논문 2005-42CI-6-3

단일 공유 메모리를 가지는 다중 프로세서 시스템의 원격 캐시 일관성 유지 프로토콜

(A Remote Cache Coherence Protocol for Single Shared Memory in Multiprocessor System)

김 성 운*, 김 보 관**

(Seongwoon Kim and Bogwan Kim)

요 약

다중 프로세서 구조는 컴퓨터 성능을 향상시키기 위한 좋은 방법이다. 물리적으로 분산된 메모리를 단일 공유 메모리 공간으로 제공하는 CC-NUMA(Cache Coherent Non-Uniform Memory Access) 시스템은 다중 프로세서 컴퓨터 시스템으로 널리 사용된다. CC-NUMA는 공유 메모리 지원을 위해 풀맵 디렉토리를 가지며, 빠른 원격 메모리 접근을 위해 원격 캐시 메모리를 사용한다. 본 논문은 CC-NUMA 시스템을 구성할 수 있는 프로세싱 노드 구조와 이러한 구조에 적합한 캐시 일관성 유지 프로토콜을 제안하여, 대량의 프로세서를 이용한 다중 프로세서 시스템의 구성을 용이하게 한다. 끝으로 제안된 프로토콜에 따른 시스템 구현 결과도 제시한다.

Abstract

The multiprocessor architecture is a good method to improve the computer system performance. The CC-NUMA provides a single shared space with the physically distributed memories is used widely in the multiprocessor computer system. A CC-NUMA has the full-mapped directory for the shared memory and uses a remote cache memory for the fast memory access. In this paper, we propose a processing node architecture for a CC-NUMA system and a cache coherency protocol on the physically distributed but logically shared system. We show an implementation result of the system which is adopted the cache coherency protocol.

Keywords: processor, cache, memory, CC-NUMA

I. 서 론

지난 십여 년 동안 고성능 컴퓨팅 사회는 다중 프로세서 시스템을 중심으로 발전하고 있다. 이러한 변화는 고성능의 마이크로프로세서가 상업적으로 저렴하게 공급됨에 따라 급격히 이루어지고 있다[1]. 대량의 프로세서를 통한 병렬 처리를 위해서는 프로그램밍 방식이 중요한 이슈가 된다. 일반적으로 프로그래머는 모든 프로세서에서 모든 데이터를 접근하고 싶어 하며, 메시지 패싱을 통한 접근보다 공유 메모리 프로그래밍 방식으로 데이터를 다루고 싶어 한다. 따라서 다중 프로세서 시스템이 직면하는 중요한 문제 중의 하나는 프로세스들이 공유변수를 통하여 통신할 수 있는 공유 메모리 프로그램 모델을 제공하는 것이다. 최근 들어 기술이 발전함에 따라 물리적으로 분산된 메모리를 가지는 시스템 상에서 공유 어드레스 개념을 지원하려는 시도가 많이 있다[2][3].

대표적인 공유 메모리 구조는 버스 기반의 대칭형 다

* 정회원, 한국전자통신연구원 서버플랫폼연구팀
(Server Platform Research Team, ETRI)
** 정회원, 충남대학교 전자공학과
(Dept. of Electronic Engineering, Chungnam Univertity)
※ 본 연구는 정보통신부의 선도기술개발사업의 연구결과로 수행되었음.
접수일자: 2005년6월3일, 수정완료일: 2005년11월3일

중프로세서 시스템이다[4][5]. 이러한 시스템에서 모든 프로세서는 버스를 통하여 모든 메모리를 접근할 수 있다. 그러나 하드웨어의 관점에서 프로세서의 개수가 적을 때에는 효율적으로 구성을 할 수 있으나, 프로세서의 개수가 많아지면 연결 방법도 힘들 뿐 아니라 비용도 급격히 증가한다. 따라서 성능을 향상시키기 위해 많은 프로세서를 연결할 때는 물리적으로 프로세서와 메모리들이 분산된 환경에서 메모리를 공유할 수 있는 구조가 이루어져야 한다. 이를 구현하기 위하여 시스템은 하이브리드 형태의 구조를 형성한다. 즉 그림 1과 같이 메모리를 내장한 독립된 프로세싱 노드들을 네트워크를 통해 메모리를 공유할 수 있는 구조로 구성하는 것이다. 하이브리드 프로세싱 노드를 구성한 시스템에서는 여러 개의 프로세서에서 동시에 발생되는 다양한 형태의 프로세서의 메모리 접근 요구로 인해 단일 공유 메모리를 유지하는데 많은 복잡한 상황을 발생된다.

본 논문은 이러한 물리적으로 분산되었으나 논리적으로 단일 공유 메모리를 제공하는 CC-NUMA 시스템에서 사용되는 캐시 일관성 유지 프로토콜을 제안한다. 이 프로토콜은 하이브리드 형태의 원격 접근 캐시 메모리를 내장하는 프로세싱 노드에서 적용이 가능하며, 다양한 프로세서의 접근 형태에 대한 캐시 일관성을 위한 동작 방법과 복잡한 캐시 상태를 모든 경우에 상세히 정의하여 대량의 프로세서를 이용한 CC-NUMA 시스템을 쉽게 구현할 수 있도록 한다.

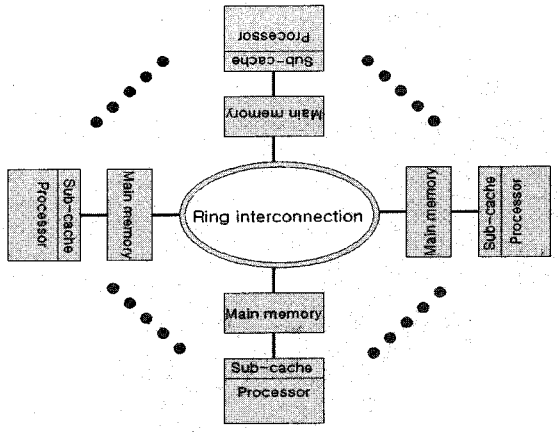


그림 1. 링 기반의 PDLS 시스템
Fig. 1. Ring-based PDLS System.

