프로세서 지역성에 기반한 원격 캐시 교체 정책

한상윤[O] 곽종욱 전주식
서울 대학교 전기 컴퓨터 공학부
{leaderhsy[O], leoniss}@panda.snu.ac.kr, csjhon@riact,snu.ac.kr

Remote Cache Replacement Policy based on Processor Locality

Sang Yoon Han[O] Jong Wook Kwak Chu Shik Jhon
Dept. of Electrical Engineering and Computer Science, Seoul National Univ.

요 약

본 논문에서는 원격 캐쉬를 추가시킨 분산 메모리 구조 다중 프로세서 시스템의 성능 향상을 위해 새로운 원격 캐쉬 교체 정책을 제안한다. 일반적으로 다중 계층 내포성(MLI)을 지키는 다중 계층 메모리 구조에서 LRU 교체 정책을 사용할 경우, 상위 계층 캐쉬의 LRU 정보와 하위 계층 캐쉬의 LRU 정보가 서로 상이함으로 인해 하위 계층 캐쉬에서의 교체가 상위 계층에서 사용 중인 캐쉬 라인의 교체를 발생시켜 전체 시스템의 성능을 저하시키는 원인이 된다. 이러한 LRU 캐쉬 교체 정책의 단점을 보완하고자 각 노드 당 프로세서들의 원격 메모리 접근 지역성을 이용한 원격 캐쉬 교체 정책의 사용으로 상위 캐쉬의 유용한 캐쉬 라인의 접근 실패율을 감소시킴으로써 다중 프로세서 시스템의 성능 향상을 꾀한다. 프로그램 기반 시뮬레이터를 통해 제안한 원격 캐쉬 교체 정책을 적용하였을 때, 기존의 LRU 교체 정책과 비교하여 무효화 수와 캐쉬 접근 실패가 평균 5%, 최대 10% 감소하였다.

1. 서 론

지금까지 연구되어 온 대규모 병렬 구조 다중 프로세서 시스템은 메모리의 결합 형태에 따라 분산 메모리 구조(Distributed Shared-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 고안된 것이 원격 캐쉬(RC : Remote Cache)이다[1].

한편, 여러 단계의 캐쉬를 계층적으로 구성한 시스템이 연구되었다. 이러한 다단계 캐쉬 구조에서, 어떤 프로세서로부터 메모리 접근 요청을 받은 원격 노드는 접근 경로 상에서 상위 레벨의 모든 캐쉬 내용을 확인함으로써 그 캐쉬 아래 레벨로의 메모리 접근 요구 전달이 이루어져야 하는지를 결정할 수 있다. 이러한 캐쉬 일관성 유지는 복잡하고 큰 오버헤드를 발생시키기 때문에 이를 해결하고자 다단계 내포성이 제시되었다. 이는 하위 레벨 캐쉬가 상위 레벨 캐쉬의 모든 내용을 포함하고 있어야 하는 성질로써, 다중 계층 캐쉬 구조에서 메모리 접근에 대한 프로토콜을 단순 명료하게 설계 및 구현을 가능하게 하고 메모리 접근 경로의 단순화로 접근 지연을 줄일 수 있게 하였다.

하지만, 다단계 캐쉬 구조에서 내포성 유지와 캐쉬 교체 정책의 결합은 캐쉬의 이용 목적인 시공간적 지역성을 최대한 지켜주지 못하는 원인으로 작용하기도 한다. 본 논문은 이러한 캐쉬 교체 정책과 다단계 내포성의 결합으로 인해 발생하는 비효율적인 캐쉬 교체를 줄이고자 프로세서 지역성을 이용한 새로운 캐쉬 교체 정책을 제시하기로 한다.

2. 관련 연구

이 절에서는 관련 연구로 다단계 캐쉬 구조와 다단계 캐쉬 내포성에 대해 서술하겠다. 프로세서와 메모리의 속도 차이가 급격히 증가함에 따라 다단계 캐쉬가 연구되었다. 다중 프로세서 시스템에서 크기가 큰 하위 레벨 캐쉬는 각 프로세서마다의 메모리 접근 수를 줄이고 메모리로부터의 데이터 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 다중 계층 캐쉬가 시스템의 정확성을 해치지 않으면서 일관성 문제를 정확히 해결하고 또한 하위 레벨 캐쉬가 메모리, I/O 또는 다른 프로세서의 트랙픽을 상위 레벨 캐쉬로의 접근을 줄여 프로세서의 캐쉬 이용율을 높이는 다중 계층 내포성(MLI)이 제시 되었다. 다중 계층 내포성(MLI)이란 캐쉬가 다중 계층 캐쉬 구조를 이루고 있을 경우, 임의의 하위 레벨 캐쉬에 상위 레벨 캐쉬의 모든 내용을 포함하고 있어야하는 성질을 말한다. 이를 유지함으로써 하위 레벨 캐쉬가 상위 레벨로의 트랜잭션 흐름을 필터링(filtering)하여 상위 레벨 캐쉬 효율성을 증가 시켜준다.

