論文93-30B-2-4

HiPi-bus 구조의 다중 프로세서 시스템에서의 잠금장치

(A Lock Mechanism for HiPi-bus Based Multiprocessor Systems)

尹 龍 鎬*, 林 寅 七*,

(Yong Ho Yoon and In Chil Lim)

要 約

잠금장치는 다중 프로세서 시스템의 동기화를 위해 필수적이다. 잠금장치는 잠금 경합이 적을 때 잠금 동작에 소요되는 시간을 줄이는 방법이 필요하며, 잠금 경합이 빈번히 발생할 경우도 고려되어야 한다. 그러나 기존의 메모리에서 잠금을 제어하는 방식은 잠금 동작시 버스 트래픽과 메모리 사용율이 증가한다. 본 논문에서는 HiPi-bus (Highly Pipelined bus)를 사용하는 다중 프로세서 시스템에서 잠금 경합이 적을때 잠금 동작에 필요한 시간을 줄이기 위해 잠금 데이타를 캐쉬에 저장하고 이를 효율적으로 관리하는 방식을 제안하고, 잠금 경합이 빈번할 때 잠금 동작에 필요한 시간을 줄이기 위해 캐쉬에서 캐쉬로의 데이타 전송 방식을 채택한 HiPi-CLOCK(HiPi-bus Cache LOCK mechanism)을 설계하였다. 그리고 시뮬레이터를 제작하고 인공적인 트레이스 상에서 메모리에서 잠금을 제어하는 방식과 본 논문에서 제안한 HiPi-CLOCK의 성능을 하나의 RMW(Read-Modify-Write) 동작 수행시간으로 비교하였다. 그 결과 잠금 경합이 적을 경우에는 언제나 두 배 이상의 성능을 보였고, 잠금 경합이 많을 경우에는 잠금 데이타의 공유수가 증가할수록 성능이 우수하였다.

Abstract

Lock mechanism is essential for synchronization on the multiprocessor systems. Lock mechanism needs to reduce the time for lock operation in low lock contention. Lock mechanism must consider the case of the high lock contention. The conventional lock control scheme in memory results in the increase of bus traffic and memory utilization in lock operation. This paper suggests a lock scheme which stores the lock data in cache and manages it efficiently to reduce the time spent in lock operation when the lock contention is low on a multiprocessor system built on HiPi-bus (Highly Pipelined bus). This paper also presents the design of the HiPi-CLOCK (Highly Pipelined bus Cache LOCK mechanism) which transfers the data from one cache to another when the lock contention is high. The designed simulator compares the conventional lock scheme which controls the lock in memory with the suggested HiPi-CLOCK scheme in terms of the RMW(Read-Modify-Write) operation time using simulated trace. It is shown that the suggested lock control scheme performance is over twice than that of the conventional method in low lock contention. When the lock contention is high, the performance of the suggested scheme increases as the number of the shared lock data increases.

* 正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
接受日字 1993年 1月 15日

I. 서론

잠금 장치(Lock mechanism)는 프로세서들에 의

해 공유되는 공유자원을 참조하는 시간에 제한을 가하여 공유자원에 대한 직렬성을 보장함으로써 시스템 내의 정보 흐름과 기능성을 보장한다.[1, 2] 다중 프로세서 시스템에서의 자원 공유에 기인한 동기화 문제는 단일 프로세서 시스템보다 더욱 복잡하여 캐쉬 일관성 유지 프로토콜과 버스 동작 프로토콜에 밀접하게 관련되어 있다.

일반적으로 하드웨어 수준에서 동기화를 지원하기 위해 연속 동작(indivisible operation)을 사용한다. 이러한 연속 동작을 구현하는 방법은 버스 홀딩(bus holding), 메모리 잠금(memory lock), memory performs the operation으로 분류된다.[1]

단일 버스 시스템에서 프로세서가 연속 동작을 수행할 때 버스를 혼자 점유하는 방식을 버스 홀딩 방식이라 한다. 메모리 잠금 방법은 연속 동작을 인터록 읽기와 인터록 쓰기(interlock write)로 분리하고 인터록 읽기에 대해 선택된 메모리 모듈은 자신을 잠금 버퍼하여 인터록 쓰기가 올때까지 자신에 대한 다른 참조 요구를 수행하지 않는다. 그러므로 인터록 읽기와 인터록 쓰기 사이의 시간 동안에 버스를 점유하지 않으므로 버스를 다른 일에 사용할 수 있다.

Memory performs the operation 방법은 RMW(Read-Modify-Write) 사이클 동안 필요한 모든 일을 메모리 모듈에서 수행하므로 RMW 사이클이 진행되는 동안에는 다른 요구를 처리하지 않는다.

한편 공유메모리 방식의 다중 프로세서 시스템에서 메모리의 접근 시간이 데이타의 전송 속도에 영향을 주지 않도록 하기 위해 펜디드 버스 프로토콜을 채택한다. 이러한 버스 프로토콜에서 연속 동작을 구현하는 방법으로는 잠금 버퍼를 이용한 메모리 잠금 방법과 에뮬레이션 버스 홀딩(emulation bus holding) 방법 그리고 잠금 버퍼 와 에뮬레이션 버스 홀딩을 혼용한 방법이 있다.[7] HiPi-bus를 이용한 메모리 잠금에서 메모리 모듈 단위의 잠금이 아니라 메모리 모듈 내의 해당 번지에 대한 잠금을 하기위해 잠금 번지를 저장할 수 있는 잠금 버퍼를 둔다.

에뮬레이션 버스 홀딩 방법은 어떤 프로세서가 인터록 읽기를 시도하여 완료한 시점부터 같은 프로세서가 인터록 쓰기를 수행할 때까지 다른 프로세서들이 시도하는 인터록 읽기와 인터록 읽기를 수행한 번지(lockvariable)에 대한 참조 요청을 처리하지 않는 방법이다. 한편 잠금 장치를 구현시 고려해야 할 문제는 캐쉬 일관성 유지 프로토콜이다. 캐쉬 일관성 유지 프로토콜(Cache Cocherence Protecl)은 하드웨어로 구현하기에 매우 복잡하고 어렵다. 따라서 가능하면 캐쉬 일관성 유지 프로토콜을 바꾸지 않고 잠금에 관련된 필요한 기능을 지원하도록 하는 것이 중요하다.

