적응적 메모리갱신 기법을 이용하여 읽기 접근시간을 줄이는 캐쉬 일관성 유지 기법

오 승 택*○ 이 윤 석** 이 준 원*
* 한국과학기술원 전자전산학과 전산학전공
** 한국외국어대학교 전자제어공학부 제어계측공학과
{stoh, joon}@camars.kaist.ac.kr, rheeys@control.hufs.ac.kr

A Cache Coherence Scheme for Reducing Read Access Latency with Adaptive Memory Update

Seung-Taek Oh* Yunseok Rhee** Joonwon Lee*
* Div. of Computer Science, Dept. of EECS., KAIST
** Div. of Electronics and Control Engineering, Hankuk University of Foreign Studies

요 약

대규모 분산 공유메모리 다중처리기는 공유메모리 접근 지연시간이 크다는 약점을 지니고 있다. 이러한 다중처리기에서 모든 메모리 요청이 홈노드를 통해 이루어지는 디렉토리 기반의 캐쉬 일관성 유지 기법의 사용은 메모리 접근 지연시간을 더욱 크게하는 요인으로 작용한다. 뿐만 아니라 메모리 접근 지연시간은 시스템의 규모가 커질수록 전체 성능에 중요한 요소로 작용하므로, 대규모 시스템에서 이를 줄이기 위해서 많은 연구들이 있었다. 본 논문에서는 메모리 읽기 지연시간을 줄이는 새로운 캐쉬 일관성 유지 기법을 제안한다. 제안된 기법은 적응적 메모리갱신을 이용하여 구현되었다. 적응적 메모리갱신은 홈노드의 메모리를 미리 갱신함으로써 읽기 접근 지연시간을 줄이는 방법이다. 이를 위해서 홈노드는 메모리 접근 유형을 분석해야한다. 대부분의 공유메모리 접근은 일정한 유형을 지니므로 이를 토대로한 홈노드의 갱신은 높은 적중률을 보인다. 제안된 프로토콜의 성능을 측정하기 위하여 모의실험을 하였다. 모의실험 결과는 제안된 프로토콜에서 읽기 지연시간과 실행시간이 감소하는 것을 나타낸다.

1. 서론

캐쉬 메모리는 프로세서의 수행속도에 비해 현저히 느린 메모리 접근 시간을 줄이기 위해 고안된 기술이다[1]. 캐쉬 메모리를 사용하는 시스템에서는 캐쉬 실패(cache miss)에 의한 메모리 접근 시간의 지연이 전체 시스템의 성능을 저하시키는 주된 요인 중 하나로 나타나고 있다. 특히 다단계 상호연결망에 의해 구성된 대부분의 다중처리기 시스템에서는 메모리 시스템이 다수의 처리기 사이에 분산되어 구현된다. 이러한 시스템에서는 원격 노드(remote node)에 위치한 메모리의 접근을 위해 긴 네트워크 지연시간까지 감수해야 하므로 캐쉬의 역할은 더욱 중요하다.

대표적인 다중처리기 시스템으로 CC-NUMA(Cache-Coherent Non-Uniform Memory Access Machines)를 들 수 있다[2]. CC-NUMA에서는 모든 메모리 블록들은 그 메모리 블록을 관리하는 노드가 정해져 있으며 이러한 노드를 홈노드(home node)라고 부른다. 홈노드가 하는 일은 크게 두 가지인데, 하나는 다른 노드에 대해서 메모리 블록을 제공하는 일이고, 다른 하나는 여러 캐쉬상에 존재하는 메모리 블록의 복사본들에 대한 일관성을 책임지는 일이다. 따라서 CC-NUMA의 메모리 시스템은 비교적 구현이 쉽고 확장성이 우수하여 많은 다중처리기에서 채택하고 있다. 그러나 CC-NUMA에서는 캐쉬의 일관성 유지가 홈노드를 통해서만 관리되므로 메모리 블록에 대한 요구가 반드시 홈노드를 거쳐야 한다는 단점이 있다. 이로 인해서 캐쉬의 읽기 실패에 따른 메모리 접근 지연시간(이하 지연시간)은 매우 길어지며, 이 문제는 대규모 다중처리기에서 더욱 심각하다.

