

홈 노드 갱신 트래픽을 줄이는 디렉토리 캐쉬 일관성 기법

박상목⁰ 이윤석
한국의외국어대학교 전자정보공학부
(psm1942, rheeys}@san.hufs.ac.kr

A Directory Cache Coherence Scheme for Reducing Write-back Traffic to Home Nodes

Sangmook Park⁰ Yunseok Rhee

School of Electronics and Information Eng., Hankuk Univ. of Foreign Studies

요 약

기존의 디렉토리 프로토콜은 독점 상태 데이터의 읽기 과정에서 홈 노드 갱신을 수행한다. 한편 이주 데이터는 한 프로세서에 의해 읽힌 뒤 곧이어 수정되므로 그와 같은 홈 노드 갱신이 전혀 쓸모없게 된다. 본 논문에서는 기존의 디렉토리 캐쉬 일관성 기법을 수정하여 이주 데이터에 대해 발생하는 불필요한 홈 노드 갱신을 줄이는 개선된 기법을 제안한다. 제안된 기법은 이주 데이터의 빈도가 높은 병렬 프로그램에서 캐쉬 일관성 트래픽을 크게 감소시켰고 이는 자연히 네트워크 지연 시간의 단축을 가져왔다.

1. 서론

병렬 프로그램에서 한 프로세서에 의해 읽기-계산-쓰기가 실행되고, 뒤이어 다른 프로세서에 의해 동일한 실행이 이뤄지는 데이터를 '이주 데이터(migratory data)'라고 일컬으며, 이와 같은 데이터의 이주성(migratory property)은 세마포어, 모니터 등의 동기화 객체를 사용하는 병렬 프로그램에서 빈번하게 나타나는 데이터 공유 형태의 하나이다 [1]. 따라서 이주 데이터를 효과적으로 지원하여 병렬 시스템의 성능을 개선하려는 여러 선행 연구들이 수행된 바 있다 [2,3].

공유 메모리 다중처리 시스템에서 각 프로세서 노드는 원격 노드에 있는 데이터의 효과적 접근을 위해 캐쉬 메모리를 사용한다. 따라서, 같은 데이터의 사본이 여러 캐쉬에 만들어질 수 있고, 이들 사이의 일관성을 유지하기 위한 '캐쉬 일관성 유지 기법'(또는 프로토콜)이 요구된다. 버스 이외의 직접 연결 상호연결망에 기반한 시스템, 소프트웨어 DSM 등에서는 디렉토리 기반 캐쉬 일관성 프로토콜(directory-based cache coherence protocol, 이하 디렉토리 프로토콜)이 사용된다 [5].

기존 디렉토리 프로토콜에서 수정된(modified 또는 dirty) 캐쉬 블록은 다른 프로세서에 의해 읽혀지는 시점 즉, 공유되는 시점에서 해당 블록의 홈 노드로 전달되어 메모리 블록을 갱신(write-back)한다 [4]. 그러나, 이주 데이터를 읽어간 프로세서는 곧 그 내용을 다시 변경함으로써 직전에 발생한 홈 노드 갱신을 쓸모없게 만든다. 제안하는 프로토콜에서는 이주 데이터에 대해 이와 같은 불필요한 홈 노드 갱신을 생략함으로써 캐쉬 일관성 메시지와 네트워크 트래픽을 줄이고자 한다.

이러한 캐쉬 일관성 메시지의 감소는 메시지 생성 및 네트워크 진입, 전달, 처리 시간의 단축을 가져올 뿐만 아니라, 제한된 대역폭에서 혼잡도를 완화시켜 네트워크 지연시간을 줄이는 효과를 가져올 수 있다. 또한 홈 노드의 갱신은 해당 블록에 대한 쓰기 순서화(write serialization)를 위해 다른 프로세서의 메모리 접근을 지연시키므로, 불필요한 갱신 횟수의 감소는 메모리 접근 지연 시간을 단축시킬 것으로 기대된다.

본 논문에서는 2절에서 디렉토리 캐쉬 일관성 프로토콜을 소개하고, 3절에서는 이주 데이터의 성질을 활용하여 불필요한 홈 갱신을 줄이는 디렉토리 프로토콜을 제안한다. 4절에서는 제안한 프로토콜에 대한 간단한 실험

결과를 제시하며, 마지막으로 5절에서는 결론과 향후 연구에 대해 기술한다.