잠금 장치를 설계할 때 고려해야 할 사항으로 잠금 경합이 없을 경우 잠금 동작의 부담을 감소시켜야 한다는 점이다. 메모리에 잠금 버퍼를 두고 잠금 번지를 저장하여 메모리가 잠금을 제어하는 방식인 메모리 잠금 방식의 경우 하나의 TAS(Test-And-Set) 사이클에 두개의 버스 요구(인터록 읽기/인터록 쓰기)를 하여야 하므로 버스의 트래픽이 증가한다. 또한 프로세서의 처리 속도가 점점 빨라지고 메인 메모리의 큐(queue)의 크기가 증가함에 따라 메모리의 이용율도 증가한다. 따라서 메모리의 이용율 증가에 따라 메모리의 응답 거부 횟수가 증가하게 되고 이에 따라 프로세서의 잠금 동작 요구가 메모리 문제로 버스의 상태 버스에서 거부될 수 있다. 따라서 버스 요구 재시도가 발생하므로 버스의 트래픽은 증가하게 된다. TICOM OS(SVR3.1)에서 가장 많이 사용하는 스핀 잠금(spin lock)[8] 의 알고리듬은 spin on read[9] 이며 이는 normal read로 잠금 비트를 참조한 후 상태가 unlock이면 TAS 동작을 수행한다. 이와 같은 스핀 잠금을 수행하면 잠금 경합이 발생할 확률이 증가하므로 잠금 경합이 발생하였을때 잠금 동작의 부담(overhead)을 줄이는 방법을 모색해야 한다. 메모리 잠금 방식에서 발생되는 부가적인 버스 트래픽 문제를 해결하기 위해 잠금 주소를 캐쉬에 저장하는 것이 유리하다.[3] 잠금 경합이 없을 경우 유리한 무효화(invalidation) 방식의 잠금 장치는 캐쉬 상태를 이용하여 잠금 부담을 감소시키는 점을 가지고 있으나 잠금 경합이 자주 발생한다.[10] 이러한 문제를 해결하기 위해 방송(broadcast) 잠금 방식이 있는데 잠금 경합이 발생하지 않아도 항상 버스에 방송을 요구하므로 잠금 동작의 부담이 높다. 따라서 잠금 경합이 없을 경우의 잠금의 부담을 줄이기 위한 무효화 잠금 방식을 채택하고 잠금 경합이 발생시 메모리의 상태와 무관하게 캐쉬에서 캐쉬로의 데이타 전송 방식을 이용하여 잠금 주소로 접근하게 잠금 장치를 설계하면 보다 효율적이다.

잠금 데이타를 캐쉬에 저장하고 제어하는 방식으로는 캐쉬 메모리에 잠금 상태를 저장하고 다른 캐쉬가 해당 잠금 주소를 요구할 때 잠금 주소의 데이타 및 잠금 상태를 전송하는 방식이 있다.[4] 그러나 HiPi-bus(Highly Pipelined bus)를 이용한 공유 메모리 방식의 다중 프로세서 시스템에서 위의 방식으로 설계하려면 버스의 변형을 피할 수 없고 2차 캐쉬가 없는 프로세서 보드에서는 잠금 상태를 저장하기 위한 설계 변경이 용이하지 않다. 또한 잠금 디렉토리

(lock directory)를 이용한 방법이 있는데 이는 본 논문에서의 대상 시스템 구조와는 다른 슈퍼 컴퓨터에서 구현하였다. [10]

본 논문에서는 HiPi-bus를 사용하는 공유 메모리 방식의 다중 프로세서 시스템에서 캐쉬 메모리에 잠금 큐를 두고 캐쉬에서 캐쉬로의 데이타 전송을 이용한 잠금 방법인 HiPi-CLOCK(Highly Pipelined bus Cache LOCK)을 새로이 제안하고, 잠금 버퍼를 메모리에 둔 메모리 잠금 방법과 성능을 비교한다. 2장에서는 잠금 방법으로 메모리 잠금 방법을 구현한 TICOM의 시스템 구성, 캐쉬 일관성 유지 프로토콜 및 HiPi-bus 프로토콜의 동작을 설명하고, 구현된 메모리 잠금 방법의 특징을 기술한다. 3장에서는 새로 제안한 HiPi-CLOCK에 대해 언급하고, 4장에서는 메모리 잠금과 HiPi-CLOCK에 대한 시뮬레이션 및 성능을 비교하며, 5장에서 전체적인 요약을 한다.

Ⅱ. HiPi-bus 구조의 다중 프로세서 시스템

1. 시스템 구조

HiPi-bus를 바탕으로 한 공유메모리 방식의 다중 프로세서 시스템의 구성은 그림 1과 같고 TICOM과 그 구조가 비슷하다.

모든 중앙처리장치는 각각 캐쉬 메모리를 갖고 있으며 데이타 전송은 팬디드 프로토콜(Pended Protocol)을 사용한다. 메모리는 버스 인터페이스와 메모리 어레이(array)로 구성되어 있고 메모리 큐는 2개이다. 모든 프로세서는 단일 버스이며 동기 버스인 HiPi-bus를 통해 연결되어 있다. 같은 메모리를 연속적으로 요구할 때의 경합 발생 문제를 위해 가장 먼저 메모리에 도착한 요구를 처리하고 나머지는 처리 거부를 하여 거부 횟수가 많을 수록 버스의 트래픽은 증가한다.

2. 캐쉬 일관성 유지 프로토콜

공유 버스의 트래픽을 감소시키기 위해서 write-back 방식을 채택하고 있으며 캐쉬 일관성 유지 프로토콜은 변형된 Illinois 방식 [6] 으로 그 특징을 요약하면 다음과 같다.

캐쉬는 데이타를 일정한 크기의 블록으로 나누어 저장하고 그것을 관리하기 위하여 블록 단위로 상태와 태그를 둔다. 그림 2에 나타난 바와 같이 Illinois 프로토콜에서는 캐쉬의 블록 상태를 Invalid, Valid-exclusive, Shared, Dirty로 나누는데 요약하면 표 1과 같다.