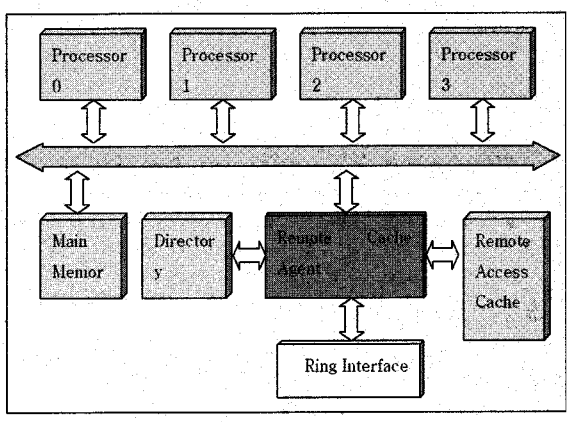


그림 2. 원격 접근 캐시 메모리를 내장하는 프로세싱 노드 구조
Fig. 2. Processing Node with Remote Access Cache.

Ⅱ. CC-NUMA 시스템의 프로세싱 노드 구조

CC-NUMA 시스템은 여러 개의 프로세싱 노드들이 물리적으로는 분산되어 있고, 프로세싱 노드 내에 존재하는 메모리들도 분산되어 있다. 그러나 분산된 메모리들이 마치 하나의 연속된 메모리 주소공간을 가지는 것처럼 논리적으로 공유된 메모리 시스템 (PDLS, Physically Distributed but Logically Shared)을 유지한다. 이러한 시스템은 두 가지 중요한 시스템 구조와 설계 요소를 가진다. 즉 상호연결망 구조와 메모리 시스템 구조이다. 메모리 모듈, 캐시, 프로세서 노드의 연결 방식이 어떤 구조를 가지는 가에 따라 메모리 접근 시간이 크게 차이 난다[6][7].

링 기반의 CC-NUMA 시스템은 전역 공유 메모리를 접근하려는 프로세서들의 요구가 링 연결망을 통해 집중 되는 정도에 따라 시스템의 특징이 결정된다. 이러한 경우에 시스템은 프로세싱 노드를 연결하는 상호연결망 링, 핫 스팟이 발생되는 핫 노드와 그렇지 않은 쿨 노드로 나뉘어 진다. 이런 경우에 핫 노드에 대한 다수개의 프로세서가 핫 노드의 원격 메모리에 대한 모든 프로세서의 평균 접근 시간은 (1) 식과 같이 나타 낼 수 있다.

$$\bar{T}_{ccnuma}=\frac{t_{local}}{N}+\frac{N-1}{N}(\bar{d}_{source}+\bar{q}_{ring\_port}+\bar{d}_{dest}+t_r) \quad (1)$$

여기서, N은 링에 연결된 프로세싱 노드의 개수 이다. (1) 식에서 나타난 바와 같이 전체 CC-NUMA 공유 메모리에 대한 접근 시간은 핫 노드의 메모리 접근 시간과 쿨 노드의 프로세서들이 유효한 데이터 (핫 노드에 위치) 접근 시간의 합으로 이루어진다.

쿨 노드의 유효 데이터 접근 시간은 근원지 메모리에서 링으로의 평균 큐잉 시간, 링 전송의 평균 큐잉 시간, 핫 노드의 링 인터페이스 큐잉 시간 과 근원지 노드로의 Ack 패킷 응답시간의 합으로 구성된다.

식 (1)에 나타난 바와 같이 유효 데이터 접근 시간은 핫 노드에 대한 원격 접근 시간에 크게 좌우가 됨을 알 수 있다. 따라서 이러한 원격 접근 시간을 줄이기 위해서 그림 2와 같이 원격 접근 캐시를 각 프로세싱 노드에 두어, 원격 메모리 접근 시간을 크게 줄일 수 있다. 따라서 그림 2와 같이 시스템 성능을 향상시킬 수 있는 원격 캐시 에이전트를 내장하는 프로세싱 노드를 대상으로 원격캐시 일관성 유지 프로토콜을 제안한다.

Ⅲ. 원격 캐시 일관성 유지 프로토콜

PDLS 시스템은 그림 3과 같이 상호 연결망을 통하여 여러 개의 프로세싱 노드들이 연결된 CC-NUMA 시스템의 일종이다. 그림에서와 같이 프로세싱 노드는 프로세서(P), 프로세서 고유의 캐시 메모리(PC), 메모리 모듈(MEM), 원격 캐시 메모리(RAC), 디렉토리(DIR), 그리고 연결망 인터페이스(NIF)로 연결 되어 있다.

프로세서 캐시 일치성은 무효화 기반의 스눕 캐시 일관성 프로토콜(invalidation-based snoopy cache coherency protocol)에 의해 유지된다[8][9]. 프로세서가 요청한 유효한 데이터가 다른 프로세서 캐시나 지역 메모리에 있을 경우에는 해당되는 캐시 라인을 읽어 들린다. 즉 프로세싱 노드 내의 지역(near) 메모리에 위치가 맵핑되는 경우에는 메모리가 해당 데이터를 제공한다. 그러나 지역 메모리에 맵핑이 되지 않는 경우는 두 가지 동작이 일어난다. 먼저, 원격 캐시 메모리에 맵핑 되고 원격 캐시 메모리가 유효한 데이터를 가지고 있으면 원격 캐시 메모리에서 제공이 되나, 그렇지 않으면 상호연결망을 통해 연결되어 있는 다른 프로세싱 노드의 원격 (far) 메모리에서 유효 데이터를 읽어 와서 제공된다.