2. 디렉토리 캐쉬 일관성 프로토콜

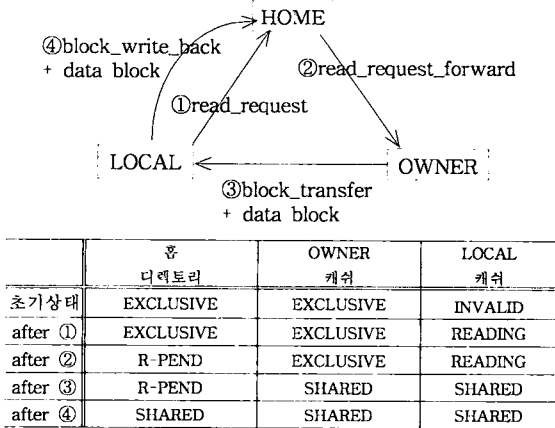
디렉토리 프로토콜은 노드간에 직접 연결망을 사용하는 대규모 다중처리 시스템에서 일반적으로 채택하는 방법이며, 다양한 변형 프로토콜들이 제안되었지만 본 절에서는 홈 노드가 존재하고 전사(full-map) 디렉토리를 사용하는 쓰기 무효화(write invalidate) 프로토콜을 중심으로 주요 구성 요소와 동작 과정을 소개한다 [8].

디렉토리 프로토콜에서는 각 메모리 블록마다 해당 블록의 공유 상태, 공유 노드를 기록하는 저장 공간을 두며 이를 '디렉토리(directory)'라 한다. 일반적으로 해당 메모리 블록을 제공하는 노드가 디렉토리 역시 관리하며, 이 노드를 '홈(home) 노드'라고 부른다. 홈 노드로부터 메모리 블록을 전달받아 이를 자신의 캐쉬에서 공유하는 노드를 '공유 노드(shared node)'라 하고, 홈 노드는 모든 공유 노드들의 위치와 공유 여부를 디렉토리에 표시한다.

읽기의 노드가 메모리 쓰기를 요구하면, 홈 노드는 모든 캐쉬 사본들을 무효화(invalidate)시키고, 쓰기를 요구한 노드에만 캐쉬 사본을 허용한다. 또한 이후의 일련의 쓰기 동작을 해당 노드 내부적으로 수행할 수 있도록 '소유권(ownership)'을 부여하며, 소유권을 부여받은 노드를 '소유자(owner) 노드'라고 한다. 소유자 노드는 해당 캐쉬 블록의 최근 내용을 캐쉬에 유지하게 되므로, 이후에 발생하는 다른 노드의 읽기 요구를 서비스하고, 메모리 블록에 최종 결과를 갱신하는 책임을 갖는다.

캐쉬 상태는 기본적으로 MESI 프로토콜[6]과 유사하지만, 디렉토리 프로토콜에서는 읽기만이 가능한 '공유(SHARED)' 상태와, 소유권을 확보하고 내용 변경이 가능한 '독점(EXCLUSIVE)' 상태로 구분된다. 또한 디렉토리 프로토콜에서는 메모리 블록의 접근 요구가 네트워크를 통해 비동기적으로 서비스되므로, 서비스 진행 중임을 나타내는 '읽기 중(READING)', '무효화 중(INVALIDATING)' 등의 상태가 필요하다.

디렉토리 상태는 캐쉬 사본의 존재, 소유권 이전 여부 등에 따라 '사본없음(ABSENT)', '공유(SHARED)', '독점(EXCLUSIVE)' 상태로 구분되며, 캐쉬 상태와 유사하게 원본과 동일한 단일 사본만이 존재하는 상태는 공유 상태로 표시한다. 또한 다른 노드에 소유권을 부여하고 블



[그림 1] 독점 상태를 갖는 메모리 블록의 읽기 접근 과정

록 내용이 변경된 경우는 독점 상태로 구분한다.

그림 1은 LOCAL 노드에서 캐쉬 읽기 실패가 발생한 경우의 처리 과정을 단계적으로 보여주며, 최종적으로 LOCAL 노드는 OWNER 노드로부터 전달받은 캐쉬 블록을 다시 HOME 노드로 보내 메모리 블록을 갱신하고, 디렉토리의 상태를 공유 상태로 변경한다. 이를 통해 이후의 해당 블록에 대한 메모리 읽기 요청은 HOME 노드가 서비스한다.