그림 2에서와 같이 Illinois 일관성 유지 프로토콜은 다른 캐쉬와 데이타를 공유하면서 동시에 메모리와 내용이 다른 상태를 허용하지 않고 이를 위해 Invalid 상태를 이용한다. 즉, S 상태에 있는 캐쉬가 쓰기 동작을 할때 캐쉬 적중(write hit)이 발생하면 그림 2에서와 같이 버스 동작을 통하여 다른 캐쉬를 Invalid 시키고 자신은 D 상태로 된다. 따라서 그림 2와 같은 일관성 유지 프로토콜에서 단순히 캐쉬에 잠금 변수를 캐쉬에 저장한다면 잠금 데이타의 공유가 증가할수록 S 상태가 많아지게 되고, RMW 사이클 중 쓰기 동작시 다른 캐쉬의 해당 블록을 Invalid로 천이시킨 후 쓰기 동작을 수행해야 하며 해당 캐

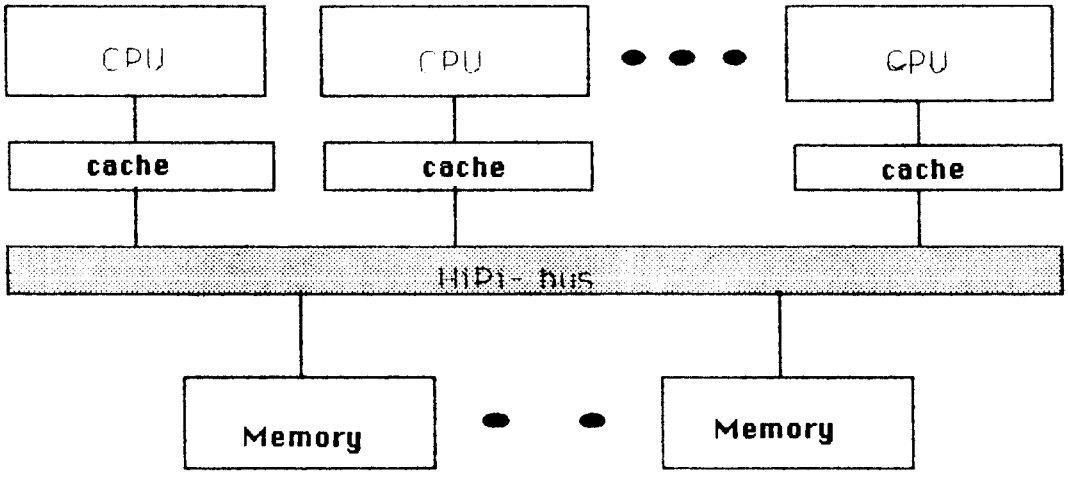

그림 1. 공유 메모리 방식의 다중 프로세서의 구조
Fig. 1. The structure of shared memory multiprocessor system.

쉬 블록의 상태를 D 로 천이해야 하는 일관성 유지 부담이 크게 된다.

. 메모리 크기 : 64 킬로 바이트
. 블록크기 : 8 바이트
. 세트 결합도 : Direct mapping
. 메인 메모리 갱신 : write-back 방법
. 캐쉬 동일성 유지 프로토콜 : 변형 Illinois 프로토콜
. 캐쉬 디렉토리 관리 : 분산 디렉토리, 버스 스누우핑
. 캐쉬 메모리 액세스 시간 : 캐쉬 히트이면 프로세서의 대기 상태없이 처리

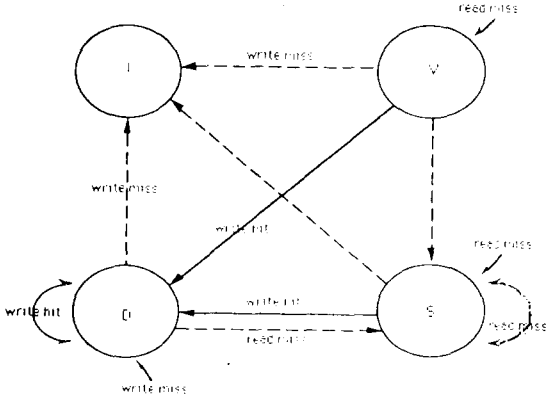

그림 2. 변형된 일리노이즈 일관성 유지 프로토콜
Fig 2. Modified Illinois coherence protocol.

표 1. 변형된 Illinois 일관성 유지 프로토콜에서의 캐쉬 상태
Table 1. Cache state in modified Illinois coherence protocol.

상 태	설 명
I(Invalid)	데이타 블록이 유효하지 않음
V(Valid-Exclusive)	다른 캐쉬는 존재하지 않은 유일한 블록임을 표시 즉 블록이 공유되지 않음을 나타냄
S(Shared)	한개 이상의 캐쉬들에 의하여 공유됨을 표시. 블록의 읽기만을 허가받은 상태이므로 쓰기를 위하여 허가를 필요로함.
D(Dirty)	캐쉬에 존재하는 유일한 블록이고 블록의 데이타를 허가없이 임의로 읽고 쓰기 할 수 있음을 표시

3) 버스 데이타 전송 프로토콜

HiPi-bus [5] 는 데이타 전송을 여러개의 주기로 나누어 수행한다. 각각의 주기는 하나의 버스 클럭 동안에 이루어지며 버스의 동작은 파이프라인(pipeline)이 되므로 여러개의 프로세서가 각각 독립적인 일을 동시에 할 수 있으므로 다중 프로세서 시스템에 효율적이다. 표 2는 버스의 전송 형태이며 그림 3은 HiPi-bus의 읽기와 쓰기 동작을 그림으로 나타낸 것이다.

표 2. HiPi-bus의 전송 형태
Table 2. Transfer type for HiPi-bus.

전송 형태	기 능	특 징
RFR	read	cache coherence
RFW	write	cache coherence
INV	invalidation	cache coherence
WRB	write back	cache coherence
NRD	read	normal
NWR	write	normal
LCR	interlock read	RMC cycle
LCW	interlock write	RMC cycle

하나의 RMW는 인터록 읽기와 인터록 쓰기로 구성되는데 HiPi-bus에서의 인터록 읽기는 그림 3(a), 인터록 쓰기는 그림 3(b) 와 같이 동작한다. 특히 인터록 읽기의 경우에 잠금한 프로세서나 메모리가 해당 블록의 잠금을 요구하는 인터록 읽기 요구를 버스상에서 거부할 수 있으며 이는 그림 3(a)에서 주기(phase) 2를 이용한다. 즉, 하나의 프로세서가 특정 블록을 잠금하였을 때 이 잠금이 해제되지 않은 상태에서 다른 프로세서가 그 블록 잠금을 요구하기 위해 버스에 인터록 읽기 요구를 하면 주기 2의 응답 사이클에서 잠금 비지(lock-busy)를 응답하여 잠금을 거부하며 잠금을 거부당한 인터록 읽기 요구는 다시 어드레스 버스 중재 사이클을 거쳐야 한다. 따라서 주기 2의 응답 사이클에 잠금 비지가 많이 사용될수록 버스의 이용율은 높아지고 하나의 RMW 동작시 소비되는 시간이 증가하므로 RMW의 성능은 저하된다.

