 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

표 2 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)

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. 마지막으로 Barnes는 Barnes-Hut Hierarchical N-body Problem 방식을 사용하여 각 입자 사이의 상호 작용을 구하는 프로그램이다. 프로그램의 인자들은 표 2와 같다.

4.3 모의 실험 결과 및 분석

4.3.1 노드 내 프로세서들의 원격 메모리 접근 패턴

우선 성능 분석에 들어가기 전에 성능 향상을 위한 응용 프로그램들의 프로세서별 원격 메모리 참조 패턴을 분석하였다. 특정 시점에 특정 프로세서가 원격 캐쉬를 독점하는 경향이 있다면, uPL 원격 캐쉬 교체 정책은 성능 향상을 가져 올 수 있기 때문이다. 이를 분석하기 위해 크기가 20인 동적 FIFO 큐(Threshold=10)를 두어 실험하였다. 앞서 제시한 SPLASH-2 응용 프로그램들의 전체 프로그램 실행동안의 원격 메모리 참조 패턴을 분석한 결과가 그림 10에서 그림 13까지 제시되어 있다.

LU는 그림 10에서와 보는 것과 같이 전체 프로그램이 실행되는 동안 특정 시점에 노드내의 한 프로세서가 원격 메모리를 독점하는 경향이 있다. 첫 시작 점에서는 P0 프로세서가 거의 원격 메모리 접근이 이루어지고 있는데, 이는 응용프로그램의 초기화 과정을 무시하더라도 프로그램 전반에 걸쳐서 P_1 프로세서가 일정 주기로 원격 메모리 접근 하는 것을 보여주고 있다. 앞서 살펴보았듯이 이처럼 한 프로세서의 원격 캐쉬에 대한 지역성이 높을 때 다른 프로세서가 참조한 캐쉬 라인은 P_1 프로세서의 지역성 이용을 떨어뜨린다. FFT의 프로세서별 원격 메모리 참조 패턴은 그림 11에 나타나 있는 것처럼 노드 내 모든 프로세서들이 균일하게 접근하는 경향을 보여준다. 이러한 경우에서는 uPL 교체 정책의 영향이 미미할 수 있다. Radix는 그림 12에서 보는 것과 같이 처음에는 P_3 프로세서가 100% 독점적으로 원격 메모리 접근을 하고 중간 중간 균일하게 접근하는 경향을 보여주었다. 그림 13을 보면, Barnes도 전체적으로 4개의 프로세서가 균등하게 원격 메모리 접근을 보여주는데 중간 중간 P_0 프로세서에 의해 독점되는 것을 볼 수 있다. 이러한 프로세서들의 원격 캐쉬 접근 패턴은

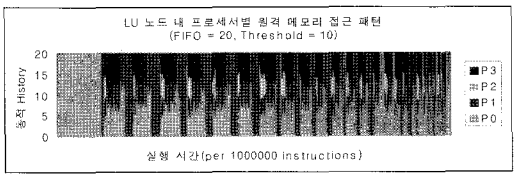


그림 10 LU 한 노드 내 프로세서별 원격 메모리 접근 패턴

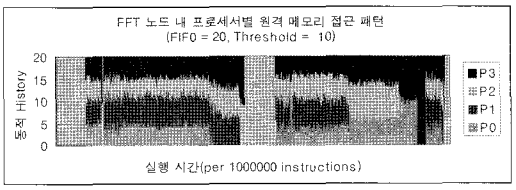


그림 11 FFT 한 노드 내 프로세서별 원격 메모리 접근 패턴

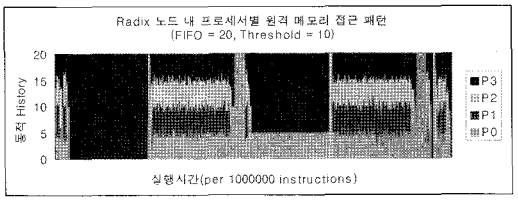


그림 12 Radix 노드 내 프로세서별 원격 메모리 접근 패턴

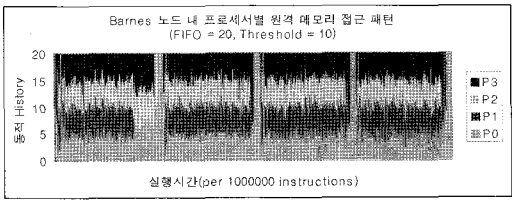


그림 13 Barnes 노드 내 프로세서별 원격 메모리 접근 패턴

uPL 캐쉬 교체 정책으로 성능 향상을 가져 올 수 있는 부분임을 보여준다.