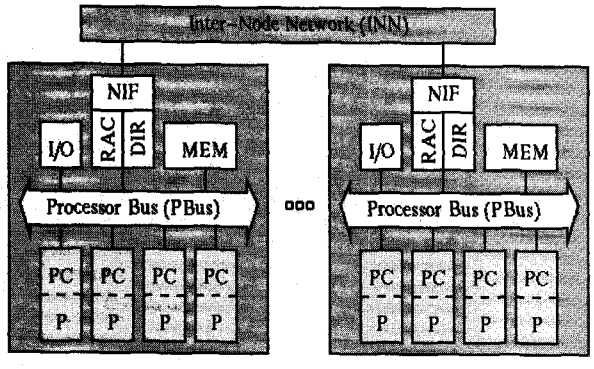


그림 3. PDLS 시스템 구성도
Fig. 3. PDLS System Configuration.

1. 메모리 접근 트랜잭션

프로세서에 의해 메모리를 접근하는 트랜잭션은 프로세싱 노드 내부에서 진행되는 프로세서 버스 트랜잭션과 상호연결망을 통한 네트워크 트랜잭션으로 구분할 수 있다. 프로세싱 노드 내부에서 진행되는 버스 트랜잭션의 종류는 크게 네 가지 형태가 있으며, 표 1과 같이 정의한다.

표 2와 같이 연결망을 통한 네트워크 트랜잭션은 세 가지 형태가 있다. 캐시, 언캐시 그리고 I/O 트랜잭션이다. 캐시 트랜잭션은 캐시 어드레스 영역으로의 읽기나 쓰기 트랜잭션을 의미한다. 네트워크 트랜잭션은 요청과 응답 트랜잭션으로 구성되며, 요청 트랜잭션은 어드레스를 동반하며 쓰기인 경우에는 데이터도 같이 제공된다. 응답 트랜잭션은 요청에 대한 처리 결과를 알려준다.

표 1. 프로세서 버스 트랜잭션
Table 1. Network Transactions.

종류	트랜잭션 이름			
	읽기		쓰기	
cached	coherent read	BusCoRd	writeback	BusWrb
	exclusive read	BusExRd	write-through	
	invalidate	BusInv		BusWrt
uncached	uncached read	BusUcRd	uncached write	BusUcWr
locked	locked read	BusLocRd	locked write	BusLocWr
input/output	i/o read	BusIoRd	i/o write	BusIoWr

표 2. 네트워크 트랜잭션
Table 2. Network Transactions.

종류	트랜잭션 이름	동작	
		요청	응답
cached	coherent read	CRDq	CRDp♦
	exclusive read	ERDq	ERDp♦
	invalidate	INVq	INVp
	writeback	WRBq	WRBp
	write through	WRTq♤	WRTp
uncached	uncached read	URDq	URDp♦
	uncached write	UWRq♤	UWRp
input/output	input/output read	IORq	IORp♦
	input/output write	IOWq♤	IOWp

♦) 데이터를 반드시 동반하는 트랜잭션
♤) 데이터를 동반하는 경우와 동반하지 않는 경우가 있는 트랜잭션

2. 메모리 계층별 캐시 상태

CC-NUMA 시스템의 각 프로세싱 노드에는 캐시 일관성을 유지를 위해 따로 관리해야 하는 세 개의 메모리 계층이 구성된다. 즉 프로세서 캐시와 원격 캐시 그리고 디렉토리이다. 각 메모리 계층의 캐시는 일관성을 유지하기 위해 고유의 상태를 가지고 있다. 프로세서 캐시 상태는 노드 내의 프로세서 간 캐시 일관성을 위해 필요하며, 원격 접근 캐시 상태는 노드 간의 캐시 일관성을 위해 필요하며, 디렉토리는 원격 캐시 제어기에 연결되어 외부의 메모리 요청에 대하여 프로세서의 캐시 상태를 알려 주어 노드가 캐시 일관성을 유지할 수 있도록 해 준다. 각 메모리 계층별 캐시 상태는 표 3과 같다. 프로세서 캐시 상태의 변화는 MESI 프로토콜에 따라서 구성되고, RAC 와 DIR의 캐시 상태의 변화는 여기서 제안된 프로토콜에 따라 구성된다.

프로세서의 관점에서 원격 접근 캐시는 프로세서가 원격 메모리 접근 시에 빠르게 원격 데이터 제공해 주기 위한 캐시이다. 프로세서가 접근하는 데이터의 원격 접근캐시의 상태가 I 인 경우에는 연결망을 통해서 데이터를 가져와서 원격 접근 캐시에 저장한다. S인 경우에는 원격 노드의 메모리와 동일한 값을 가지며, M인 경우에는 원격 캐시가 원격 노드의 메모리보다 최근 값을 가지고 있다. L은 해당 블록이 인터록된 값을 가지고 있음을 의미한다. 이 외에도 원격 접근 캐시는 펜딩 비트를 가지고 있다. 해당 블록에 대한 전송이 진행 중임을 의미하며, 이런 경우에는 펜딩 비트가 해제 될 때까지 해당 캐시 라인의 접근을 진행할 수 없다. RAC가 M 상태인 경우에는 치환이 일어나지만 RAC가 S 상태에는 통지도 없이 치환이 일어난다. 이로 인해 디렉토리는 고스트 공유가 생길 수 있다.