본 논문에서는 디렉토리 프로토콜 기반의 CC-NUMA에서 캐쉬 메모리의 읽기 실패가 일어났을 경우 지연시간을 줄이기 위한 프로토콜(protocol)을 제안한다. 본 프로토콜에서는 지연시간을 줄이기 위해서 적응적 메모리갱신(adaptive memory update) 기법을 사용한다. 적응적 메모리갱신 기법이란 메모리 읽기 쓰기 유형을 관찰하고 있다가 쓰기 이어서 바로 다른 노드의 읽기가 예상 될 경우 메모리를 쓰는 노드가 오너쉽(ownership)을 가져오지 않고 메모리를 갱신 시키는 방법이다. 이 경우는 이어서 오는 읽기 요청에 대하여 홈노드가 바로 데이터를 보낼 줄 수 있어서 지연시간을 줄일 수 있다. 본 논문에서는 모의 실험을 통해서 제안된 프로토콜이 기존의 프로토콜에 비하여 성능이 향상됨을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 컴파일러에 의한 메모리 갱신 기법에 대하여 살펴본 후, 3장에서는 적응적 메모리갱신 기법을 이용하여 읽기 실패의 지연시간을 줄이는 새로운 프로토콜을 제안한다. 4장에서는 제안된 프로토콜의 성능에 관한 모의실험 결과를 살펴본 후 마지막으로 5장에서 결론과 향후 연구 과제를 밝힌다.

2. 컴파일러에 의한 메모리 갱신

컴파일러에 의한 메모리 갱신(compiler-driven memory update)은 컴파일러가 프로그램을 분석하여 적당한 시점에서 홈노드의 메모리 블록을 갱신(update)시켜 주는 방법이다[3]. 컴파일러는 하나의 노드가 메모리 블록에 대하여 쓰기를 하는 유형을 분석하여 마지막 쓰기에 대해서는 update라는 새로운 명령어로 대치하여 컴파일한다. 프로세서는 프로그램 수행 시 update명령어를 발견하면 자신의 캐쉬 블록에서 쓰기를 수행하는 대신 홈노드의 메모리 블록을 갱신시키는 작업을 수행한다. 따라서 이후에 나타나는 읽기에 대해서는 홈노드가 직접 메모리 블록을 제공해 줄 수 있으며 이에 따라 읽기 요청의 지연시간을 줄일 수 있다.

컴파일러에 의한 메모리 갱신의 장점은 컴파일러가 프로그램을 분석하여 갱신의 시점을 결정하기 때문에 예측을 잘못해서 생기는 부담이 없다는 것이다. 그러나 이 방법의 단점으로는 컴파일러에 부담을 준다는 것과 추가적인 명령어의 수행이 필요하다는 것이다. 특히 update라는 새로운 명령어가 들어가야 하므로 기존의 실행코드를 그대로 사용할 수 없고 모든 코드를 다시 컴

파일해야 한다는 커다란 단점이 있다.

3. 적응적 메모리갱신을 이용한 디렉토리 프로토콜

본 장에서는 읽기 실패에 따른 지연시간을 줄이기 위해 적응적 메모리갱신 기법을 이용한 디렉토리 프로토콜을 소개한다. 먼저 시스템의 구성에 관련한 기본 가정을 기술하고 새로운 프로토콜의 동작과정을 상세히 설명할 것이다.

3.1 시스템 구성

- 시스템 유형 : CC-NUMA
- 캐쉬 일관성 프로토콜 유형 : 디렉토리 프로토콜
- 캐쉬 컨트롤러 : 프로그램이 가능한 전용 프로세서
- 상호연결망 유형 : 2차원 메쉬
- 라우팅 기법 : 웜홀 라우팅(wormhole routing)
- 라우팅 알고리즘 : 결정적 라우팅(deterministic routing)
- 동기화 기법 : 대기행렬 기반 록(Array-Based Queuing Lock)