또한, 이상의 프로그램별 원격 메모리 접근 경향 그림들은 분석의 편의를 위해 긴 시간 간격으로 분석한 것이라, 자세히 나타나 있지 않지만 세밀한 분석결과 실행 도중에 많은 부분에 있어서 한 프로세서의 독점 경향이 나타났다. 이는 곧 uPL 캐쉬 교체 정책의 유효성을 보여주는 메모리 접근 경향이라 할 수 있다.

4.3.2 동적 History FIFO 큐 크기 결정

최대한 많은 프로그램의 성능을 골고루 향상시키고 큰 성능 향상, 즉 프로세서 캐쉬의 유용한 무효화를 줄여 전체 시스템의 성능을 향상시키기 위한 동적 History FIFO 큐 크기를 결정하여야 한다. FIFO 큐도 하

드웨어 비용과 복잡성을 증가시키는 원인이 되므로 크기를 최대한 줄이면서 큰 효과를 내는 크기를 찾아야 한다.

그림 14는 동적 History 큐 크기를 10, 15, 20, 30, 50으로 증가시켜 가며 각 응용 프로그램의 성능 향상을 LRU 캐쉬 교체 정책에 정규화하여 비교하였다. 그림 14에서 보는 것과 같이 너무 큰 History 큐는 오래된 정보를 원격 메모리 접근 정보로 이용함으로써 잘못된 예측이 될 수 있다. 그 예로 Barnes에서 큐 크기가 50일 때 성능이 다소 나빠진 것을 볼 수 있다. 본 논문에서는 큐 크기가 20일 때 전체적으로 성능 향상이 크게 이루어진 것을 바탕으로 앞으로의 실험은 History 큐 크기를 20으로 하였다. 그리고 큐 크기가 주어졌을 때, 프로세서에게 특혜를 주는 한계 값은 여러 번의 실험을 통해 전체 큐 크기의 절반으로 정하였다. 그림 15에 보이는 바와 같이 임계 값이 10일 때 가장 성능이 우수한 것을 볼 수 있다. 임계 값이 너무 작을 경우, 특정 프로세서의 접근 경향 즉 빈번하게 접근한다는 것을 너무 자주 특혜를 주게 되는 결과를 나으므로 성능 향상에 영향을 끼치지 않음을 보여준다. 그리고 너무 클 경우, 상대적으로 우선권을 가지는 프로세서의 빈도가 많지 않아 성능 향상에 크게 기여하지 못한다. 이러한 정보들을 바탕으로 앞으로의 실험들은 큐 크기가 20일 경우 10이상의 접근 패턴을 보일 때 그 프로세서에게 우선권을 주기로 한다.

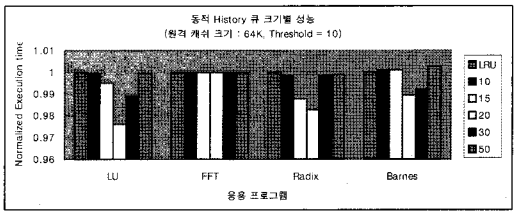


그림 14 동적 History 큐 크기별 성능 향상

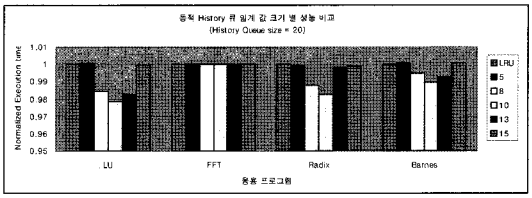


그림 15 동적 History 큐 임계 값 크기 별 성능 비교

4.3.3 프로세서 캐쉬의 무효화 개수 비교

이 절에서는 기존의 LRU 교체 정책과 uPL 교체 정책의 프로세서 캐쉬의 무효화 개수를 실험을 통해 비교한다. 여기서 무효화 개수는 일관성 프로토콜을 유지하기 위해 무효화를 시킨 횟수가 아니라, 원격 캐쉬에서 교체가 발생하여 프로세서 캐쉬에 가해지는 무효화의 수만을 측정한 것이다. 이는 프로세서 캐쉬와 원격 캐쉬의 왜곡된 LRU 정보로 인해 시간적 지역성을 지닌 프로세서 캐쉬의 라인이 무효화됨으로써, 프로세서 캐쉬에