3. 이주성을 활용한 디렉토리 프로토콜

3.1 프로토콜의 구성

앞서 살펴 본 바와 같이 기존 프로토콜에서 독점 상태를 갖는 데이터를 읽는 과정에 홈 노드로의 갱신이 수반된다. 그러나, 이주 데이터를 읽어 간 프로세서는 곧 이를 변경하고 독점 소유권을 확보하므로 홈 노드 갱신이 필요없어 지므로 이 과정을 생략할 수 있다. 그러나, 독점 블록의 읽기 접근 과정에서 홈 노드로의 갱신(그림 1의 ④)을 없애면 두 가지 문제가 제기된다.

첫째는 기존 프로토콜에서는 홈 노드의 갱신과 함께 디렉토리 상태와 캐쉬 사본들의 상태가 공유 상태로 정의된다. 그러나, 이를 생략하면 홈 노드의 메모리 블록은 수정되기 이전 값(stale value)을 여전히 갖고, 동시에 이 블록이 여러 캐쉬에서 공유되는 상황이 발생한다. 기존 프로토콜이 정의하는 독점, 공유 상태 중 어느 것도 이 상황을 표현할 수 없다(표 1 참조). 이를 해결하기 위해, 제안 프로토콜에서는 디렉토리 상태에 각각 '경과(TRANSIENT, 이하 T로 표현)'라는 새로운 상태를 다음과 같이 추가 정의한다.

- 디렉토리에서의 경과 상태(T)는 "해당 메모리 블록이 다른 프로세서의 캐쉬에서 수정되었으며, 이 캐쉬 블록의 사본이 존재한다"는 것을 뜻한다.
- 캐쉬에서의 경과 상태(T) 역시 디렉토리에서와 동일한 상황을 표현하지만, 이에 덧붙여 "해당 캐쉬가 이 블록의 소유자이며, 홈 노드에 갱신할 책임을 갖는다"는 것을 뜻한다. 주의할 점은 이 상황에서 해당 사본을 제외한 나머지 캐쉬 사본들은(경과 상태가 아닌) 공유 상태를 갖는다는 것이다.

둘째는 경과 상태(T)를 갖는 메모리 블록에 대한 읽기 접근을 서비스할 소유자 노드의 관리 방법을 개선해야

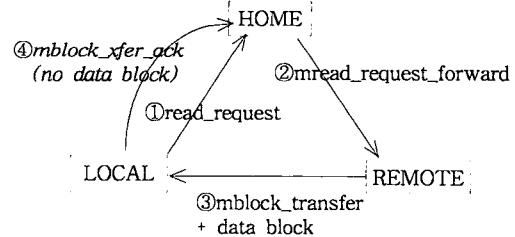
한다. 기존 프로토콜을 단순히 확장하면, 처음에 해당 블록을 독점 상태로 만든 소유자 노드를 경과 상태(T)에서도 여전히 소유자로 유지할 수 있다. 그리고, 이 블록에 대해 읽기 접근을 시도하는 다른 프로세서들의 캐쉬에는 공유 상태의 사본들이 만들어진다.

그러나, 이주 데이터는 새로 이주한 노드에서 활발히 사용되고 해당 노드의 캐쉬에 더 오래 머무를 것으로 예측된다. 즉, 이주 전 노드에 있던 캐쉬 사본은 곧 캐쉬에서 다른 블록으로 교체될 가능성이 높고, 이에 따른 홈 노드 갱신이 발생한다. 따라서, 경과 상태(T)를 갖는 블록에 대해서는 단순히 소유자를 고정시키는 방법보다는 최근 접근 노드를 소유자로 변경하는 것이 합리적이다.

3.2 개선된 메모리 읽기 과정

제안하는 프로토콜은 기본적으로 읽기, 무효화, 쓰기 등의 모든 메모리 접근에 대한 구현 방법에서 기존 프로토콜과 약간씩 다르지만, 본 논문에서는 독점 상태를 갖는 메모리 블록에 대한 읽기 접근 과정만을 기술한다.

그림 2에 보이는 것처럼, 이주 데이터 블록으로 판단된 블록이 독점 상태에 있을 때 읽기 접근 요청이 오면, 홈 노드는 소유자 노드에게 '이주 데이터 전송을 요청'한다(after ②). 이를 전달받은 소유자 노드는 해당 블록과 함께 이주 데이터임을 알리는 메시지를 LOCAL 노드에게 전송하고, 자신의 캐쉬 사본은 공유 상태로 변경한



	홈 디렉토리	REMOTE 캐쉬	LOCAL 캐쉬	소유자
초기상태	EXCLUSIVE	EXCLUSIVE	INVALID	REMOTE
after ①	EXCLUSIVE	EXCLUSIVE	READING	
after ②	R-PEND	EXCLUSIVE	READING	
after ③	R-PEND	SHARED	T	
after ④	T	SHARED	T	LOCAL