4. TICOM에서의 잠금 장치

TICOM에서는 메모리에 잠금 버퍼를 두었고 이를

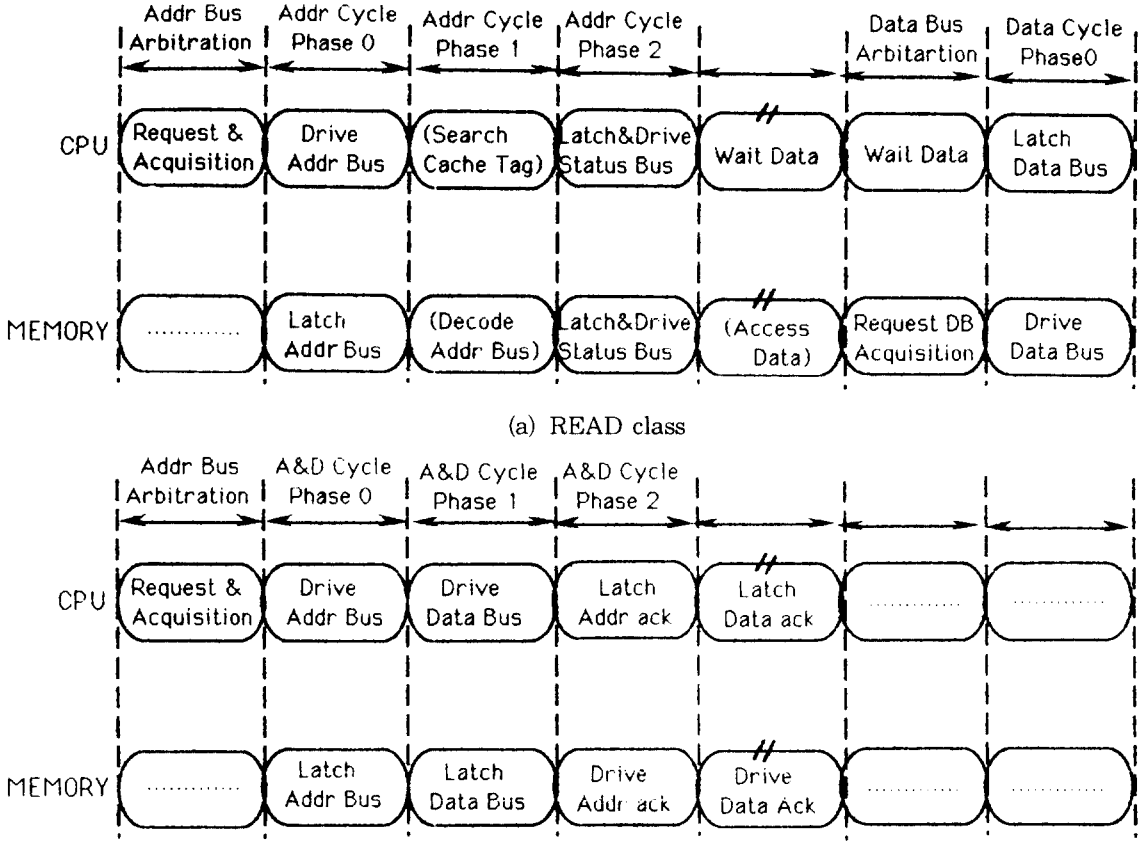

그림 3. HiPi-bus의 read/write class operation
Fig. 3. Read/write class operation in HiPi-bus.

메모리가 제어하는 메모리 잠금 방법과 캐쉬 일관성 유지 프로토콜을 사용하였다. TAS 명령어를 처리하기 위해 인터록 읽기(버스 전송 종류는 LCR)와 인터록 쓰기(버스 전송 종류는 LCW)를 버스에 요구한다. LCR을 버스가 서비스할 때 버스 주기 2에서 상태 버스에 이상이 없을 때 메모리는 잠금을 요구한 주소를 메모리에 있는 잠금 버퍼에 저장하고 해당 주소에 대한 LCW가 전송될 때까지 다른 프로세서의 잠금 동작을 재시도하게 하여 이루어진다. 한편 각 프로세서의 스누프는 버스 주기 2에서 다른 프로세서의 잠금 요구가 성공인지 아닌지의 상태를 알 수 있는데 만약 잠금 요구가 성공이면 자신의 캐쉬 메모리에 해당 블록의 상태를 Invalid로 한다.

이러한 방법의 잠금 장치는 잠금 경합이 없을 경우에도 두번의 버스 사용을 요구하고 인터록 쓰기의 서비스가 메모리에서 수행되어야만 잠금이 풀리므로 메모리의 수행 시간 때문에 잠금 동작의 성능에 영향을 받는다. 또한 잠금 경합이 많을 경우 메모리가 다른 일을 처리할 경우 메모리의 잠금 서비스를 받지 못하고 잠금 동작을 위한 버스 사용 재시도가 발생하므로 버스의 트래픽은 증가한다.

Ⅲ. HiPi-CLOCK

1. 하드웨어

캐쉬의 태그에 잠금 동작을 지원하기 위한 잠금 비트를 둔다. 또한 잠금 주소와 상태를 저장하는 잠금 버퍼를 둔다. 여기서 잠금 동작에 사용되는 잠금 변수는 여러 프로세서가 공유할 수 없는 것이므로 캐쉬 메모리에서는 Shared 상태가 되어서는 안된다. 잠금 버퍼의 상태는 다음과 같다.

. 잠금(Lock) : 인터록 읽기가 올바르게 수행된 상태

. 해제(Unlock) : 인터록 쓰기가 올바르게 수행된 상태

잠금 변수를 캐쉬에 저장하면 잠금 경합 발생시 write-back이 빈번히 발생하게 된다. 이러한 write-back을 효율적으로 수행하기 위해서는 캐쉬에서 캐쉬로의 데이타 전송이 필요한데, HiPi-bus는 그림 3과 같이 캐쉬에서 캐쉬로의 데이타 전송이 불가능하다. 따라서 HiPi-bus에서 캐쉬에서 캐쉬로의 데이타 전송이 가능하게 하기 위해서 다음과 같은 기능을 추가해야 한다.