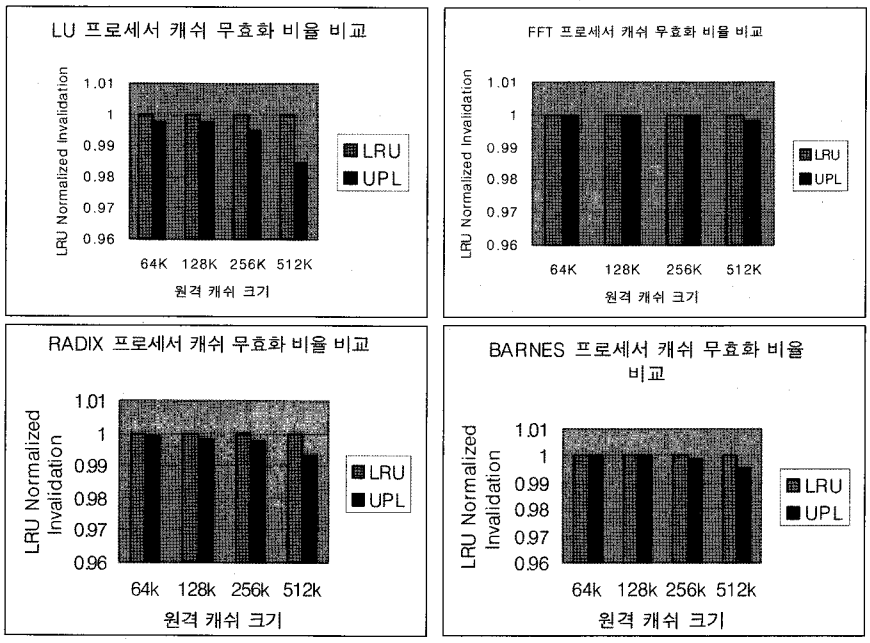


그림 16 프로세서 캐쉬의 무효화 비교

충돌 미스(Conflict Miss)를 일으켜 적중률을 떨어뜨리는 원인이 되기 때문이다.

각 벤치마크 프로그램의 프로세서 캐쉬의 무효화 수를 LRU 교체 정책에 정규화 하여 나타내었다. 그림 16를 보면 우선 모든 응용 프로그램 중에 LU가 0.02%에서 1.6% 정도로 다른 프로그램에 비해 무효화 수가 많이 줄어든 것을 볼 수 있다. 그리고 원격 캐쉬 크기가 증가할수록 전체 프로그램의 원격 캐쉬 교체로 인한 프로세서 캐쉬의 무효화 수가 LRU 교체 정책에 비해 크게 감소하고 있는 것을 볼 수 있다. LU 다음으로 Radix와 Barnes 프로그램의 무효화 수가 크게 감소되었고 전체 프로세서의 균등한 접근이 이루어지는 FFT는 uPL 캐쉬 교체 정책의 효과를 받지 못하여 무효화 감소가 미미한 것을 알 수 있다.

4.3.4 프로세서 캐쉬의 적중률 비교

이 절에서는 앞 절의 프로세서 캐쉬의 무효화 감소로 인한 프로세서 캐쉬의 적중률 증가를 LRU와 비교하고자 한다. 그림 17에서처럼 LU는 무효화 개수가 줄어든 만큼 원격 캐쉬 크기가 증가할수록 프로세서 캐쉬의 적중률(Hit Rate)이 증가함을 볼 수 있다. 프로세서 캐쉬의 적중률의 경우 보통 90%가 넘는 적중률을 보이기 때문에 uPL 교체 정책에 의한 향상된 적중률을 정규화하면 평균 0.2%정도의 상승을 보여준다. 수치자체는 작지만 높은 적중률에서는 의미있는 숫자이다. FFT는 무효화 개수에서도 변화가 없었듯이 프로세서 캐쉬 적중률에서도 LRU와 비슷한 적중률을 보였다. 이는 FFT의 특성상 프로그램 전체 실행에 있어서 노드 내의 프로세서들이 균등하게 원격 메모리에 접근하는 경향을 가지기 때문에 uPL 교체 정책의 효과를 볼 수 없음을 보여준다. Radix도 LRU 교체 정책과 비교해 프로세서 캐쉬 적중률의 상승이 있는 것을 볼 수 있다. Barnes 경우 256K 크기의 원격 캐쉬에서 0.3%정도의 상승을 볼 수 있는데 분석결과 256K 크기에서 LRU의 프로세서 캐쉬 적중률이 상대적으로 크게 감소하여 uPL 교체 정책의 적중률이 높아졌다. 이처럼 uPL 교체 정책은 프로세서 캐쉬의 불필요한 무효화를 줄여 프로세서 캐쉬의 적중률을 높이는데 기여할 수 있었다.