3. 프로세서 지역성을 이용한 원격 캐쉬 교체 정책

본 논문에서 제안하는 교체 정책을 RMAL 캐쉬 교체 정책(Cache Replacement Policy using Remote Memory Access Locality)이라 한다. RMAL 교체 정책은 다단계 내포성과 다단계 캐쉬 구조에서 발생 가능한 시스템 성능 저하 원인을 분석하여 이를 줄여 성능향상을 가져오기 위해 고안된 것이다. LRU 교체 정책은 다중 계층 내포성을 유지하는 다중 계층 캐쉬에서 가장 이상적이라고 가정한 캐쉬 교체 정책, 즉 모든 단계의 캐쉬가 왜곡되지 않은 미사용 캐쉬 라인 정보를 안다고 가정한 교체 정책의 성능에 미치지 못한다. 이는 곧 성능 향상의 여지가 남아있다는 것을 의미한다. 그림 1에서 보는 것과 같이 레벨 2 캐쉬와 원격 캐쉬에서 LRU 교체 정책을 사용할 경우 상이한 LRU 정보를 가지는 것을 볼 수 있다. 4-way 집합 연관성을 가정한 그림 1을 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 1에서 마지막 원격 메모리 접근 주소인 A는 레벨 2 캐쉬에서의 LRU 정책이었다면 여전히 교체되지 않고 접근 실패가 발생하지 않았을 것을 원격 캐쉬의 왜곡된 LRU 정보에 의해서 교체됨으로써 레벨 2 캐쉬의 캐쉬 적중률을 떨어뜨리는

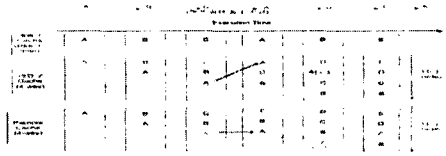

그림 1. LRU에 의한 캐쉬 교체 시나리오

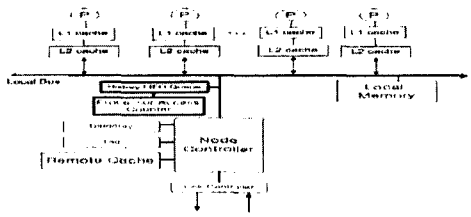

그림 2. RMAL 교체 정책을 위해 추가된 하드웨어 장치

원인이 된다. 이는 곧 다중 프로세서 시스템의 전체 시스템 성능 저하의 원인이 되기도 한다.

분산 메모리 구조 다중 프로세서 시스템은 하나의 노드 내에 여러 개의 프로세서가 있고, 각 노드마다 지역 메모리를 가지고 있다. 원격 캐쉬는 노드 내에 있는 모든 프로세서에 의해 공유되어 사용된다. RMAL 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(이하 프로세서 지역성), 이 프로세서에게 우선권(Priority)을 주어 LRU에 의해서 쫓겨나지 않게 하여 상위 레벨의 캐쉬 적중률을 높이기 위한 정책이다.

RMAL 캐쉬 교체 정책 방법 및 하드웨어의 구현 방안은 다음과 같다. 그림 2에서와 같이 한 노드 내의 존재하는 4개의 프로세서를 P_0, P_1, P_2, P_3라고 하자. 프로그램이 실행하는 도중 원격 캐쉬에서 충돌 실패가 발생하여 교체될 캐쉬 라인을 찾고 있다. 그리고 이 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 시점에서 가까운 과거동안 나머지 프로세서에 비해 원격 메모리에 대한 접근이 많았다는 것을 뜻한다. 이는 곧 그 시점에서 원격 캐쉬에 접근이 많은 프로세서 P_0는 원격 메모리에 대한 참조가 많고 이렇게 하여 사용된 메모리 영역은 시공간적 지역성이 있어 상위 레벨 캐쉬에서는 사용되어 LRU 정보가 갱신되었으나, 가장 하위 레벨에 있는 원격 캐쉬에는 그 후 프로세서의 메모리 접근 정보가 갱신되지 않아 LRU 라인으로 왜곡되어 교체 대상이 되는 것으로 생각할 수 있다. 이러한 교체는 노드 내 특정 프로세서에게는 유효한 라인임에도 불구하고 무효화가 되어 P_0 프로세서 상위 레벨 캐쉬의 적중실패율을 높이는 원인이 된다. 이처럼 RMAL 캐쉬 교체 정책은 왜곡된 LRU 교체 라인보다는 원격 메모리 참조 경향이 낮은 프로세서의 참조라인을 교체 대상으로 결정함으로써 프로세서 캐쉬에서의 전체 무효화 수와 모든 다중 계층 캐쉬의 적중 실패율을 감소시킴으로써 전체적인 시스템의 성능을 향상시킬 수 있다. 이와 같은 RMAL 캐쉬 교체 정책을 구현하기 위해서는 두 가지 정보가 필요하다. 첫째로 원격 메모리 전체 라인에 대한 프로세서 참조 정보가 필요하다. 이를 위해서 그림 3처럼 각 라인에 한 노드 내의 프로세서 수만큼의 공유 비트를 두어 각 라인에 대한 프로세서의 참조 여부를 나타낸다. 이는 프로세서가 읽기 또는 쓰기 요청을 지역 공유 버스에 내보낼 때 어느 프로세서가 요청하는지를 나타내는 프로세서 아이디(Processor ID)를 함께 내보내는데, 이를 보고 각 라인의 프로세서 공유 비트에 프로세서 아이디에 맞도록 1을 설정하면 된다.