HiPi-bus의 캐쉬에서 캐쉬로의 데이타 전송 방법은 그림 3에서 버스 주기 2 사이클에서 write-back하고자 하는 스누프가 ITV(InTerVention)를 응답(acknowledge)으로 구동하고 데이타를 그림 3의 read class의 MEM과 같이 해당 캐쉬에서 전송한다. 캐쉬에서 캐쉬로의 데이타 전송에 대한 버스의 동작은 그림 4와 같다.

2. 잠금 동작 제어

이 발생하여 읽기를 캐쉬에서 수행하며 잠금 버퍼에 해당 주소를 저장한 후 잠금 버퍼의 상태를 잠금으로 한다. 잠금 버퍼의 상태를 해제(Unlock) 하는 경우는 인터록 읽기가 선행되어 해당 번지는 캐쉬에 있으므로 캐쉬에 쓰기를 시도할 때 잠금 버퍼에서 해당 번지의 상태를 해제 상태로 하여 잠금을 해제하며 해당 캐쉬 블록의 태그에 잠금 비트를 "1"로 한다. 한편 잠겨진 주소에 대한 다른 프로세서의 잠금 동작을 버스에 요구될 때 스누프는 잠금 버퍼를 참조하여 해당 주소가 잠금 상태가 되어 있으면 메모리의 접근을 막는 신호인 잠금 비지를 보내며 이 신호를 받은 프로세서는 버스에 다시 잠금 전송을 재시도 한다. 또한 인터록 쓰기로 인하여 캐쉬 상태가 Dirty인 블록의 다른 프로세서가 참조하였을 때 스누프는 캐쉬에서 캐쉬로의 데이타 전송을 구동하고 해당 블록의 잠금 비트가 "1" 상태이면 캐쉬의 해당 블록 상태를 Invalid로 만들고 잠금 비트를 "0"로 한다. 따라서 기존의 방식인 메모리 잠금 방법이 잠금 경합이 발생

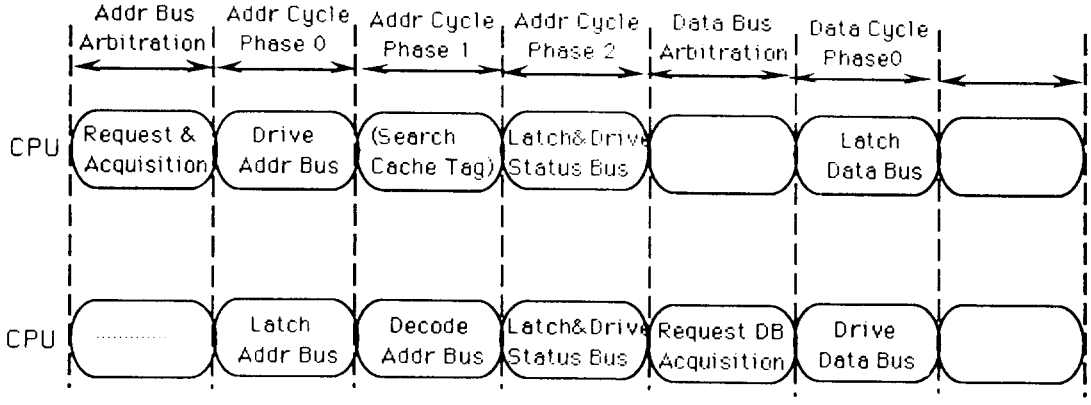

그림 4. HiPi-bus에서 cache-to-cache 동작 순서
Fig. 4. Cache-to-cache operation sequence in HiPi-bus.

다중 프로세서 시스템에서 잠금 동작은 인터록 읽기(IR)와 인터록 쓰기(IW)로 구분할 수 있다. 하나의 atomic RMW 동작을 지원하려면 IR과 IW 사이에 다른 프로세서의 간섭을 배제하여야 한다.

다음은 잠금 버퍼에 잠금 주소를 저장하고 상태를 잠금(Lock)으로 하는 경우의 동작 설명이다.

. 캐쉬 실패(miss) : 인터록 읽기 번지가 캐쉬 실패가 발생하여 버스에 LCR을 한다. 버스 주기인 phase 2에서 응답을 받고 정상이면 잠금 버퍼에 해당 주소를 저장한 후 잠금 버퍼의 상태를 잠금으로 한다.

. 캐쉬 성공(hit) : 인터록 읽기 번지가 캐쉬 성공

하지 않아도 두번의 버스를 사용하는 대비해 본 논문에서 제안한 잠금 장치는 잠금 번지의 블록이 캐쉬 미스가 발생하여도 한번의 버스를 사용하게 된다. 또한 잠금 경합의 발생시 발생하는 write-back도 캐쉬에서 캐쉬로의 데이타 전송을 이용하여 전송하므로 응답시간이 단축된다. 앞에서 설명한 잠금 동작을 그림으로 나타내면 그림 5과 같다.