4.3.5 원격 캐쉬 적중률 비교

이 절에서는 uPL 교체 정책을 사용했을 경우의 원격 캐쉬 적중률를 비교한다. 그림 18에서 보는 것처럼 전반적인 원격 캐쉬의 적중률이 uPL 정책의 사용으로 상승한 것을 볼 수 있다. LU, Barnes, Radix는 프로세서 캐쉬의 무효화 수 감소와 프로세서 캐쉬의 적중률 상승으로 인해 원격 캐쉬로의 접근 수가 줄어들었고, uPL 캐쉬 교체 정책으로 인해 특정 프로세서의 원격 영역에 대한 시공간적 지역성의 증가로 원격 캐쉬의 적중률 상승을 가져왔다. 원격 캐쉬 적중률 증가는 대체로 2%에서 10%까지 높은 향상을 보였다. 이 숫자의 의미는

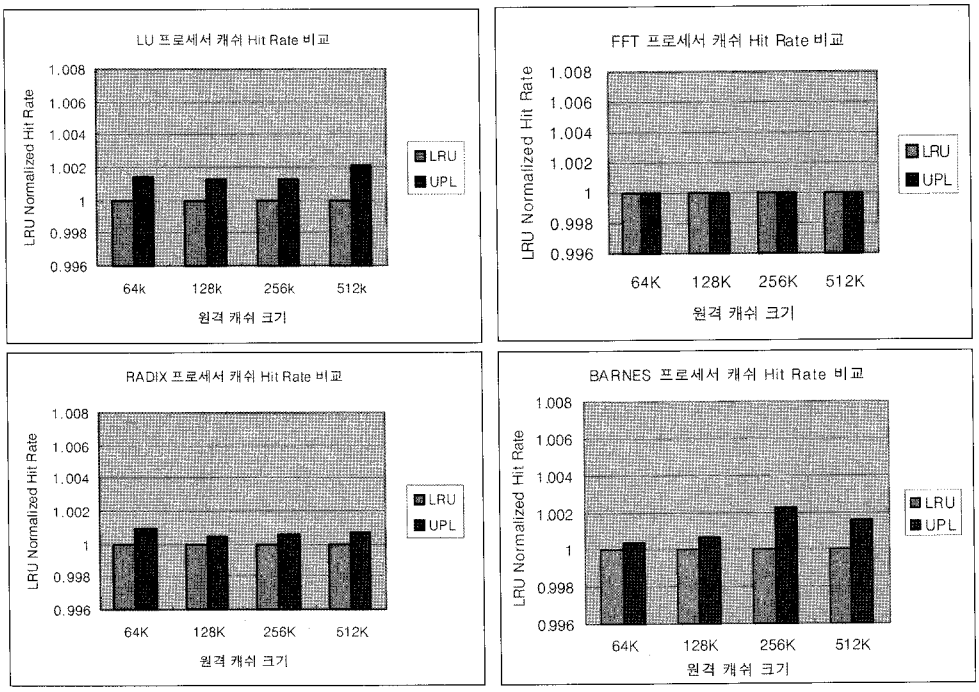


그림 17 프로세서 캐쉬의 적중률 비교

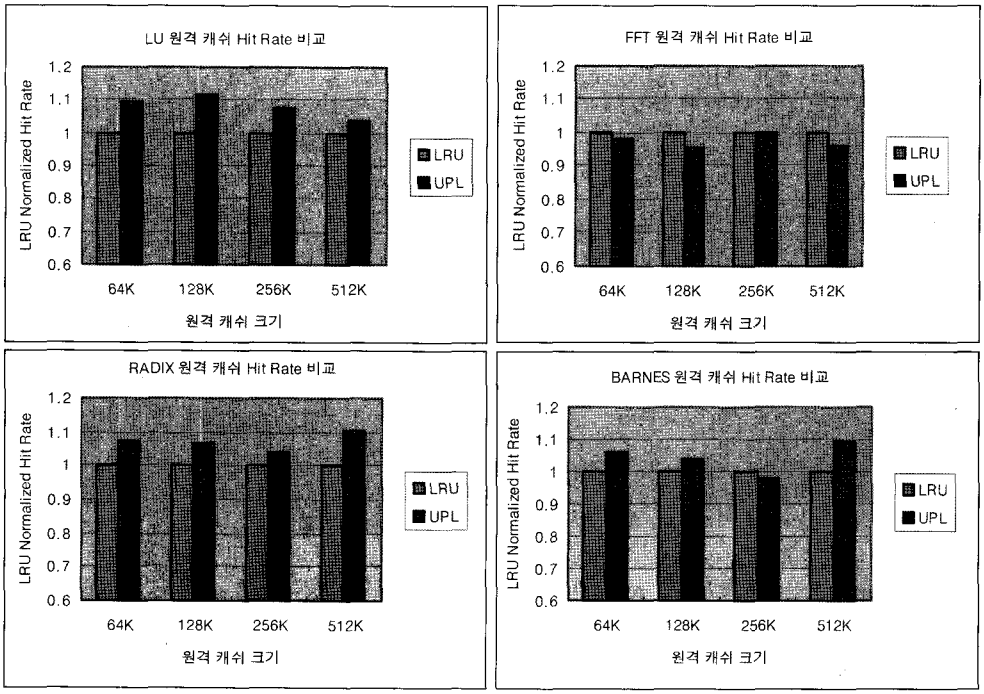


그림 18 원격 캐쉬 적중률 비교

LRU 캐쉬 교체 정책에 정규화 한 값이다. 하지만 FFT 경우 uPL 캐쉬 교체 정책의 효과를 낼 수 없는 원격 메모리 접근 경향에 의해서 적중률이 그대로이거나 약간의 감소가 있었다. Barnes 경우 원격 캐쉬 크기가 256K일 때 프로세서 캐쉬 적중률과는 달리 크게 떨어졌는데 이때에는 LRU 캐쉬 교체 정책으로 실행한 결과가 원격 캐쉬 적중률이 예외적으로 증가하여 이런 결과가 나왔다. 이상에서 보는 바와 같이 대체적으로 uPL 교체 정책은 프로세서 캐쉬에서의 불필요한 무효화 감소와 이로 인한 프로세서 캐쉬 적중률 상승과 원격 캐쉬의 적중률 상승을 가져와 전체 시스템 성능 향상에 기여하였다.

4.3.6 수행 시간 비교

