캐시 일관성 유지를 위한 전용 버스 시스템

천희식°, 김우완
경남대학교 컴퓨터공학과

A Dedicated Bus System for Cache Coherence

Hee-Sik Chun, Wu-Woan Kim
Dept. of Computer Engin., Kyungnam Univ.

요 약

멀티프로세서 시스템을 설계할 경우에는 공유메모리 구조와 메시지 전달방법의 두 가지의 패러다임을 바탕으로 하게 된다. 데이터 분할과 동적 부하 분산 문제를 단순화시킬 수 있으며 확장성을 용이하게 지원하는 장점을 가지고 있는 공유메모리 구조의 멀티프로세서 시스템에서 각 프로세서가 자신의 전용 캐시를 가지는 경우에는 메인 메모리와 이러한 전용 캐시내에 존재하는 데이터 사본간에 일관성 문제가 발생한다. 본 논문에서는 일관성 유지를 위해 제안되어 있는 여러 알고리즘 중 처리 노드와 고대역 저지연 인터커넥션 네트워크로 구성되는 공유메모리 구조의 멀티프로세서 프로토타입인 DASH 프로토콜을 지원하기 위한 전용 버스 시스템을 완전 개방형인 IEEE Futurebus+ 스탠다드에 준하여 설계한 다음, 이 시스템이 DASH 프로토콜을 지원하며 캐시의 일관성을 유지하기 위해 필요한 각종 행동과 기존의 범용 버스 시스템이 수행하는 행동의 병렬 처리를 지원할 수 있음을 시뮬레이션으로 증명한다.

1. 서론 - 캐시 및 캐시의 일관성 유지 필요성

일기예보, 핵융합 모델링, 항공기 관련 시뮬레이션 등을 포함하는 여러 분야에서는 여전히 현재 수준 이상의 고속 · 고성능 컴퓨팅 능력을 필요로 하고 있다. 이를 위한 여러 대안 중 비용효과적인 방법으로 성능 및 속도를 향상시키는 방법으로 특히 주목받고 있는 공유메모리 구조의 멀티프로세서 시스템에서는 저가의 멀티프로세서를 복수개 사용하는 경제적인 방식으로 속도와 성능관련 문제를 주로 해결하게 된다[1]. 반면, 메모리 점유 경쟁, 인터커넥션 네트워크상의 통신 경쟁, 그리고 메모리 요청이 네트워크를 통과하는 시간이 증가한다고 하는 지연 등의 문제점 역시 발생하게 되는데, 이에 대처하기 위한 방안중의 하나가 바로 캐시 메모리(cache memory)이다. 공유 메모리를 가지는 멀티프로세서 시스템은 보편적으로 하나 이상의 캐시가 공유가능한 블록을 각각 복사하여 유지하게 된다. 이때, 메모리의 일관성을 유지하기 위해서는 이러한 복수의 사본이 서로 일치하는, 즉 동일한 상태를 유지할 수 있도록 해야 하는데 이가 바로 캐시 일관성 문제인 것이다. 캐시 시스템이 일관성을 유지한다라고 할 때는 임의의 특정 메모리 위치의 내용이 변경되는 경우 복수의 캐시내에 존재하는 이 메모리 위치에 대한 사본 모두가 일치하는 경우를 말하며, 캐시 일관성 유지 프로토콜이란 캐시의 일관성을 유지시켜 주는 메커니즘을 말한다

이하의 제2장에서는 이러한 캐시 일관성을 지원하는 여러 알고리즘을 살펴보며, 제3장과 제4장에서는 처리 노드와 고대역 저지연 인터커넥션 네트워크로 구성되는 공유 메모리 구조의 멀티프로세서 프로토타입인 DASH 프로토콜과 이를 위한 전용 버스 시스템의 준거가 되는 완전 개방형인 IEEE Futurebus+ Standards를 살펴본 다음, 제5장에서는 DASH 프로토콜을 지원하는 전용 버스 시스템의 설계와 검증을, 그리고 마지막 제6장에서는 댓음말을 싣기로 한다.

2. 캐시 일관성을 지원하는 알고리즘

캐시 메모리의 일관성을 유지하기 위하여 제안되어 있는 기법은 하드웨어적인 구현방법에서부터 캐시 일관성 유지 정책을 소프트웨어로 구현하여 다양한 하드웨어를 지원해 주는 방법에 이르기까지 여러 가지가 있다.