디렉토리의 각 엔트리는 상태 비트와 존재 비트로 구성된다. 존재 비트는 공유 노드의 위치를 지정하고 상태 비트는 원격 접근 캐시들 간의 상태 정보를 공유하기 위해 구성된다. U는 메모리 블록이 유효하고 다른 원격 캐시는 복사본을 가지고 있지 않음을 표시한다. S는 메모리 블록이 유효하고 다른 원격 캐시가 Read-Only 복사본을 가지고 있을 수 있음을 표시한다. M은 메모리 블록이 홈 메모리에 더 이상 유효한 상태로 있지 않고 원격 캐시가 Writeable Exclusive 복사본을 가지고 있음을 의미한다. 또한 원격 접근 캐시와 마찬가지고 펜딩 비트를 가지고 있어, 현재 해당 캐시라인의 접근이 진행 중임을 표시한다.

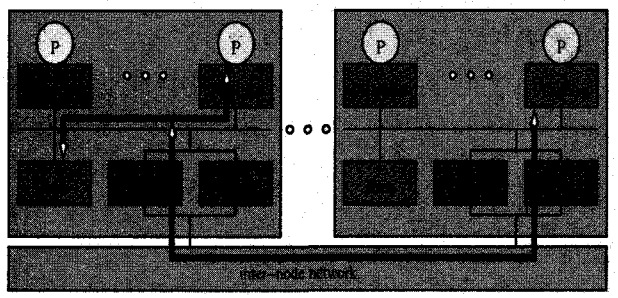
표 3. 메모리 계층별 캐시 상태
Table 3. Cache States in Memory Hierarchy.

캐시 종류	캐시 상태
프로세서캐시	Invalid(I), Shared(S), Exclusive(E), Modified(M)
원격접근캐시 (RAC)	Invalid(I), Shared(S), Modified(M), Locked(L)
디렉토리 (DIR)	Uncached(U), Shared(S), Modified(M)

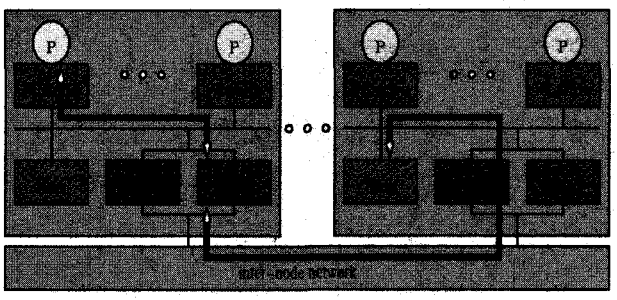
3. 메모리 계층 간의 접근 방법

프로세서에 의해 데이터 미스가 발생하게 되면 그 요청은 데이터가 로컬인지 원격에 있는 지를 확인하기 위해 메모리 블록의 어드레스를 본다.

그림 4(a)와 같이 어드레스가 로컬 메모리 영역이면 로컬 메모리에서 데이터가 제공되고, 원격지에 데이터가 있으면 원격지로부터 데이터가 공급된다. 요청된 캐시라인이 로컬 메모리 영역일지라도 원격 노드에서 해당 데이터를 가지고 있는 경우가 있기 때문에, 항상 메모리 접근 트랜잭션은 로컬 디렉토리의 상태를 확인해야 한다. 그림 4(b)와 같이 원격지로부터 데이터를 가져와야 하는 경우에는 원격지에서 전송된 데이터를 RAC에 저장하고 데이터를 해당 프로세서에게 넘겨준다. 또한 요청된 트랜잭션에 따라서 RAC 내에 적절한 캐시 상태를 유지한다.



(a) 로컬 접근



(b) 원격 접근

그림 4. 로컬 및 원격 접근
Fig. 4. Local and Remote Access.

4. 노드 내부와 노드 간의 트랜잭션 관계

단일 어드레스 영역을 가지는 공유 메모리를 유지하기 위한 원격 캐시 일관성 유지 프로토콜은 다음과 같은 상태를 가정하고 제안된다.

- 원격캐시 미스를 인한 모든 원격 접근은 제일 먼저 홈 노드로 트랜잭션이 전송된다.
- 홈 노드는 전체 메모리 접근 순서를 결정하기 위해 접근되는 순서대로 메모리 트랜잭션을 진행한다.
- 변경된 블록에 대한 요청은 홈 노드에 의해 블록을 소유하고 있는 노드로 응답을 넘긴다. 블록을 소유하고 있는 노드는 로컬 노드에 요청된 블록을 제공하기 위해 직접 응답하고 홈 메모리와 디렉토리를 수정한다. 그러나 응답이 불가능한 경우에는 홈 노드에만 보고한다.
- 네트워크를 통한 노드간의 데이터 전송 프로토콜은 메시지의 순서가 보장된다. 즉, 근원지에서 보내진 메시지는 보낸 순서대로 목적지에 도착한다. 즉 동일한 목적지를 가지고 전송되는 메시지는 다른 순서를 가지지 않는다.
- 고스트 공유를 허용한다. 공유 상태로 있는 원격지 캐시는 홈 노드에 알리지 않고 치환이 일어날 수 있다. 이로 인해 고스트 공유가 발생하여 존재하지 않는 블록에 INV 요청이 발생할 수 있다.

프로세서에 의해 메모리 접근 요구를 위한 프로세싱 노드 한 개에서 볼 때, 프로세서에 의해 제기되는 트랜잭션과 다른 노드에 의해 제기되어 노드 내부로 전달되는 트랜잭션으로 나뉜다.

표 4, 5는 노드 내의 프로세서에 의해 제기된 트랜잭션에 따라 진행되는 네트워크 트랜잭션의 종류를 정의하고 각 트랜잭션의 진행결과 변하는 캐시의 상태를 나타낸다.

노드 내부 로컬 영역의 트랜잭션

표 4는 프로세서가 로컬 영역에서 발생되는 모든 트랜잭션에 대하여 가능한 DIR, RAC 상태와 해당 트랜잭션에 대하여 DIR, RAC의 상태변화를 나타낸다.