이 절에서는 캐쉬 크기가 증가함에 따라 각 응용 프로그램의 수행 시간을 비교한다. 그림 19에서 보는 것처럼 벤치마크 프로그램당 평균적으로 LRU에 비교하여 2% 정도 향상된 것을 볼 수 있다. LU와 같이 특정 프로세서의 원격 메모리 접근 경향이 강한 응용 프로그램의 경우 최대 3.5%의 성능 향상을 보여주었다. 원격 메모리에 대한 접근 패턴이 균등한 프로그램인 FFT의 경우는 성능 향상이 없거나 약간의 성능 저하가 있었다. 4.3.1절에서 보았듯이 LU는 프로세서의 원격 메모리 접근이 한 프로세서에 의해 일정 시간 한 프로세서를 제외한 나머지는 모두 대기 상태에 있는 경향이 많은 프로그램으로서 원격 캐쉬 크기가 증가함에 따라 점진적으로 성능 향상을 보여주고 있다. 그림 19를 보면 LRU 캐쉬 교체 정책에 비교해서 2.2~3.5% 정도 실행 시간이 줄어들었음을 알 수 있다. 또한 Radix나 Barnes의 경우도 LRU 캐쉬 교체 정책과 비교하여 성능이 향상되었음을 보여 주었다. Radix와 Barnes 프로그램은 프로그램 전반적으로 특정 프로세서의 독점현상은 없으나 프로그램 실행 도중 일시적으로 노드 내 한 프로세서의 의한 독점 현상이 일어남을 확인할 수 있었다. 이러한 원격 메모리 접근 경향은 uPL 캐쉬 교체 정책에 의해 LRU 캐쉬 교체 정책을 이용한 때보다 1.2%에서 2.5%까지의 성능 향상의 원인으로 작용하였다. 하지만 FFT의 경우는 전체적인 프로그램의 실행이 노드 내의 모든 프로세서들이 골고루 원격 메모리 접근 경향을 가지기 때문에 uPL 캐쉬 교체 정책이 성능 향상에 영향을 끼치지 못함을 보여주고 있다.