2.1 하드웨어 기반 프로토콜

이에는 캐시 무효화 및 갱신 명령을 동일한 해당 블록의 사본을 가지고 있는 캐시로 전송해 주는 스누피 캐시 프로토콜(snoopy cache protocols)[3], 일관성 유지 명령을 해당 블록의 사본을 가지고 있는 캐시로만 전송해 주는 디렉토리 기법(directory schemes)[6] 및 Cache-Coherent Network 구조[2] 등이 있으며, 각각은 일정한 캐시

일관성 유지 정책을 사용하고 있다. 이러한 기법에서의 하드웨어 메커니즘은 캐시의 불일치를 감지하면 하드웨어로 구현하고 있는 프로토콜에 준하여 일정한 행위를 수행하게 된다. 하드웨어 프로토콜은 임의의 특정 블록에 대한 복수개의 사본을 허용해 주는데, 이러한 캐시의 일관성 유지 정책은 크게 특정 프로세서의 전용 캐시가 해당 블록을 변경시키면 동일한 블록을 보유하고 있는 다른 프로세서는 자신의 전용 캐시내에 있는 해당 블록을 무효화시켜주는 write-invalidate기법과, 해당 블록을 무효화시키는 방법에 의존하는 것이 아니라 자신의 캐시내의 블록도 마찬가지로 갱신시켜 주는 write-update기법이라고 하는 두 가지로 다시 구분할 수 있다.

2.2 소프트웨어 기반 기법

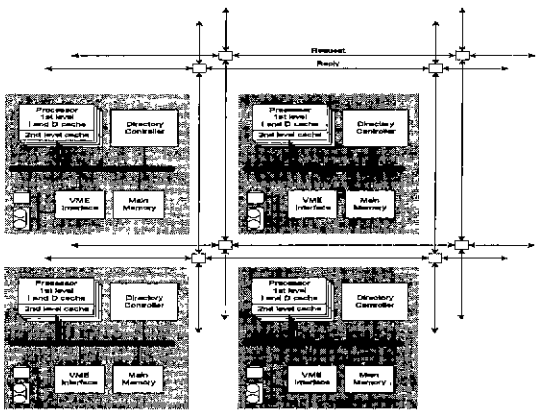
소프트웨어 기반의 기법[4]은 복잡한 하드웨어 메커니즘의 필요성을 배제하고자 한다 여기에는, 첫째, 소프트웨어기반의 불일치 캐시 사본을 방지하기 위해 해당 자료 구조의 캐싱이 허용가능한 시점에서만 이의 캐싱을 허용해 주는 방법이 있다. 이를 위해서는 변수를 캐시할 수 있는지의 여부를 프로그램이 나타내 주어야 하는데, 정교한 컴파일러나 프리프로세서가 이를 담당할 수 있다. 둘째로는, 컴파일러가 read-write 공유 자료 구조를 캐싱하여도 아무런 문제점이 발생하지 않는 시점을 분석해 내는 방법 등이 있다.

3. DASH 프로토콜

DASH란 공유 메모리 구조를 가지는 멀티프로세서 프로토타입으로써 처리 노드와 고대역 저지연 인터커넥션 네트워크를 그 구성 요소로 하고 있다.

3.1 DASH의 구조

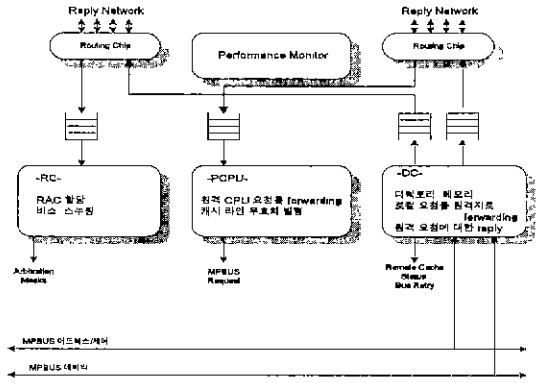
DASH 멀티프로세서는 그림 3과 같이 상용 버스를 기반으로 하는 멀티프로세서로써 한 노드는 4 대의 고성능 프로세서(Silicon Graphics POWER Station 4D/240)를 가진다[5]. 각 프로세서는 64 KB의 1차 명령어 캐시와 write-through 데이터 캐시를 가지며, 데이터 캐시는 read 버퍼 및 4 워드 용량의 write 버퍼를 통하여 256 KB의 2차 write-back 캐시와 접속시켜 준다. 1차 및 2차 캐시는 모두 direct-mapping 방식을 채택하고 있다.



<그림 1> 2×2 DASH 시스템의 블록 다이어그램

2차 캐시는 버스 스누핑을 담당하며, Illinos coherence protocol(Invalidate를 기반으로 하는 프로토콜)을 사용하여 일관성을 유지해 낸다.