BusCoRd 트랜잭션이 발생하면 DIR 상태는 U, S, M 상태로 존재할 수 있으며, U 와 S 상태인 경우에는 네트워크로 트랜잭션이 발생하지 않고, M 상태에서는 네트워크(즉, 다른 프로세싱 노드를 향함)로 CRD 트랜잭션이 일어난다. 또한 BusCoRd 트랜잭션이 완료 되면 DIR, RAC 상태는 각각 S 로 변한다.

이때 M 상태에서 발생되는 BusCoRd은 원격 노드가 M 상태의 데이터를 가지고 있기 때문에 먼저 Writeback 동작이 수행되어야 한다. 그림 5는 이러한 경우의 트랜잭션 흐름과 캐시 상태를 표시하고 있다. 데이터를 가지고 있는 원격 노드는 최신의 데이터를 요청한 홈 노드에게 돌려준다. 트랜잭션이 완료된 후의 최종 캐시 상태는 모두 S로 바뀐다. 트랜잭션이 진행되는 동안에는 그림 5에서와 같이 펜딩 상태로 두어 이때 다른 노드에 의해 해당 라인에 대한 접근을 막아서 캐시 일관성을 유지할 수 있도록 한다. 프로세서 캐시 라인이 원격 접근 캐시 블록보다 작은 경우에는 CRD 동

표 4. 프로세서 로컬 트랜잭션으로 인한 상태 변화
Table 4. State Transitions by Local Transaction.

프로세서 메모리 접근	현재 DIR 상태	DIR에 의한 네트워크 동작	RAC의 현재 가능한 상태	최종상태		비고
				DIR	RAC	
BusCoRd	U	없음	I	U	I	
	S	없음	S	S	S	
	M	CRD	M	S	S	writeback
BusExRd	U	없음	I	U	I	
	S	INV	S	U	I	
	M	ERD	M	U	I	writeback
BusInv	U	없음	I	U	I	
	S	INV	S	U	I	
	M	ERD	M	U	I	발생 않음
BusWrb	U	없음	I	U	I	발생 않음
	S	INV	S	U	I	발생 않음
	M	ERD	M	U	I	
BusWrt	U	없음	I	U	I	
	S	INV	S	U	I	
	M	ERD	M	U	I	발생 않음
BusLocRd	U	없음	I	U	I	
	S	-	S	U	I	
	M	-	M	U	I	발생 않음
BusLocWr	U	없음	I	U	I	
	S	-	S	U	I	발생 않음
	M	-	M	U	I	발생 않음
BusUcRd	U	없음	I	U	I	발생 않음
	S	-	S	S	S	발생 않음
	M	-	M	M	M	
BusUcWr	U	없음	I	U	I	발생 않음
	S	-	S	U	S	발생 않음
	M	-	M	U	M	

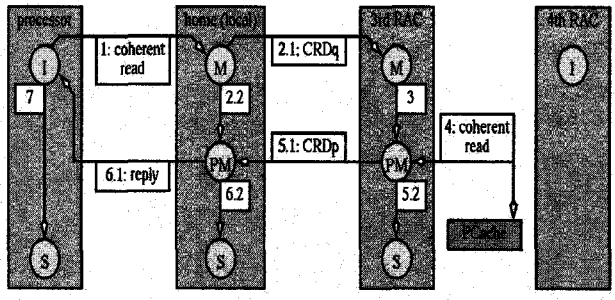


그림 5. 디렉토리 M에 대한 BusCoRd
Fig. 5. BusCoRd to Directory M.

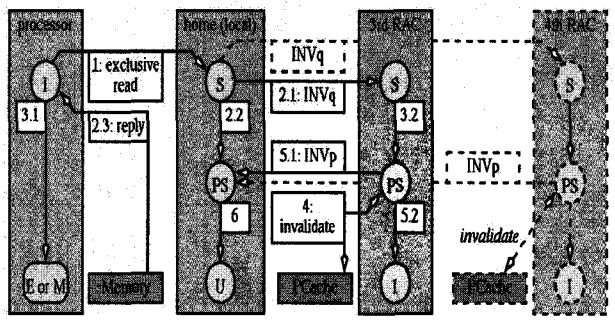


그림 6. 디렉토리 S에 대한 BusExRd
Fig. 6. BusExRd to Directory S.

작으로 받은 모든 데이터를 로컬 메모리로 보낸다.

BusExRd 트랜잭션이 요청되면, 디렉토리가 U인 경우에는 아무런 영향이 없고, S 상태인 경우에는 그림 6과 같이 INV 동작을 통해 데이터를 가진 모든 노드의 캐시를 무효화 시킨다. INV가 진행되는 동안에 캐시는 펜딩 상태로 있다가 INV 동작의 완료를 알리는 INVp 응답이 오면 U 상태로 바뀐다.

디렉토리의 상태가 M인 경우에는 다른 노드에 의해 해당 데이터가 변경된 경우이므로 그 데이터를 소유하고 있는 노드에게 ERD를 요청한다. 데이터를 소유하고 있는 노드는 가장 최신의 데이터를 홈 노드로 돌려 준다. 로컬의 DIR과 RAC는 U, I로 바뀐다. 만약 프로세서 캐시 라인의 크기가 원격지에서 읽어온 데이터의 크기보다 작을 때는 해당 라인을 제공한 후에 나머지 라인도 메모리로 써 넣는다.

BusInv 트랜잭션은 디렉토리가 U인 경우에는 아무런 영향이 없고, S 상태인 경우에는 그림 6과 같이INV 동작을 통해 데이터를 가진 모든 노드의 캐시를 무효화 시킨다. INV가 진행되는 동안에 캐시는 펜딩 상태로 있다가 INV 동작의 완료를 알리는 INVp 응답이 오면 U 상태로 바뀐다. DIR이 M인 경우에는 BusInv 트랜잭션이 발생할 수 없다.