5. 결 론

원격 캐쉬는 분산 메모리 구조 다중 프로세서 시스템에서 원격 메모리 접근을 최소화하기 위해 제안되었다. 원격 캐쉬는 각 프로세서의 상위 캐쉬들과 다단계 캐쉬 구조로 구성되며, 스누프 필터링을 위해서는 일반적으로

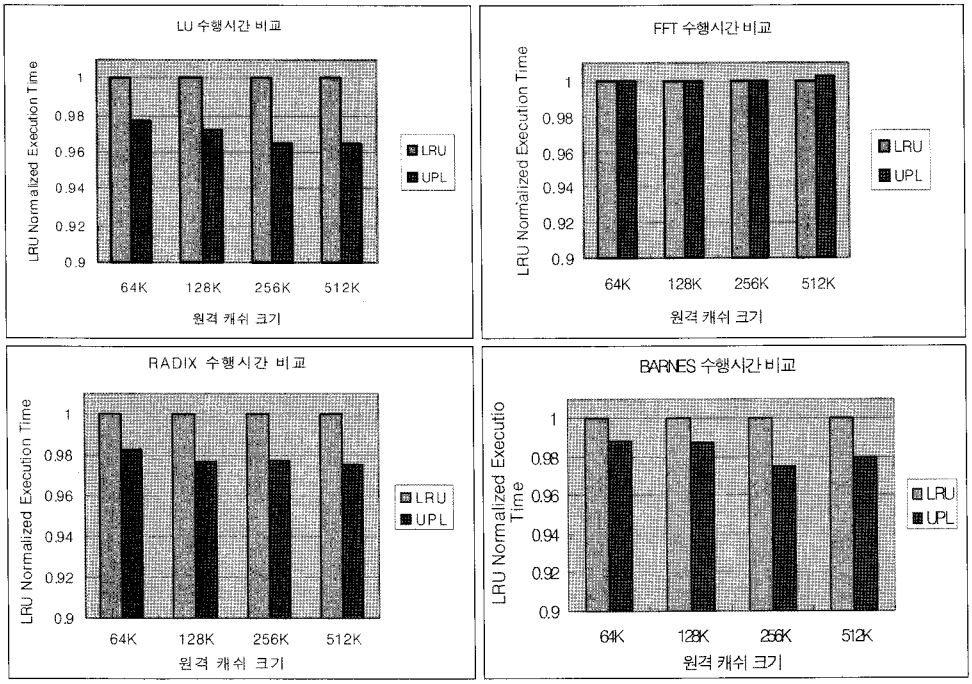


그림 19 원격 캐쉬 크기별 성능 향상 비교

다단계 캐쉬 내포성을 유지하여야 한다. 다단계 캐쉬 내포성은 각 캐쉬의 지역 LRU에 따르는 교체 정책으로 인해 프로세서로부터의 시간적 접근 순서에 따르는 지역성을 반영시키지 못하며, 내포성을 유지하기 위하여 상위 캐쉬의 LRU 정보와 다른 순서의 무효화 요청을 상위 캐쉬로 발생시킨다.

본 논문은 분산 공유메모리 다중프로세서 시스템에서 프로세서의 원격 메모리 접근이 프로세서 지역성을 나타낼 수 있음에 주목하여, 특정 프로세서의 원격 메모리 접근이 시간적 지역성을 나타낼 경우 이러한 프로세서 지역성에 따른 원격 캐시 라인 교체가 일어날 수 있도록 uPL 캐쉬 교체 방법을 제시하였다.

논문에서 제안한 캐시 교체 정책은 상위 단계 캐쉬로의 불필요한 무효화를 줄여, 상위 단계 캐시 적중률을 향상시켰으며, 원격 캐시의 적중률 또한 5~10% 개선시켜, 원격 메모리 접근 횟수를 감소시킬 수 있었으며, 이 결과 2~3.5%의 수행 시간의 향상을 나타냈다. 향후 연구 과제로 History 큐의 프로그램별 동적 크기 변화와 임계값을 동적으로 관리하는 알고리즘 개선 등이 남아 있다.