디렉토리 프로토콜은 버스가 아닌 직접 상호연결망을 사용하는 다중처리기에서 일반적으로 사용되는 캐쉬 일관성 프로토콜이다. 프로그램이 가능한 전용 프로세서로 이루어진 캐쉬 컨트롤러는 복잡한 프로토콜을 쉽게 구현할 수 있다는 장점이 있어서 최근 여러 다중처리기 시스템에서 채택되고 있다[4]. 2차원 메쉬도 구조가 간단하고 확장성이 좋은 상호연결망 유형으로 알려져 있다. 웜홀 라우팅은 패킷을 전달하는데 있어서 전송 최소 단위의 플릿으로 작게 나누어 파이프라인 형태로 전송되는 방법으로 지연시간이 노드간의 거리 차에 큰 영향을 받지 않는다는 장점을 가지고 있고, 메쉬 등과 같은 구조가 간단한 상호연결망에서 주로 사용되는 라우팅 기법이다[5]. 결정적 라우팅은 시작노드와 목적노드가 정해짐과 동시에 경로가 정해지는 방법이다. 대기행렬 기반 록 (Array-Based Queuing Lock)의 동기화는 동기화과정에서 통신량이 적고 고르게 발생하여 대규모 병렬 시스템에서 많이 사용되는 방법이다.

3.2 기존 프로토콜에서의 메모리 읽기

본 절에서는 무효화 기법을 사용하는 기존의 디렉토리 프로토콜에서의 메모리 읽기에 대하여 살펴본다. 디렉토리 프로토콜의 디렉토리 상태는 기본적으로 공유상태(shared state)와 독점상태(exclusive state)로 나눌 수 있다. 독점상태에서 메모리 블록을 독점하고 있는 노드를 오너노드라고 하며, 오너노드(owner node)가 되기 위해서는 오너쉽(ownership)을 가져야만 한다. 이와 같은 디렉토리 상태를 갖는 프로토콜에서 메모리 읽기는 다음과 같이 이루어 질 수 있다(그림1참조). 지역노드는 자신의 캐쉬에 유효한 메모리 블록이 없을 경우 홈노드로 메모리 요청을 한다. 이 때 디렉토리 상태가 공유상태이면 요청 받은 메모리 블록을 지역노드에 보내준다. 반면에 디렉토리 상태가 독점상태이면 오너노드에 있는 오너쉽을 빼앗아서 디렉토리 상태를 공유상태로 만든 후에 해당하는 메모리 블록을 지역노드에 보내준다. 메모리 쓰기가 일어났을 경우는 디렉토리 상태를 독점상태로 만들면서, 쓰기 요청을 한 노드에게 오너쉽을 준다.

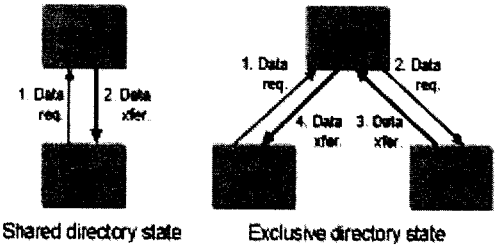

그림 1: 기존 프로토콜에서의 메모리 읽기 과정

3.3 적응적 메모리갱신 기법

본 절에서는 적응적 메모리갱신 기법을 사용하는 프로토콜이 동작하는 방법에 대하여 살펴본다.

3.3.1 적응적 메모리갱신 기법의 원리

기존의 프로토콜에서는 독점상태에서 메모리 읽기가 일어났을 경우 메모리블록을 가져오기 위하여 4개의 메세지가 필요하다(그림1). 이 중 홈노드와 오너노드 사이의 메세지는 읽기 접근시간이 커지는 주된 요인으로 작용하고 있다. 따라서 본 논문에서는 적응적 메모리갱신 기법을 이용하여 이를 줄이고자 한다.

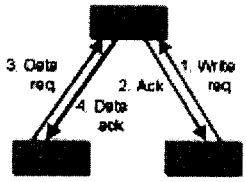

그림 2: 적응적 메모리갱신 기법에서 읽기

그림2은 오너노드에 의해서 메모리 쓰기가 일어나 후 다른 노드가 그 블록을 읽어가는 경우를 나타내고 있다. 이 때 오너노드가 홈노드의 메모리 블록을 갱신시켜주면, 다음 읽기요청에 대하여 홈노드는 오너노드의 도움없이 데이타를 보내줄 수 있다. 이와 같은 방법을 이용하면 읽기를 위한 메세지가 2개로 줄어들게 되고, 읽기 접근시간은 줄어들게 된다.

3.3.2 적응적 메모리갱신 기법에서의 쓰기 정책

적응적 메모리갱신 기법은 두 가지의 메모리 쓰기 정책을 가지고 있다. 어느 정책을 쓸 것인가는 쓰기 시점의 쓰기 정책 상태(write-policy state)에 따라 결정된다. 이 장에서는 적응적 메모리갱신의 두 가지 쓰기 정책을 위한 쓰기 정책 상태와 쓰기 정책 상태간의 전이에 대하여 알아본다.