Tag	Processor Sharing bit	LRU	Count
A	P0 (1000)	1	
B	P1, P3(0101)	2	
C	P2(0010)	3	
D	P1(0100) -> 0000	4 LRU	

그림 3. 프로세서 공유 비트를 가진 원격 캐쉬 라인 구조

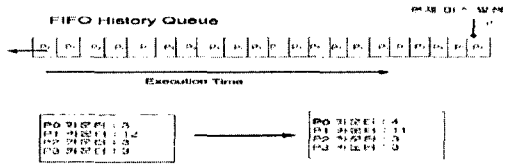

그림 4. 동적 FIFO 큐와 프로세서 접근 카운터

다음으로 RMAL 캐쉬 교체 정책을 위해 캐쉬 교체가 발생한 시점에서 필요한 정보는 가까운 과거에서 현재 교체 시점까지 노드 내 프로세서들의 원격 메모리 참조 패턴(Pattern History)이다. 이 정보를 얻기 위해서는 동적인 패턴 분석이 필요한데 그림 4와 같다. 그림 4에서처럼 요청 기록 FIFO(Request History First-In First-Out) 큐를 두어 지역 버스에서 발생하는 원격 메모리 참조 프로세서들의 프로세서 아이디를 저장하면 된다. FIFO 구조이기 때문에 오래된 프로세서 아이디는 큐에서 없어지고 새로이 지역 버스에 발생하는 원격 메모리 참조 프로세서 정보만 큐에 유지된다. 이러한 정보는 동적으로 바뀐다는 것을 알 수 있다. 그리고 프로세서 접근 카운터는 들어오는 프로세서의 카운터를 1 증가시키고, 큐에서 나가는 프로세서는 1 감소시켜 캐쉬 미스 발생시 이 카운터 정보를 이용한다.이제는 앞서 언급한 두 가지 정보를 가지고 RMAL 캐쉬 교체 정책이 실제로 어떻게 동작하는지에 대해 설명한다. 원격 캐쉬에서 적중 실패가 발생하여 교체 대상을 선택해야 한다고 하자. 그림 3에 의하면 하나의 4-way 집합 연관성에서 일반적인 LRU에 의해 마지막 라인인 태그 D가 교체 대상이 된다. 하지만 RMAL 캐쉬 교체 정책의 경우 그림 4에서 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 4를 보면 P_1의 접근 횟수가 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. 그림 3에서 LRU에 의해 선택된 교체 라인의 프로세서 공유 비트를 보면 P_1이 참조하고 있는 것을 알 수 있고, 현재 P_1의 원격 캐쉬 접근 수가 임계값을 넘었다면, 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 다음번 교체 시에는 또 그런 특혜를 받는 것을 막기 위해 이 라인의 프로세서 공유 비트의 P_1을 나타내는 디렉토리 비트를 0으로 설정한다. 이는 적절한 길이의 시간적 지역성을 유지해주면서 LRU를 따르는 원리이다. 다음 LRU 라인인 태그 C는 P_2가 참조한 라인이고, 현재 P_2의 원격 캐쉬 접근 수가 3이므로 이를 교체 대상으로 선택한다. 한편, 모든 집합 내의 라인에 대해 RMAL 정책을 적용하면 가장 최근(MRU)에 접근된 라인이 교

체 되어 성능 저하의 원인이 될 수 있다. 따라서 MRU인 태그 A는 제외한다. 보통 MRU 라인은 시간적 지역성이 크기 때문에 제외하는 것이 LRU 교체 정책의 장점을 살려주기 때문이다. 만약 집합 내의 모든 라인이 P_I에 의해 참조되는 것이라면 기존 LRU 교체 정책에 의해 교체 대상이 선택된다. 이처럼 RMAL 캐쉬 교체 정책도 프로세서의 원격 캐쉬 접근 지역성을 바탕으로 LRU 교체 정책을 효율적으로 이용함으로써 상위 레벨 캐쉬의 불필요한 무효화를 줄여 적중률을 높이고, 이를 바탕으로 전체 시스템의 성능 향상을 가져올 수 있다.