[그림 2] 제안 프로토콜에서 독점 상태를 갖는 메모리 블록의 읽기 접근 과정

다(after ③). 이를 전달받은 LOCAL 노드는 자신이 해당 블록의 소유자임을 기록하고 홈 노드에게 전달 확인 메시지만을 전송한다. 홈 노드는 이 메시지를 받은 후에 해당 블록의 소유자를 LOCAL로 수정한다(after ④). 이 방법을 통해 홈 노드 갱신이 없이 수정된 데이터를 공유할 수 있다.

4. 성능 평가

4.1 실험 방법

제안하는 프로토콜의 성능을 측정하기 위해 표 1의 구성을 갖는 모의 다중처리 시스템을 Augmint[6]를 사용하여 구현하였다. Augmint는 x86 프로세서 명령어를 지원 하는 실행 주도(execution-driven) 모의 실험 패키지로서, 코드 첨가(augmentation) 방법과 명령어 해석 방법을

보완적으로 사용한다.

모의 실험기는 다중처리기의 각 프로세서에서 발생시키는 메모리 접근 명령을 가로채어 상호연결망, 메모리, 캐쉬 등의 시스템 자원 사용에 따른 지연 시간을 측정하도록 구현되었다. 실험 대상인 메쉬 상호연결망은 워홀스워칭 방식을 채택하고 매 클럭 사이클마다 동작 과정을 상세히 구현함으로써 네트워크 용량과 통신량에 따른 지연 시간의 비교적 정확한 측정이 가능하다. 모의 실험기에서 채택한 기본 캐쉬 일관성 프로토콜은 전사 디렉토리를 사용하며, 이주 데이터의 빈도와 영향을 자세히 관찰하기 위해 제안 프로토콜에서 이주 데이터를 따로 구분하지 않았다.

모의 실험기를 통해 성능을 살펴 볼 병렬 프로그램으로는 SPLASH-2 벤치마크[7]에서 FFT, Ocean, LU를 선정하였다. FFT는 기수 \sqrt{n} 6단계 푸리에 변환 알고리즘의 1차원 버전으로 6개의 배리어(barrier)로 이뤄지며, 각 구간에서 두 노드 간에 데이터 수수가 이뤄져 이주 데이터의 특성이 잘 나타나는 프로그램이다. LU는 행렬을 두 개의 삼각 행렬로 분해하는 프로그램으로, 캐쉬 적중률을 높이기 위해 전체 행렬을 캐쉬 크기에 맞도록 부분행렬로 나누어 각 프로세서에 할당한다. 할당된 부분 행렬의 계산 결과는 다시 나머지 프로세서들에게 전달되는 과정에서 프로세서간 데이터 공유가 발생한다. Ocean은 해류에 따른 대규모 해양 운동을 모의 실험하는 프로그램으로, 운동 요소를 정방형 그리드들로 분할하여 이들을 각 프로세서에 할당하고, 멀티그리드 방식적 해법을 통해 계산된 결과를 각 프로세서간에 교환함으로써 데이터 공유가 일어난다.

4.2 실험 결과

표 2는 기존 프로토콜의 성능을 기준으로 제안 프로토콜의 성능을 대비한 결과를 보인다. 갱신 트래픽 감소율은 기존 프로토콜에서 발생한 홈 노드로의 갱신 트래픽(그림 1의 ④)의 양, 즉 (갱신 횟수)×(기본 메시지 크기 + 캐쉬 블록 크기)와, 제안 프로토콜에서 발생한 홈 노드로의 응답(acknowledge) 트래픽(그림 2의 ④)의 양, 즉 (응답 횟수)×(기본 메시지 크기)의 차이로부터 구해진다. 따라서 이 결과는 캐쉬 블록과 기본 메시지 크기와도 밀접한 관계가 있다. 최근 시스템에서 캐쉬 블록의 크기가 점차 커가는 추세이므로 트래픽 감소율은 더욱 클 것으로 기대된다.

FFT와 Ocean은 대부분의 공유 데이터가 이주성을 띄어, 홈 노드 갱신 트래픽이 거의 완전히 감소하였다. 그러나, LU는 동일 계산 결과가 다수 프로세서에 의해 읽혀지는 까닭에 소유자 노드에게 부가적인 메시지 전달이 증가했고, 이는 오히려 트래픽의 감소율을 낮추는 결과를 가져왔다.