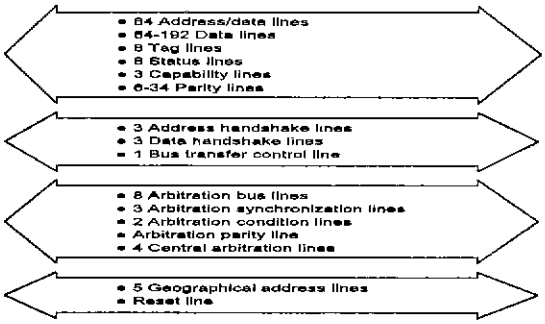
DASH 시스템은 노드간의 캐시 일관성을 유지하고 인터커넥션 네트워크와의 인터페이스를 담당하는 디렉토리 컨트롤러 보드를 장치하고 있다. 디렉토리 보드는 DC(Directory Controller), PCPU(Pseudo-CPU), RC(Reply Controller) 등을 단일 PCB로 구현하고 있다. 각 요소의 기능 및 역할은 다음 그림과 같다.



<그림 2> 디렉토리 보드 블록 다이어그램

4. IEEE Futurebus+ Standards

64 비트의 주소 공간, RISC가 요구하는 처리율, 차세대 멀티프로세서 구조 등을 지원하는 실질적인 오픈 시스템을 개발하기 위하여 여러 단체와 전문가들의 참여와 노력으로 지금도 계속 진화되고 있는 Futurebus+ Standard가 개발되어 있다. 이 스탠다드는 확장성을 지원하며 특정의 구조나 프로세서 기술에 종속되지 아니함을 내세우고 있어 컴퓨터 발전 단계에서 나타나게 될 향후 수 세대동안에도 안정적인 플랫폼을 제공해 줄 것으로 판측되고 있다.



<그림 3> IEEE Futurebus+ (Standard 896.1-1991)의 구조

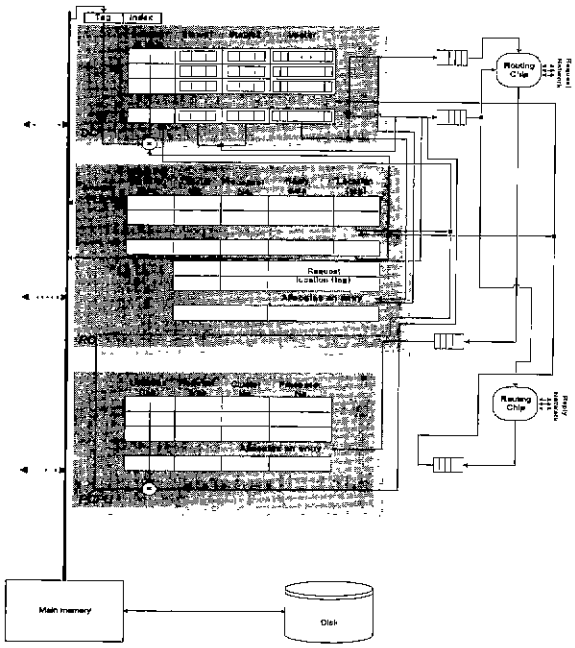
5. 버스 시스템 구현 및 검증

5.1 설계

<그림 4>는 본 논문에서 목표로 하고 있는 DASH 프로토콜을 지

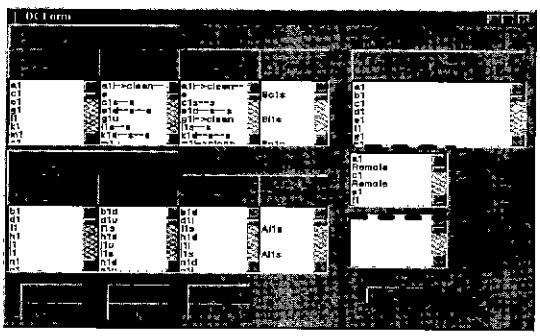
원하는 공유 메모리 멀티프로세서의 한 클러스터 중 위 <그림 2>의 디렉토리 보드 부분을 나타내는 버스 시스템이 된다

이 버스 시스템은, 다음 절에서 설명하고 있는 시뮬레이션 및 이의 결과가 말해주고 있듯이, 제3장에서 짚어본 바 있는 DASH 프로토타입을 지원하므로 해당 멀티프로세서 시스템이 공유 메모리 구조를 바탕으로 캐시의 일관성을 유지해 낼 수 있게 된다고 할 수 있다. 물론 이러한 버스 시스템은 여러 설계가능한 구조중의 한 방법이지 유일한 구현방법이라고 할 수는 없다.