프로세서가 BusWrb 트랜잭션을 요청하는 경우는 이

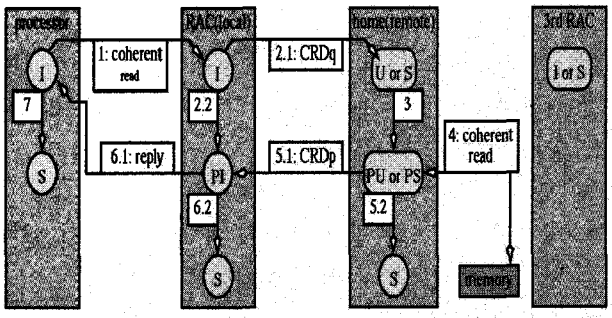


그림 7. RAC I, DIR S, DIR U의 BusCoRd
Fig. 7. BusCoRd to RAC I, DIR S, DIR U.

표 5. 프로세서 원격 트랜잭션으로 인한 상태 변화
Table 5. State Transitions by Remote Transaction.

프로세서 버스 접근	현재 RAC 상태	DIR에 의한 네트워크 동작	DIR의 현재 가능한 상태	최종상태		비고
				RAC	DIR	
BusCoRd	I	CRD	U/S/M	S	S	
	S	없음	S	S	S	
	M	없음	M	M	M	
	L	-	M	L	M	재시도
BusExRd	I	ERD	U/S/M	S	S	
	S	INV	S	S	S	
	M	없음	M	M	M	
	L	-	M	L	M	재시도
BusInv	I	ERD	U/S/M	S	S	발생않음
	S	INV	S	S	S	
	M	없음	M	M	M	
	L	-	M	L	M	재시도
BusWrb	I	ERD	U/S/M	S	S	발생 않음
	S	INV	S	S	S	발생 않음
	M	없음	M	M	M	
	L	-	M	L	M	재시도
BusWrt	I	ERD	U/S/M	M	M	
	S	INV	S	M	M	
	M	없음	M	M	M	
	L	-	M	L	M	재시도
BusLocRd	I	ERD	U/S/M	L	M	
	S	INV	S	L	M	
	M	없음	M	L	M	
	L	-	M	L	M	
BusLocWr	I,S	-	-	-	-	발생 않음
	M	-	-	-	-	발생 않음
	L	없음	M	M	M	발생 않음
BusUcRd	I	URD	U/S/M	I	U	
	S	-	S	S	S	발생 않음
	M	-	M	M	M	발생 않음
	L	-	M	L	M	재시도
BusUcWr	I	UWR	U/S/M	I	U	
	S	-	S	S	U	발생 않음
	M	-	M	M	M	발생 않음
	L	-	M	L	M	발생 않음

미 캐시 라인이 M 상태이므로, 디렉토리가 S, M 인 경우는 발생하지 않는다.

BusIoRd 와 BusIoWr은 캐시 관련 버스 동작과는 아무런 관계가 없으며, 이러한 트랜잭션은 BusUcRd, BusUcWr과 같이 취급해도 된다.

노드 내부 원격 영역의 트랜잭션

표 5는 원격 메모리 영역의 데이터에 대하여 로컬 프로세서가 접근을 요청한 경우에 발생되는 트랜잭션을 나타내었다. 이때 요청되는 원격 데이터에 대한 RAC에 가능한 캐시 상태를 나타내고, 또한 각 상태에 대하여 DIR에서 요청되는 네트워크의 트랜잭션과 그때의 DIR 상태를 보여 준다. 그리고 트랜잭션이 완료되는 경우에 DIR과 RAC의 상태가 어떻게 변하는 지를 나타내었다.

원격 영역에 대하여 BusCoRd 트랜잭션이 발생된 경우에는 디렉토리가 U, S인 경우에는 그림 7과 그 데이터의 소유하고 있는 노드에게 해당 데이터를 요구한다.

만약 디렉토리가 M 인 경우에는 그림 8과 같이 그 데이터를 소유하고 있는 노드로 CRD 트랜잭션을 전송한다. CRD를 요청 받은 노드는 최신의 데이터를 가지고 있는 노드에서 데이터를 가져와서 최초로 요청한 노드로 데이터를 전송한다. 홈 노드에서는 해당 데이터를 writeback 동작을 수행하여 메모리에 최신의 데이터가 유지되도록 한다. BusCoRd 트랜잭션에 관련된 모든 동작이 끝나면, RAC, DIR의 상태와 홈 노드의 캐시 상태는 모두 S로 변한다.

원격 영역이 RAC M인 경우의 BusExRd는 트랜잭션이 일어나지 않지만, S인 경우에는 그림 9와 같이 INV 트랜잭션이 발생된다. INV 트랜잭션이 끝날 때까지 캐

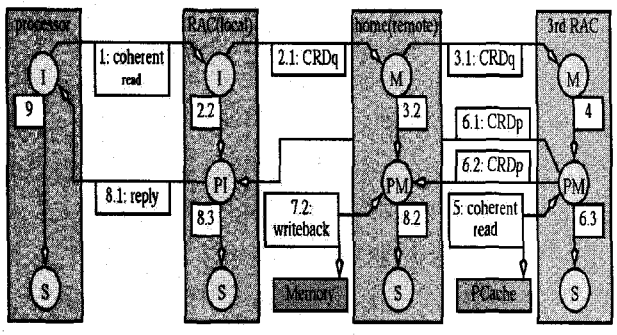


그림 8. RAC I, DIR M의 BusCoRd
Fig. 8. BusCoRd to RAC I, DIR M.