4. 성능 평가

4.1 모의 실험 환경

본 논문에서 제안한 원격 캐쉬 교체 정책을 평가하기 위해서 MINT[3] simulator와 SPLASH-2 벤치마크 프로그램[4]을 이용하여 수행하였다. 시스템 인자 값은 표 1과 같다.

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 데이터 전송 시간	18 Bus Clock

4.2 모의 실험 결과

4.2.1 프로세서 캐쉬의 적중률 비교

그림 5는 프로세서 캐쉬의 적중률을 LRU에 정규화하여 비교한 그림이다. LU와 Radix, Barnes는 프로세서 캐쉬의 무효화 수가 줄어 적중률이 높아졌다. 하지만 FFT는 거의 동일한 결과를 보이고 있다. 보통 LRU 정책을 사용할 경우 프로세서 캐쉬의 적중률은 90%를 상회하는데, 이러한 LRU 정책의 적중률에 정규화하여 0.2%에서 0.4%까지 상승하였다.

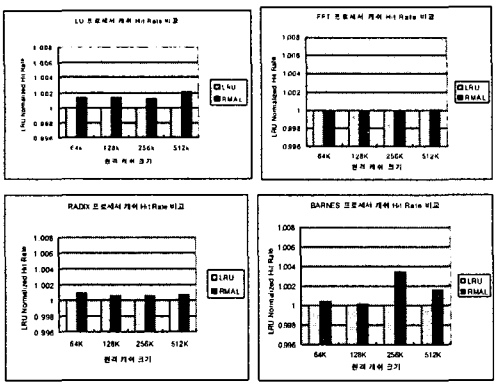

그림 5. 프로세서 캐쉬의 적중률 비교

4.2.2 원격 캐쉬 적중률 비교

그림 6은 원격 캐쉬의 적중률 비교로 전반적인 원격 캐쉬의 적중률이 RMAL 정책의 사용으로 상승한 것을 볼 수 있다. LU, Barnes, Radix는 프로세서 캐쉬의 무효화 수 감소와 프로세서 캐쉬의 적중률 상승로 인해 원격 캐쉬로의 접근 수가 줄어들었고, 또한 이는 원격 캐쉬의 적중률 상승을 가져왔다. 원격 캐쉬 적중률 증가는 평균 5%에서 최대 10%까지 높은 향상을 보였다.

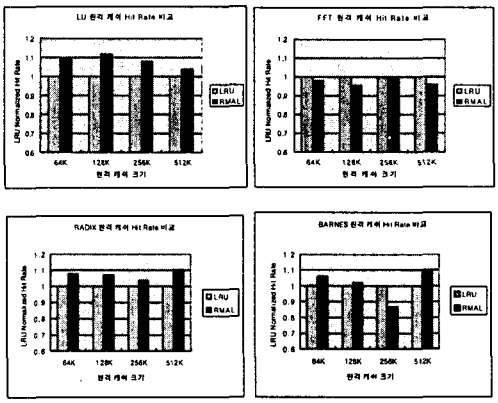

그림 6. 원격 캐쉬 적중률 비교

5. 결 론

본 논문에서는 한 노드 내의 여러 프로세서를 가진 분산 메모리 구조 다중 프로세서 시스템에서 프로세서들 마다 원격 메모리 접근 패턴을 달리하는 특성을 이용하였다. 하나의 프로세서가 프로그램 실행 전체에서 원격 메모리에 접근 할 수도 있고, 때로는 노드 내 모든 프로세서들이 균등하게 원격 메모리 영역을 접근할 수도 있다. 또한 특정 시점에 한 프로세서의 원격 메모리 접근 독점이 일어날 수 있다. 이런 프로세서 지역성을 이용한 RMAL 캐쉬 교체 정책은 유효한 프로세서 캐쉬의 무효화를 줄이고 적중률을 높여, 원격 메모리 접근 지연을 줄임으로써 다중 프로세서 시스템의 전체 성능 향상을 가져 올 수 있었다. 실험 결과 원격 캐쉬 적중률의 경우 LRU에 비교하여 평균 5%에서 최대 10%까지 증가하였다,

참고 문헌

[1] John L. Hennessy, David A. Patterson. "Computer Architecture: A Quantitative Approach" 3th Edition, 2003.

[2] 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.

[3] 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.

[4] 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.