Ⅳ. 시뮬레이션 결과 및 분석

시뮬레이션은 메모리 잠금 방식의 잠금 장치와 제안된 잠금 장치의 성능을 비교하기 위해 TICOM의

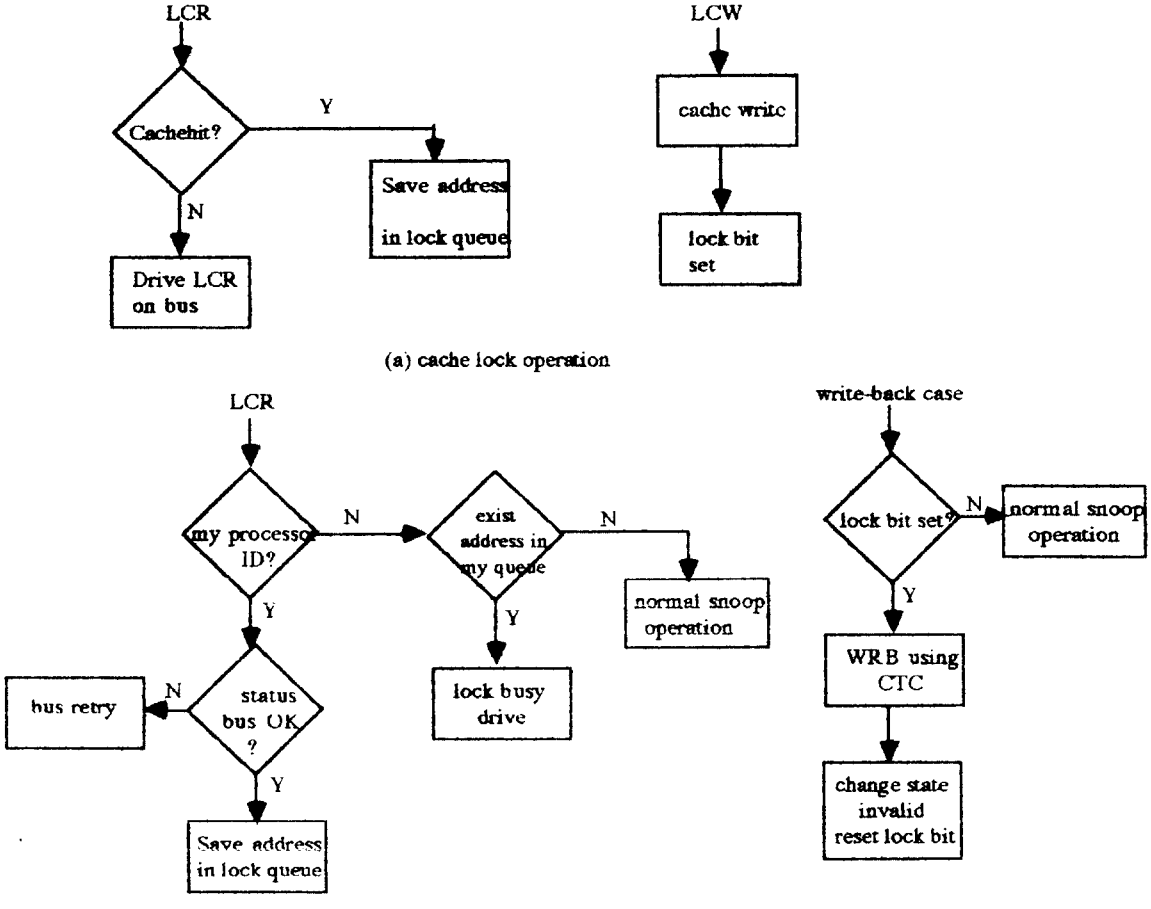

그림 5. 잠금 장치의 동작 흐름도
Fig. 5. Flow diagram of lock operation.

하드웨어를 반영한 소프트웨어 시뮬레이터를 이용하여 수행한다. 트레이스(trace) 방식의 시뮬레이터에서 트레이스는 실제 시스템에서 얻는 것이 시뮬레이션 결과의 정확도를 높이는데 가장 좋다. 다중 프로세서 시스템에서 프로세서의 갯수가 증가할수록 시스템에서 소비되는 시간 중 순수한 사용자 영역의 프로그램을 수행하는 시간보다 시스템 영역의 프로그램이 수행되는 시간의 비율이 높으므로 실제 시스템에서 트레이스를 구하려면 커널(kernel)의 변형은 불가피하게 되고 특히 RMW에 관한 트레이스를 구하려면 기계어 해석기(machine code interpreter)가 있어야 함으로 트레이스를 구하는 비용이 매우 비싸다. 따라서 본 논문에서는 트레이스 생성기를 두어 인공적으로 트레이스를 구한다. 시뮬레이터는 트레이스 생성기, 트레이스 화일, 캐쉬 시뮬레이터로 구성되며 그림 6와 같다.

트레이스 생성기는 잠금 빈도율, 프로세서 갯수, 잠금 공유율 등을 변수로 하여 메모리의 주소와 명령형태(읽기, 쓰기, RMW등)를 트레이스 화일에 기록한다. 캐쉬 시뮬레이터 및 버스 시뮬레이터는 앞에서 언급한 하드웨어 동작 특성을 반영하여 동작한다. 제작된 시뮬레이터로 부터 메모리에 잠금 버퍼를 두고 잠금의 제어를 메모리가 하는 방식과 HiPi-CLOCK 방식의 성능을 비교 분석한다. 성능을 분석하기 위해 잠금의 빈도율, 잠금 공유율 및 잠금 경합에 따른 하나의 RMW 평균 수행 시간과 HiPi-bus상에서 잠금에 의한 버스의 부하를 관측하기 위해서 하나의 RMW 수행에 대한 평균 잠금 비지 횟수를 관측한다.

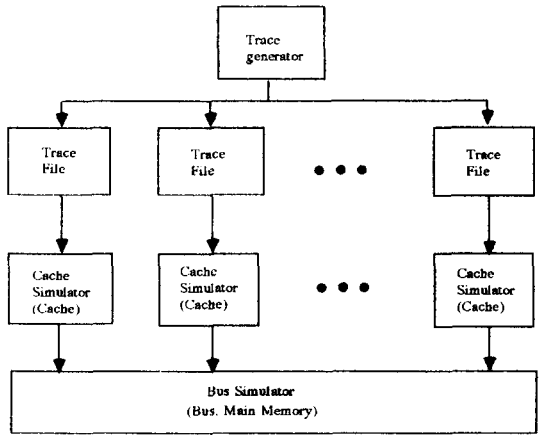

그림 6. 시뮬레이터의 구성도
Fig. 6. The structure of the simulator.

1. 시뮬레이션 결과

1) 잠금 빈도율

잠금이 발생하는 확률을 변수로 하여 하나의 RMW의 평균 수행 시간을 관측한다. 하나의 RMW 평균 수행 시간(MR)은 아래와 같다.

$$MR = \frac{\text{총 RMW 사이클 시간}}{\text{RMW 명령어 수 x 프로세서 수 x 버스 사이클 시간}}$$

이 MR은 버스 클럭 속도의 단위로 표시하여 MR이 1인 경우 버스 클럭 속도인 80 nsec가 된다. 아래의 결과는 프로세서가 10개인 경우이다.

그림 7에서 FL(Frequency of Lock) 축상에 있는 100 트레이스는 100개의 트레이스당 하나의 RMW가 발생하는 경우이다.