참 고 문 헌

[1] J. L. Hennessy and D.A Patterson, Computer Architecture : A Quantitative Approach, Third Edition, Morgan Kaufmann Publishers, Inc, 2003.

[2] Manuel E. Acacio, Jose Gonzalez, "A Two-Level Directory Architecture for Highly Scalable cc-NUMA Multiprocessors," IEEE Transactions on Parallel and Distributed System, Vol. 16, No. 1, January 2005.

[3] Zhang, Z. and J. Torrelas, "Reducing Remote Conflict Misses : NUMA with Remote Cacae versus COMA," In Proc. of the 3rd IEEE Symp. on High performance Computer Architecture (HPCA-3), pp. 272-281, Feb. 1997.

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

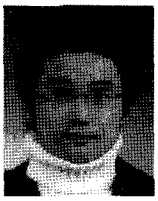
[5] B. Nelson, J. Archibald, and K. Flanagan, "Performance Analysis of Inclusion Effects in Multi-Level Multiprocessor Caches," Proceedings of the Third IEEE Symposium, pp. 513-516, Dec. 1991.

[6] J.-L. Bear and W.-H. Wang. "On the inclusion properties for multi-level cache hierarchies," In proceeding of 15th International Symp. on Computer Architecture, pp. 73-80, IEEE, 1988.

[7] 서효중, 장성태, 전주식, "내포성이 제거된 공유 캐쉬에 기반한 계층 버스 CC-NUMA 다중처리기", Journal of KISS A, vol. 25, no. 3, pp. 306-321, 1998.

[8] E. Cecchet, "Parallel pull-based LRU : a request distribution algorithm for clustered Web caches using a DSM for memory mapped networks," in Proceedings of the Cluster Computer and the Grid, IEEE/ACM International Symposium on,

15-18 May 2001.
[9] Wayne A. Wong and Jean-Loup Baer, "Modified LRU Policies for Improving Second-level Cache Behavior," In Proceedings of the 6th International Symposium on High-Performance Computer Architecture (HPCA), pp. 49-60, January 2000.
[10] J. Alghazo, A. Akaaboune, N. Botros, "SF-LRU Cache Replacement Algorithm," in Proceedings of Memory Technology, Design and Testing, 2004. Records of the 2004 International Workshop on, 9-10 Aug. 2004.
[11] D. Lee, J. Choi, H. Choe, Noh, S. Min, Y. Cho, "Implementation and Performance Evaluation of the LRFU Replacement Policy," In Proceedings of 23rd Euromicro Conference: New Frontiers of Information Technology-Short Contributions, September 01-04, 1997.
[12] Jeffrey D. Gee, Alan Jay Smith, "Analysis of Multiprocessor Memory Reference Behavior," in Proceedings of the IEEE International conference on, 10-12 Oct. 1994.
[13] P. Foglia, R. Giorgi and C.A. Prete, "Simulation study of memory performance of SMP multiprocessors running a TPC-W workload," in Proceedings of the IEE Proc.-Comput. Digit. Tech, Vol. 151, No. 2, March 2004.
[14] JACK E. Veenstra, Robert J. Fowler, "MINT : A front end for efficient simulation of shared-memory multiprocessors," In Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems(MASCOTS), pp. 201-207, 1994.
[15] S. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," In Proceedings of 22nd International Symposium on Computer Architecture, pp. 24-36, June 1995.



한 상 윤
2002년 2월 아주대학교 정보 및 컴퓨터공학과 공학사. 2004년 2월 서울대학교 대학원 전기 컴퓨터공학과 석사. 2004년 3월~현재 서울대학교 대학원 전기컴퓨터공학부 박사과정. 관심분야는 컴퓨터 구조, 병렬 처리 시스템, 저전력 내장형 시스템, 차세대 모바일 시스템

곽 종 욱
정보과학회논문지 : 시스템 및 이론
제 32 권 제 5 호 참조

장 성 태
1986년 2월 서울대학교 전자계산기공학과 공학사. 1988년 2월 서울대학교 대학원 컴퓨터공학과 석사. 1994년 2월 서울대학교 대학원 컴퓨터공학과 박사. 1994년 3월~현재 수원대학교 정보공학대학 컴퓨터학과 부교수. 관심분야는 다중 프로세서 시스템, 병렬 처리, 차세대 모바일 시스템, 저전력 임베디드 프로세서 설계

전 주 식
정보과학회논문지 : 시스템 및 이론
제 32 권 제 5 호 참조