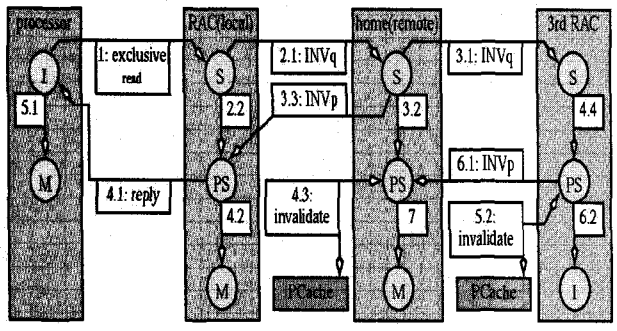


그림 9. RAC S의 BusExRd
Fig. 9. BusExRd to RAC S.

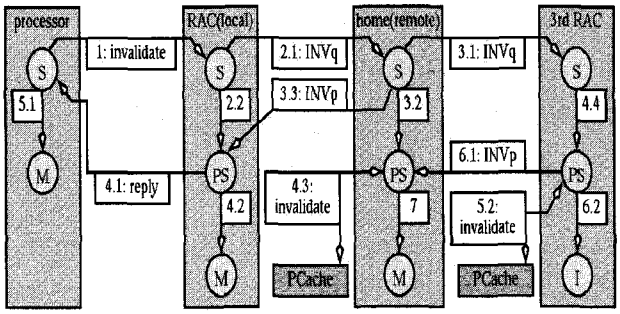


그림 10. RAC S의 BusInv
Fig. 10. BusInv to RAC S.

표 6. 로컬 영역에 대해 인바운드된 네트워크 트랜잭션에 의한 상태 변환
Table 6. State Transition by Inbounded Network Transaction to a Local Area.

네트워크 인바운드 트랜잭션	현재			DIR에 의한 버스 동작	DIR에 의한 네트워크 동작	최종상태			비고
	홈 DIR 상태	가능한 상태 RAC	가능한 상태 RAC'			DIR	RAC	RAC'	
CRD	U	I	I	BusCoRd	없음	S	S	I	
	S	I	S	BusCoRd	없음	S	S	S	
	M	I	M	BusWrb	CRD	S	S	S	
ERD	U	I	I	BusExRd	없음	M	M	I	
	S	I	S	BusExRd	INV	M	M	I	
	M	I	M	없음	ERD	M	M	I	
INV	U	I	I	-	없음				발생않음
	S	S	S/I	BusInv	INV	M	M	I	고스트공유
	M	I	M	-					발생않음
WRB	U	I	I	-					발생않음
	S	I/S	S	-					발생않음
	M	I	M	-					발생않음
	M	M	I	BusWrb	없음	U	I	I	
URD	U	I	I	BusUcRd	없음	U	I	I	
	S	I/S	S	-					발생않음
	M	I/M	M	-					발생않음
UWR	U	I	I	BusUcWr	없음	U	U	I	
	S	I/S	S	-					발생않음
	M	I/M	M	-					발생않음

시의 상태는 펜딩상태를 유지해야 한다.

RAC가 I인 경우는 ERD 트랜잭션이 일어난다. 이때 홈 노드의 디렉토리가 U이면 디렉토리는 프로세서 캐시에서 데이터를 읽어와서 요청한 노드로 전송한다. 이때 홈 노드의 디렉토리가 S이면 다른 노드의 캐시를 무효화 시키기 위해 홈 노드에 의해 INV 트랜잭션이 먼저 진행되어야 한다.

BusInv 트랜잭션이 RAC가 M인 경우에 발생되면 외부로 트랜잭션은 일어나지 않는다. 그러나 I인 경우는 그 데이터가 필요하여 BusInv가 진행된 경우이므로 ERD가 진행되고, S인 경우는 그림 10과 같이 INV 트랜잭션이 진행된다. 트랜잭션이 끝나면 RAC, 디렉토리 모두 S 상태가 된다. BusWrb 트랜잭션은 RAC가 I, S 경우는 발생하지 않고, M인 경우에는 Writeback 되는 데이터를 RAC에 반영하고 외부로 트랜잭션이 발생하지 않는다. BusWrt 트랜잭션은 BusErd와 유사하게 동작된다. RAC가 I인 경우에는 ERD, S 경우는 INV 트랜잭션이 외부로 진행된다.

표 7. 원격 영역에 대해 인바운드된 네트워크 트랜잭션에 의한 상태 변환

Table 7. State Transitions by Inbouned Network Transaction to a Remote Area.

네트워크 인바운드 트랜잭션	원래			RAC에 의한 버스 동작	RAC에 의한 네트워크 동작	최종상태				비고
	RAC 상태	가능한 상태				RAC	DIR	RAC'		
		DIR	RAC'							
CRD	I	S	I/S	-						발생않음
	S	M	I	-						발생않음
	M			BusCoRd	없음	S	S	S		
ERD	I	S	I/S	-						발생않음
	S	M	I	-						발생않음
	M			BusExRd	없음	I	M	M		
INV	I	S	S	없음	없음	I	M	M		
	S	S	I/S	BusInv	없음	I	M	M		고스트공유
	M	M	I	-	-					
URD	I				없음					발생않음
	S									발생않음
	M		I							발생않음
UWR	I				없음					발생않음
	S									발생않음
	M		I							발생않음

노드 외부에서 요청된 로컬 영역의 트랜잭션

표 6, 표 7은 네트워크로부터 요청되어진(인바운드) 트랜잭션에 대한 상태 변화를 나타낸 것이다. 표 6은 외부에서 요청된 트랜잭션이 로컬 영역의 메모리 접근인 경우의 상태 변화를 나타내고 있다. RAC에 의해 네트워크 트랜잭션이 프로세서 버스로 요청된다.

이때 DIR의 상태에 따라 진행되는 버스 트랜잭션은 상태에 따라 다르다. 표 9에서 RAC'는 원격에 있는 다른 RAC 들의 상태를 표시한다. 즉 해당 데이터에 대한 다른 프로세싱 노드가 가지고 있는 RAC 상태를 의미한다.