아래의 결과는 잠금 빈도율에 따른 하나의 RMW 동작이 HiPi-bus에서 주기 2에서 받은 평균 잠금 비지 횟수인 NLB(Number of Lock Busy) 결과이다. 100 개의 트레이스당 하나의 RMW가 발생시 MR은 거의 2.5 배 차이가 나며 이때 버스의 응답주기에 사용된 잠금 비지는 2배이상 차이가 난다. 또한, 잠금 비지 횟수가 0.5 이하인 500 트레이스 부터는 MR이 약 2배이상 차이가 난다. 즉, 잠금 비지 횟수가 0.5 이하인 경우는 잠금 경합이 거의 없을 경우이므로 잠금 경합이 없을 때 하나의 RMW에 소비되는 시간인 MR이 약 2배 차이가 난다.

2) 잠금 데이타 공유율

잠금 경합율을 1로 하였을때 하나의 RMW의 평균 수행 시간을 관측한다. 잠금 경합을 1로하기 위해 하나의 번지에 잠금 공유를 하는 프로세서가 RMW를 수행한다

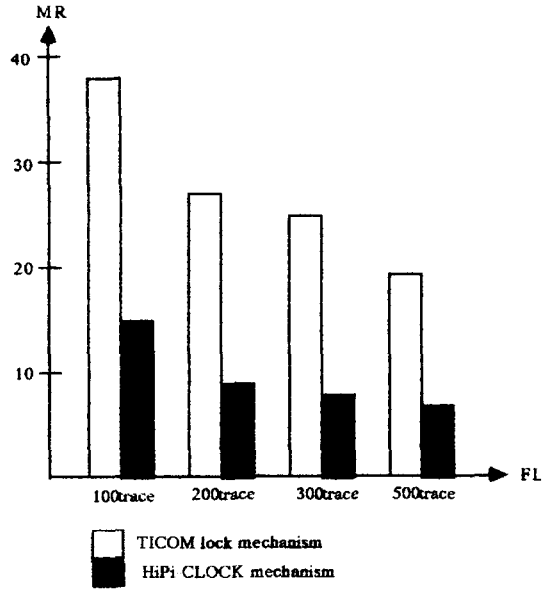

그림 7. 잠금 빈도율에 따른 RMW 평균 수행 시간
Fig. 7. The RMW execution time according to the frequency rate.

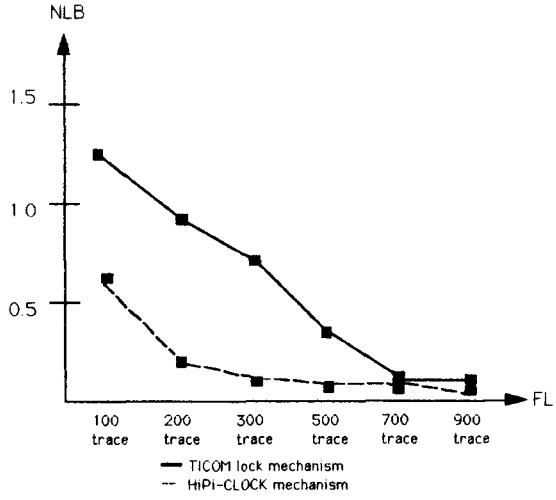

그림 8. 잠금 빈도율에 따른 RMW 평균 잠금비지 횟수
Fig. 8. The average number of RMW lock busy according to the lock frequency rate.

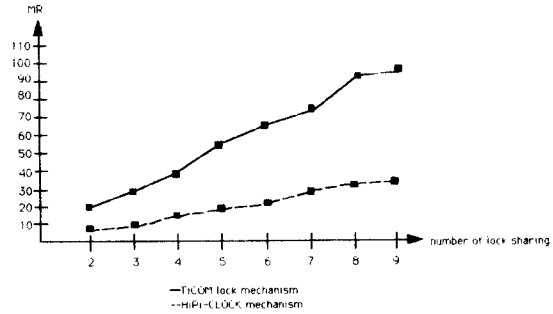

그림 9. 잠금 데이타 공유율에 따른 RMW 평균 수행 시간

Fig. 9. The average RMW execution time according to the lock data sharing rate.

아래의 결과는 잠금 데이타 공유율에 따른 하나의 RMW 동작이 HiPi-bus에서 주기 2에서 받은 평균 잠금 비지 횟수 결과이다.

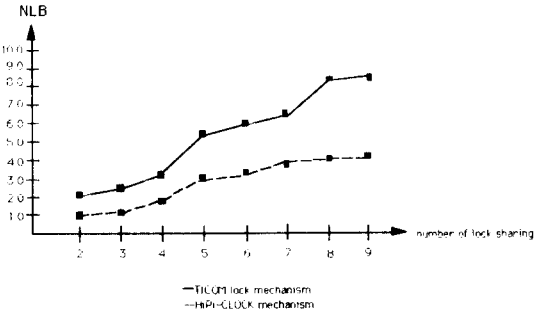

그림 10. 잠금 데이타 공유율에 따른 RMW 평균 잠금 비지 횟수

Fig. 10. The average number of RMW lock busy according to the lock data sharing rate.

잠금 데이타 공유율이 증가할수록 잠금 비지 횟수는 증가하고 MR 차이도 증가한다. 잠금 데이타 공유율이 2인 경우 MR이 약 2배인데 8인 경우 약 3배로 차이가 증가한다. 그런데 잠금 데이타 공유율이 증가할수록 MR의 차이보다 훨씬 큰 폭으로 잠금 비지 횟수의 차이가 날 것으로 예상했으나 그렇지 못했다. 그 이유는 잠금 데이타 공유율이 증가할수록 잠금 비지가 많아지고 버스의 이용율이 증가하므로 어드레스 버스 중재 사이클에 많은 버스 요구가 참가하여 중재 부담이 높아지기 때문이라 생각된다. 잠금 경합 발생시 잠금 데이타의 공유율이 높을수록 HiPi-CLOCK의 우수함을 알 수 있다.