1. 독점적 쓰기 상태

   독점적 쓰기 상태(exclusive-write state)에서는 일반적인 캐쉬 무효화(cache invalidation) 방법과 같은 쓰기 정책을 취한다. 메모리 블록에 대해 쓰기가 일어날 경우, 다음과 같은 동작이 일어난다.

   - 메모리 블록을 공유하고 있는 모든 노드에게 메세지를 보내서 캐쉬의 복사본을 무효화시킨다.
   - 디렉토리 상태를 독점상태로 만든다.
   - 쓰기를 요청한 노드는 메모리 블록에 대한 오너쉽을 가져서 오너노드가 된다.

   따라서 쓰기과정이 완료되면 오너노드의 캐쉬에만 유효한 메모리 블록이 존재하게 된다. 오너노드는 오너쉽의 반환을 요청 받을 때까지 자신의 캐쉬에 있는 메모리 블록을 언제라도 읽고 쓸 수 있으면, 그 사실을 아무에게도 알릴 필요가 없다.

2. 메모리 갱신 상태

   메모리 갱신 상태(memory-update state)에서는 쓰기가 일어날 때마다 홈노드의 메모리 블록을 갱신시킨다. 메모리 블록에 대해 쓰기가 일어날 경우, 다음과 같은 동작이 일어난다.

   - 메모리 블록을 공유하고 있는 모든 노드에게 메세지를 보내서 캐쉬의 복사본을 무효화시킨다.
   - 홈노드의 메모리를 갱신시킨다.
   - 디렉토리 상태를 공유상태로 한다.
   - 쓰기를 요청한 노드는 메모리 블록의 오너쉽을 가져가지 않는다.

   메모리 갱신 상태에서는 쓰기 요청을 한 노드가 오너쉽을 가져가지 않으므로 오너 노드가 될 수 없고, 자신의 캐쉬에 있는 데이타를 갱신할 수 없다. 따라서 모든 쓰기에 대하여 홈노드

로 쓰기 요청을 해야 한다. 그러나 메모리 갱신 상태에서는 홈노드가 항상 유효한 메모리 블록을 지니고 있으므로 읽기 요청을 받은 즉시 데이터를 보내줄 수 있다는 장점이 있다.

3. 상태의 전이

독점적 쓰기 상태에서는 한 노드에 의한 연속적인 쓰기가 발생할 경우 홈노드에 메모리 쓰기요청을 하지 않아도 된다는 장점이 있다. 반면에 다른 노드에서 읽기 요청이 발생할 경우 오너노드로부터 유효한 메모리 블록을 받아와야 한다는 단점이 있다. 메모리 갱신 상태에서는 메모리 쓰기 후에 다른 노드로부터 발생하는 읽기에 대해 홈노드가 직접 데이타를 전달해 줄 수 있다는 장점이 있지만, 모든 쓰기에 대해 홈노드로 요청을 해야 한다는 단점이 있다.

이런 상황에서 쓰기 정책에 대한 예측을 잘못할 경우 전체적인 성능이 저하되는 결과를 가져오게 된다. 따라서 쓰기 정책은 메모리에 대한 읽기,쓰기 유형을 잘 관찰해서 결정되어야 하며, 상황에 따라서 바뀔 수 있어야 한다.

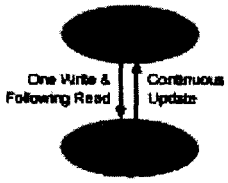

그림 3: 적응적 메모리갱신 기법의 쓰기정책 상태전이도

그림3는 쓰기 정책의 상태전이를 나타낸다. 독점적 쓰기 상태에서 어떤 메모리 블록에 대하여 쓰기가 한 번만 일어나고, 바로 이어서 다른 노드에 의해 읽기가 일어날 경우는 메모리 갱신 상태로 바뀐다. 또한 메모리 갱신 상태에서 한 노드에 의해 연속적인 메모리 갱신이 일어나면 독점적 쓰기 상태로 바뀐다.

4. 성능 평가

본 장에서는 모의 실험을 통하여 본 논문이 제안하는 적응적 메모리갱신 기법을 사용하는 프로토콜의 성능을 측정한다.