메시지 증가율은 제안 프로토콜에서 증가한 메시지 수를 나타낸다. 메시지 증가의 원인은 홈 노드에서 서비스되지 못한 읽기 요구가 소유자 노드에게 재전달됨에 의해 발생한다. 이주 데이터가 대부분인 FFT와 Ocean에 비해, LU는 매우 심각하게 메시지가 증가했다. 특히 이 결과는 전체 메모리 접근 시간에 직접적으로 영향을 끼친다. 따라서, 제안 프로토콜은 반드시 이주 데이터의 탐지를 통해 선별적으로 적용해야 함을 알 수 있다.

[표 1] 모의 실험기의 구성 요소

processor clock	600 MHz	L1 cache	16 KB
network clock	150 MHz	L2 cache	1 MB
network width	16 Bytes	L3 cache	32 MB
cache block size	128 Bytes	interconnection	8x8 mesh
set associativity	4-way	#nodes	64

[표 2] 실험 결과

프로그램	갱신 트래픽 감소율 (%)	메시지 증가율 (%)	네트워크 지연시간 감소율 (%)
FFT	94.2	0.0	8.3
LU	77.9	31.5	6.2
Ocean	93.5	2.6	11.1

네트워크 지연시간 감소율은 네트워크 트래픽 양에 비해서는 극적인 결과를 보이지 않는다. 특히 이 결과는 시스템이 제공하는 네트워크 대역폭과 트래픽 양에 의해 직접적으로 영향을 받는다. 열악한 네트워크에서는 제안 프로토콜이 네트워크 혼잡도를 더욱 크게 낮출 것으로 예상된다. 그러나 최근 네트워크 대역폭이 급속하게 향상되는 상황에서 이와 같은 트래픽의 감소가 큰 의미를 갖는 사용 환경이나 어플리케이션에 대한 연구가 필요하다.

5. 결론

본 논문에서는 기존의 디렉토리 프로토콜이 독점 상태 데이터에 대한 읽기 과정에서 불필요한 홈 노드 갱신을 일으키는 상황을 관찰하고 이를 효과적으로 줄일 수 있는 개선된 프로토콜을 제안하였다. 특히 이주 데이터에 대해서 제안 프로토콜이 캐쉬 일관성 트래픽의 양을 크게 줄임으로 그 성능을 입증하였다. 이러한 트래픽의 감소는 네트워크 혼잡도를 낮춰 자연스럽게 네트워크 지연 시간과 실행 시간의 단축으로 연결된다. 따라서 대역폭이 충분치 못한 환경이나 다양한 병렬 프로그램의 동시 수행으로 트래픽이 폭주하는 환경에 제안 프로토콜이 기여할 것으로 기대된다.

제안 프로토콜은 디렉토리과 캐쉬에 새로운 상태를 추가해야 하지만, 적은 비용으로 표현 공간과 로직을 구현할 수 있다. 그러나, 실험 결과에서 본 바와 같이 이주 데이터가 아닌, 즉 다수 프로세서에 의해 동시 읽기 접근이 요구되는 경우에는 제안 프로토콜이 오히려 큰 부담을 더한다. 따라서 프로그램 수행 중에 이를 정확히 탐지할 수 있는 안정적인 알고리즘의 적용이 반드시 필요하다. 향후 연구에서 이를 함께 구현하고 성능을 분석하고자 한다.

참고문헌

- [1] A. Cox and R. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data", Proc. of the 20th ISCA, pp. 98-108, 1993.
- [2] Per Stenstrom, Mats Brorsson, and Lars Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing", Proc. of the 20th ISCA, pp. 109-118, May 1993.
- [3] S. Kaxiras and J. R. Goodman, "Improving CC NUMA Performance Using Instruction-based Prediction," Proc. of the 5th International Symposium on High Performance Computer Architecture, pp. 161-170, Jan. 1999.
- [4] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based Cache Coherence in Large-scale Multiprocessors", IEEE Computers, pp. 49-58, June 1990.
- [5] Y. Rhee and J. Lee, "Broadcast directory: A scalable cache coherent architecture for mesh-connected multiprocessors", Journal of Systems Architecture 46(10), pp. 903-918, 2000.
- [6] A-T. Nguyen, M. Michael, A. Sharma, J. Torrellas, "The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures", Proceedings of 1996 International Conference on Computer Design, October 1996.
- [7] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", Proc. of the 22nd ISCA, pp. 24-36, June 1995.