3) 결과 분석

그림 7에서 보는 바와 같이 100개의 트레이스당 하나의 RMW 동작을 수행시킨 결과 하나의 RMW 평균 수행 시간이 약 2.5배가 차이가 난다, 그리고 잠금 빈도율과는 무관하게 언제나 2배이상의 수행시간이 차이가 난다. 이는 근본적으로 잠금 제어를 메모리에서 하는 방식은 하나의 RMW 동작을 위해서 2개의 버스 요구를 하기 때문에 최소한 하나의 버스 요구를 하는 HiPi-CLOCK보다 버스 트래픽을 증가시키기 때문이다. 그림 8에서 두개의 잠금 장치의 RMW 동작이 받는 잠금 비지 횟수가 작으므로 작업부하가 잠금 경합이 적을 경우임을 의미한다. HiPi-CLOCK은 잠금 빈도수가 100개의 트레이스일 경우에서 급격히 감소하고 수행시간도 감소함을 알 수 있다. 또한 TICOM의 잠금 장치는 비교적 선형적으로 감소하고 잠금 빈도수가 500개의 트레이스일 경우부터 0에 근접한다. 그러나 RMW 수행시간은 크게 감소하지 않으므로 근본적인 잠금 오버헤드가 HiPi-CLOCK보다 많음을 말한다. 그림 9에서 보는바와 같이 잠금 경합이 최대일 경우 잠금 데이타의 공유가 증가함에 따라 두개의 장치의 RMW 수행시간도 증가함을 알 수 있다. TICOM 잠금 장치는 잠금 데이타 공유수가 5에서 약 50%의 RMW 응답시간이 증가하였고 이때 잠금 비지 횟수는 그림 10에서 처럼 2배가 증가하였다. HiPi-CLOCK은 RMW 수행시간이 비교적 완만한 증가를 보였다. 잠금 데이타의 공유수가 증가할수록 잠금의 성능이 더욱 차이가 나는 것은 TICOM 방식의 잠금 장치는 메모리가 잠금을 제어함으로 하나의 RMW 동작에 필요한 버스 사용이 HiPi-CLOCK보다 2배이고 이는 버스의 트래픽의 증가를 초래하였다. 또한 RMW 사이클중 쓰기인 경우 메모리를 참조하므로 메모리의 사용율이 증가하고 이에따라 HiPi-bus 상에서 메모리의 응답거부가 발생하여 버스의 트래픽이 더욱 증가하기 때문이다. 이는 잠금 경합이 빈번할 때도 하나의 RMW 동작 수행 시간이 HiPi-CLOCK이 짧음을 말한다.

V. 결 론

잠금 장치는 다중 프로세서 시스템의 동기화를 위해 필수적이다. 잠금 장치는 잠금 경합이 없을때 잠

금 동작에 필요한 시간을 줄이고 잠금 경합이 빈번히 발생할 경우에도 잠금 동작에 필요한 시간을 줄이는 방법이 필요하다. 그러나 기존의 메모리에서 잠금을 제어하는 방식은 버스 트래픽의 증가와 메모리의 사용율의 증가를 초래한다. 본 논문에서는 HiPi-bus를 사용하는 다중 프로세서 시스템에서 잠금 경합이 없을때 잠금 동작에 필요한 시간을 줄이기 위해서 잠금 데이타를 캐쉬에 저장하고 이를 효율적으로 관리 하는 방식을 제안하였고, 잠금 경합이 빈번할때 잠금 동작에 필요한 시간을 줄이기 위해 캐쉬에서 캐쉬로의 데이타 전송 방식을 설계하였다. 이의 결과를 위해 시뮬레이터를 제작하고 인공적인 트레이스 상에서 메모리에서 잠금을 제어하는 방식과 제안한 HiPi-CLOCK의 성능을 하나의 RMW 동작의 수행시간으로 비교하였다. 잠금 경합이 적을 경우에 언제나 두배 이상의 성능을 보였고 잠금 경합이 많을 경우에는 잠금 데이타의 공유수가 증가할수록 성능이 우수하였다. 앞으로 실제 시스템에서 트레이스를 얻어 보다 잠금 장치의 정확한 성능 분석이 필요하다.

參 考 文 獻

[1] Kain, Richard Y, Dan I, "Computer Architecture Software and Hardware," vol. 2. Prentice Hall, New Jersey, 1989.

[2] Moldovan, Dan I, "Modern Parallel Processing," Dept. of Elec. Eng. - systems, Univ. of Southern California, LA, CA 90089-071.

[3] Thomas E. Anderson, "The Performance of Spin Lock Alternatives for Shared Memory Multiprocessors," *IEEE Trans. on Paralledl and Distributed systems*, vol. 1, bi. 1,. Jan. 1990.

[4] P. Bitar and A. M. Despain, "Multiprocessors Cache Synchronization Issues, Innovations, Evolution." Proceedings of the 13th ISCA, pp. 424-423, 1986.

[5] 기 안도외 3인, "TICOM 시스템 버스 사용자 지침서 (V1.1)," TD88-6121-133.B, 행정전산망 주전산기 개발본부, 한국전자통신연구소, 1990.11.

[6] 원 철호외 4인, "다중 프로세서의 캐쉬 메모리, "The ETRI journal, vol.10, no.3, 한국전자통신연구소, 1988.

[7] 기 안도, 박 병관, 윤 용호, "펜디드 프로토콜에서의 동기화 요소," The ETRI journal, vol.12, no.1, 한국전 자통신연구소, 1990.

[8] 장 준석외 5인, "프로세서 관리 유니트 설계서 R2.0," TD89-6140-302.B, 행정전산망 주전산기 개발본부, 한국전자통신연구소, 1989.

[9] Rudolph, 1., and Segall., Z, "Dynamic Decentralized cache schemes for MIND parallel processors," In Proceedings of the International Symposium on Computer Architecture, 1984.

[10] Toshiaki Tarui, Takayuki Nakagawa, Noriyasu Ido, Machiko Asaie and Mamoru Sugie, "Evaluation of the Lock Mechanism in a Snooping Cache," In Proceedings of International Conference on Supercomputing, 1992.

著 者 紹 介

尹 龍 鎬 (正會員)

1949年 10月 15日生. 1975年 2月 한양대학교 전자공학과 졸업(공학사). 1981年 2月 연세대학교 전자계산학과 졸업(공학석사). 1988年 9月 한양대학교 전자공학과 박사과정 수료. 1993年 2月 현재 한국전자통신연구소 책임연구원. 주관심분야는 컴퓨터구조, 병렬처리, ASIC 설계, CAD 등임.

林 寅 七 (正會員) 第29卷 A編 第8號 參照

현재 한양대학교 전자공학과 교수