4.1 모의 실험 환경

본 논문의 모의 실험에서는 MINT 모의 실험기를 사용하였다[6]. MINT는 MIPS 프로세서로 제작한 병렬프로세서 시스템을 모의실험하는 프로그램- 구동 모의실험기로서 프로그램을 직접 수행하며, 사용자는 모의 실험에 필요한 상호연결망의 동작과 캐쉬 일관성 유지 프로토콜을 추가하여 실험할 수 있다. 상호연결망은 8x8 메쉬로 이루어져 있으며, 연결망 내부에 웜(worm)들이 돌아다니면서 진행(progress)하거나 정지(block)하도록 구현하였으므로, 통신량에 따른 지연시간을 정확하게 측정할 수 있다.

본 논문의 모의실험을 위해서 사용한 응용 프로그램들은 SPLASH-2[7]의 FFT, Radix, Barnes이다.

4.2 메모리 읽기 지연시간의 비교

그림4는 적응적 메모리갱신 기법을 사용했을 때 메모리 읽기 지연시간의 감소를 나타낸다.

Radix는 적응적 메모리갱신 기법을 사용하기 좋은 메모리 접근 유형을 많이 갖는 프로그램으로 약 13.2%의 성능 개선을 나타내었다. 적응적 메모리갱신 기법을 사용하기 좋은 메모리 접근 유형이란 한 프로세서가 쓰기를 실행한 후 연이어 다른 프로세서에서 읽기가 일어나는 빈도가 높으며, 메모리 접근 유형이 일정해서 쓰기 정책의 상태 전이가 잘 일어나지 않는 프로그램이다. 또한 FFT도 적응적 메모리갱신 기법을 이용했을 때 비교적 많은 성능개선이 있는 프로그램이다. 하지만 Barnes는 적응적 메모리갱신이 잘 적용되지 않는 프로그램으로 성능의 개선이 거의 없었다.

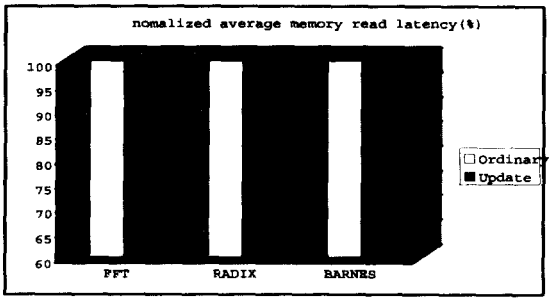

그림 4: 메모리 읽기의 평균 지연시간

5. 결론

분산 공유 메모리를 가지는 병렬처리 시스템에서 공유메모리에 대한 읽기는 큰 지연시간을 필요로 한다. 이러한 시스템에서 사용되는 캐쉬 일관성 유지 프로토콜은 메모리 읽기 지연시간을 줄일 수 있도록 설계되어야 한다.

본 논문에서는 적응적 메모리갱신을 이용하여 메모리 읽기 지연시간을 줄일 수 있는 캐쉬 일관성 유지 프로토콜을 제안하고 성능을 평가하였다. 적응적 메모리갱신 방법은 공유 메모리의 읽기 쓰기 패턴을 분석하여서 한 노드가 연속적인 쓰기를 하지 않을 경우에는, 메모리 쓰기 시 홈노드의 메모리를 갱신하고 오너쉽을 가져오지 않는 방법이다. 모의실험 결과 적응적 메모리 갱신 기법은 최대 13.2%까지 메모리 읽기 접근 시간이 감소하는 것으로 확인되었다.

참고 문헌

[1] A. Smith, "Cache Memories," *ACM Computing Surveys*, pp. 473–530, 1982.

[2] D. E. Lenoski, J. P. Laudon, K. Gharachorloo, A. Gupta, and J. L. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 148–159, 1990.

[3] A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data," in *Proceedings of 20th Annual International Symposium on Computer Architecture*, pp. 98–108, 1993.

[4] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The Stanford FLASH Multiprocessor," in *Proceedings of the 21st International Symposium on Computer Architecture*, pp. 302–313, Apr. 1994.

[5] X. Lin, P. K. Mckinley, and L. M. Li, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers," *IEEE Trans. on Parallel and Distributed Systems*, pp. 793–804, Aug. 1994.

[6] J. E. Veenstra and R. J. Fowler tech. rep., Rochester University, June 1993.

[7] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and methodological Considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 24–36, June 1995.