<그림 4> DASH 프로토콜을 지원하는 전용 버스 시스템 구조

5.2 검증



<그림 5> 시뮬레이터 인터페이스
(Status1 · DC의 상태, Status2 : 해당 프로세서 캐시내의 상태, → : 일관성 유지 명령의 수행 전 · 후,
알파벳 : 메모리 블록, i : Invalid 상태, s : Shared 상태, d : Dirty 상태, v : Valid 상태)

지금부터는 "5.1"에서 설계한 버스 시스템의 동작 여부에 관한 시뮬레이션 및 이에 필요한 각종 설명을 부가하기로 한다.

앞서 설계한 버스 시스템이 과연 캐시 일관성 유지 프로토콜을 제대로 지원하는지의 여부를 파악하기 위해 본 논문에서는 DASH 프로토콜에서 등장하는 각 자료구조를 정의해 준 다음, 이러한 자료구조간의, 즉, 메인 메모리, DC, PCPU, RC, 등간의 캐시 일관성 유지 관련 제반 행동을 윈도우 및 윈도우 NT 배하에서 동작하는 C++ RAD(Rapid Application Development) 툴인 C++ Builder(Client/Server v1.0)를 바탕으로 하여 시뮬레이션하였다. <그림 5>는 이러한 시뮬레이션을 위해 개발한 사용자 인터페이스를 나타내는 동시에 본 시뮬레이션에 의한 결과를 나타내고 있다. 시뮬레이터를 개발할 때에는 임의의 프로세서가 특정 작업을 수행하고 있을 경우에는 독점적으로 사용가능한 자원의 범위를 최소화하려고 한다는 점을 충분히 반영하였으며, 이를 바탕으로 하여 시뮬레이션을 수행하였다.

시뮬레이터는 메인 메모리내의 각 블록을 프로세서 A와 B가 양분하여 가진다고 할 경우, 그리고 또 다른 임의의 프로세서(C)가 이러한 블록들을 메인 메모리내에서와 동일한 순서로 읽어야 할 필요가 있다고 할 경우의 프로세서 내에서의 각 블록의 상태 변화를 나타내고 있다. 점선으로 된 사각형은 "5.1"에서 설계한 전용 버스 시스템을 사용하는 프로세서 C에서의 시뮬레이션 결과와 각 블록 중 비공유상태의 블록을 원격지로부터 읽었는지의 여부를 나타내고 있다.

6. 결론

DASH 디렉토리 프로토콜은 스누피 프로토콜과 비교해 볼 때 단일의 대기점(serialization point)이 존재하지 않고 프로토콜 실행을 위해 인터랙션(interaction)을 수행할 필요가 있으며, 메모리 시스템이 계층화되어 복잡도의 증가 요인을 안고 있기는 하지만, 비트 벡터를 사용하여 메모리 블록을 캐싱하고 있는 원격 클러스터를 식별해내므로 확장성 문제를 보다 용이하게 지원해 준다. 본 논문에서는 DASH 프로토콜을 지원하는 전용의 버스 시스템을 IEEE Futurebus+ Standards에 준하여 설계하였다. 범용 구조가 아닌 캐시 일관성 유지 프로토콜을 지원해 주는 이러한 전용 버스 시스템은 그 전용성으로 인하여 보다 뛰어난 캐시 일관성 유지를 지원해 주는 장점을 가지고 있다. 또한 프로세서 보드로 메모리와 디렉토리를 분산시키면 이들에 대한 대역폭의 제약을 완화시킬 수도 있게 된다

참 고 문 헌

[1] P C Patton, "Multiprocessors Architecture and Applications," IEEE Computer, 18(60) 29-40, June 1985.

[2] Stenström, P., "A Survey of Cache Coherence Schemes for Multi-processors," IEEE Computer, Vol. 23, No 6(June 1990), pp. 12-24

[3] Karlin, A R, M S. Manasse, L Rudolph, and D D Sleator, "Competitive Snoopy Caching," Algorithmica, 3, 1988, pp 79-119

[4] IL Cheong, and A. Veidenbaum, "A Cache Coherence Scheme with Fast Selective Invalidation," Proc. 15th Int'l Symp. Computer Architecture, 1988, pp. 299-307

[5] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, " The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor, " In Proc of the 17th Annual Int. Sym. on Computer Architecture, May 1990

[6] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, P Gibbons, A Gupta, and J Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multi-processors," In Proc of the 17th Annual Int Sym on Computer Architecture, pp 15-26, May 1990