표 7은 인바운드된 트랜잭션이 원격 영역의 메모리 접근인 경우의 상태 변화를 나타내고 있다. 필요하다면 RAC에 의해 네트워크 트랜잭션이 프로세서 버스로 요청된다. 표 7에서 RAC의 상태가 L이 없는 이유는 이 상태에서 발생된 트랜잭션은 L이 해제될 때까지 트랜잭션이 발생한 노드에서 재시도를 진행하고 있기 때문이다. 즉 원격 영역에 대해 인바운드 되는 경우는 발생되지 않는다.

Ⅳ. 결 론

본 논문에서는 CC-NUMA 시스템에서 사용될 수 있는 프로세싱 노드의 구조를 제안하고, 노드간의 공유 메모리 환경에서 캐시 일관성을 유지하기 프로토콜을 제안하였다. 물리적으로 분산되어 있으나 논리적으로는 공유되는 메모리들 사이에서 원격 캐시를 제공하면서 캐시 일관성을 유지하는 일은 복잡하다. 분산된 프로세서들 간에 동시에 진행되는 트랜잭션은 다양한 상황을 발생시키고 이들 간에 전송되는 데이터의 일관성 유지는 결코 쉬운 일이 아니다. 따라서 본 논문은 분산된 프로세싱 노드들에서 발생될 수 있는 모든 트랜잭션의 상태와 캐시 상태를 일목요연하게 정리하여 각각의 경우에 따라 진행될 수 있는, 혹은 진행되어야 하는 동작을 일일이 나열하고 있다. 본 논문에서 정의된 프로토콜에 따라 CC-NUMA 시스템을 구현한다면 복잡하게 진행되는 분산된 데이터의 일관성 유지를 보다 쉽게 할 수 있다.

본 논문에서 제안된 프로토콜에 따라 그림 11과 같은 원격 캐시 일관성 유지 제어기를 이용한 CC-NUMA 시스템을 구현하였다. 이 제어기는 프로세서 버스 인터페이스[10], 디렉토리 제어기, 원격 캐시 제어기 및 링 인

그림 11. 원격 캐시 일관성 유지 제어기
Fig. 11. Remote Access Cache Controller.

터페이스로 구성되어 있다. 이 제어기는 프로세서 버스에 직접 연결하여 프로세서의 트랜잭션에 따라 분산 공유 데이터의 일관성을 유지한다.

향후 다양한 형태의 물리적 분산형 병렬 컴퓨터 시스템에서 본 프로토콜을 변형하여 적용할 수 있는 방법을 찾아보고, 더욱 성능을 개선하기 위해 원격 메모리 접근 시간을 단축할 수 있는 다양한 구조에 대한 연구가 필요하다.

참 고 문 헌

[1] David Culler, Jaswinder Pal Singh, and Annop Gupta, "Parallel Computer Architecture : A Hardware/Software Approach",Morgan Kaufmann Publishers, 1997.

[2] Anant Agarwal, Richard Simoni, Mark Horowitz, and John Hennessy, "An Evaluation of Directory Schemes for Cache Coherence", In Proceedings of the 15th Annual International Symposium on Computer Architecture, pp.280-289, 1988.

[3] Lynn Choi, and Anddrew A. Chien, "Integrating Network and Memory Hierachies in a Multicomputer Node Architecture", 8th International Parallel Processing Symposium, pp.10~17, April 26-29, 1994.

[4] SeongWoon Kim, Chulho Won, and SangMan Moh, "The Main Processing Unit for the High-Speed Midrange Computer TICOM-III", In Proceedings of the Joint Technical Conference 1995, pages 455-458, 1995.

[5] Paul Sweazey and Alan Jay Smith, "A class of compatible cache consistency protocols and their support by the IEEE Futurebus," In Proceeding of th 13th Annual International Symposium on Computer Architecture", pages 412~423, 1986.

[6] SeongWoon Kim and SangSeok Shin, "Modeling and Simulation of Memory Architecture on a Message Passing System", In Proceedings of the Joint Technical Conference 1996, pages 685-688, 1995.

[7] Hung-Chang Hsiao and Chung-Ta King, "Performance Evaluation of Cache Depot on CC-NUMA Multiprocessors", Parallel and Distributed Systems, Proceedings International Conference, pp519 - 526,Dec. 1998.

[8] Adi Golbert, Bob Farrell, Pete MacWilliams, Nabeel Sakran and Isic Silas, "A Second Level Multiprocessing Cache for the i486DX and i860 Processors", Compcon Spring '92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers., 24-28 Feb. 1992.

[9] James Archibald, "High Performance Cache Coherence Protocols For Shared-Bus Multiprocessors", Technical Report., CS Department, University of Washington, Nov 17, 1987.

[10] SeongWoon Kim, Ando Ki, and Bogwan Kim, "IA-32 Processor Interface Design for CC-NUMA system", In Proceedings Volume 3 of ITC-CSCC 2003, pages 1634~1637, July 9, 2003.

저 자 소 개

김 성 운(정회원)

1987년 부경대학교 전자공학과 학사 졸업

1998년 충남대학교 전자공학과 석사 졸업

2003년 충남대학교 전자공학과 박사 수료

1989년~현재 한국전자통신연구원 디지털홈연구단 서버플랫폼 연구팀장

<주관심분야 : 컴퓨터 구조, SoC 설계, 스토리지 시스템, TOE>

김 보 관(정회원)

1976년 서울대학교 전자공학과 학사 졸업

1978년 KAIST 전기전자공학과 석사 졸업

1989년 University of Wisconsin, Electrical and Computer Engineering 박사 졸업

1980년~1991년 금호공과대학 조교수

1991년~현재 충남대학교 전자공학과 교수

<주관심분야 : VLSI CAD, VLSI 시스템 설계 및 모델링, HW/SW Co-Design, 통신시스템용 VLSI 설계>