슬롯링으로 연결된 다중처리기 시스템에서 최적화된 캐쉬일관성 프로토콜

민 준 식[†] · 장 태 무[††]

요 약

다중처리기 시스템에서 여러 처리기 캐쉬들 간에 일관성을 유지하기 위한 정책에는 기록무효화 정책과 기록갱신 정책이 있다. 기록 무효화 정책은 처리기가 캐쉬 블록에 기록을 시도할 때마다 다른 캐쉬에 저장된 동일한 모든 복사본을 무효화한다. 이러한 빈번한 무효화로 인하여, 기록 무효화 정책은 캐쉬 적중률이 낮다. 반면에 기록 갱신정책은 동일한 블록을 무효화 시키는 것이 아니라 동시에 갱신하는 정책이다. 이러한 정책의 경우에 블록의 공유 여부에 상관없이 갱신된 내용을 상호 연결망을 통하여 전송해야만 하며 이로 인하여 상호 연결망상에 교통량이 폭주하게 된다. 본 논문에서는 슬롯링으로 연결된 공유메모리 다중처리기 시스템에서 효율적인 캐쉬 일관성 정책을 제안한다. 제안된 프로토콜은 기록 갱신정책을 기반으로 하며 공유된 블록을 갱신할 경우에만 갱신된 내용을 전송한다. 반면 갱신된 블록이 공유되지 않은 블록이면 갱신된 내용을 전송하지 않는다. 본 논문에서는 제안된 프로토콜을 분석하고 시뮬레이션을 통하여 기존의 프로토콜과 성능을 비교한다.

An Optimized Cache Coherence Protocol in Multiprocessor System Connected by Slotted Ring

Jun-Sik Min[†] · Tae-Mu Chang[††]

ABSTRACT

There are two policies for maintaining consistency among the multiple processor caches in a multiprocessor system : *Write invalidate* and *Write update*. In the write invalidate policy, whenever a processor attempt to write its cached block, it has to invalidate all the same copies of the updated block in the system. As a results of this frequent invalidations, this policy results in high cache miss ratio. On the other hand, the write update policy renew them, instead of invalidating all the same copies. This policy has to transfer the updated contents through interconnection network, whether the updated block is private or not. Therefore the network suffer from heavy transaction traffic. In this paper we present an efficient cache coherence protocol for shared memory multiprocessor system connected by slotted ring. This protocol is based on the write update policy, but the updated contents are transferred only in case of updating the shared block. Otherwise, if the updated block is private, the updated contents are not transferred. We analyze the proposed protocol and enforce simulation to compare it with previous version.

1. 서 론

공유 메모리 다중 처리기 시스템(Shared Memory Multiprocessor System)은 공유 메모리 모듈에 연결된 다중 처리기에 의해서 단일 처리기의 처리능력과 속도의 한계를 극복하기 위하여 개발되었다[11]. 이러한 시스템 환경하에서 처리기들을 연결하는 상호 연결망(Interconnection Network)에 대한 접근 충돌(Access Conflict)과 메모리 접근 지연(Access Latency) 문제를 해

※ 본 연구는 동국대학교 논문게재연구비 지원으로 이루어졌음.
† 준 회 원 : 한국전산원 주임연구원
†† 정 회 원 : 동국대학교 컴퓨터·멀티미디어공학과 교수
논문접수 : 2000년 9월 6일, 심사완료 : 2000년 12월 18일

결하기 위해서 각각의 처리기에 고성능의 작고 빠른 캐쉬 메모리(Cache Memory)를 설치하게 되었다[1]. 이 경우에 동일한 메모리 블록(Block)의 많은 복사본이 특정한 시간에 시스템 내에 있는 여러 개의 캐쉬에 존재하게 되어 이들간에 일관성을 유지하기 위한 적절한 방법이 모색되어야 한다[12]. 공유 메모리 다중 처리기 시스템에서 일관성을 유지하기 위한 정책은 크게 기록 무효화 정책(Write Invalidate Policy)과 기록 갱신 정책(Write Update Policy)으로 나누어진다[1].

기록 무효화 정책은 처리기가 캐쉬 블록에 대해서 갱신을 시도할 경우에 시스템 내에 있는 동일한 모든 캐쉬 블록들을 무효화 시키는 것이다. 이런 일관성 정책은 공유 블록들간에 빈번한 무효화로 인하여 캐쉬 실패(Cache Miss)가 상대적으로 높다. 또한 이러한 캐쉬 실패들은 수정된 블록이 시스템 내에 있는 캐쉬들 사이에 빈번하게 이동하는 Pingpong Effect[1] 현상을 유발하며 상호 연결망에 교통량을 증가 시킨다. 이러한 원인으로 인해서 메모리 참조 요구에 대한 지연시간이 증가하며 결국은 성능의 감소를 가져온다.

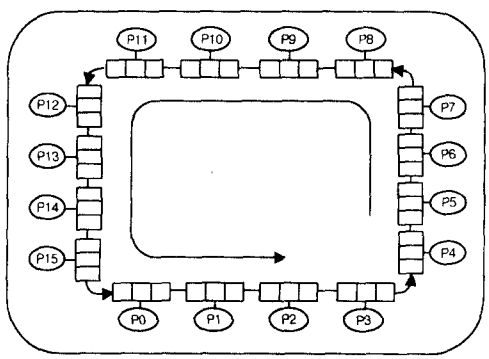
반면에 기록 갱신 정책은 처리기가 캐쉬 블록의 내용을 갱신할 경우에 시스템에 있는 여러 개의 캐쉬에 저장되어 있는 모든 동일한 블록들을 무효화 시키는 것과는 달리 동시에 갱신하는 방법이다. 기록 갱신 정책의 가장 큰 단점은 일관성 유지를 위한 동작이 공유 블록에 대해서만 필요함에도 불구하고 모든 캐쉬 블록에 대해서 고려하는 것이다. 따라서 상호 연결망상에 불필요한 메세지 전송으로 인해서 교통량이 폭주하게 되며 그 결과로 인한 연결망에 대한 접근 충돌은 시스템의 성능을 저하시키는 주요 원인이 된다.

본 논문에서는 슬롯링(Slotted Ring)[2]을 상호 연결망으로 하는 공유 메모리 다중 처리기 시스템에서 메모리 참조 요구시 캐쉬들간에 공유되는 블록의 정보에 따라서 효율적으로 일관성을 유지하는 새로운 형태의 캐쉬 일관성 프로토콜(Cache Coherence Protocol)을 제안한다. 새로운 프로토콜은 기본적으로 기록 갱신 정책과 지연 쓰기(Write Back)를 바탕으로 하며 처리기가 블록을 갱신할 경우에 해당 블록이 공유되는 블록이면 갱신된 내용을 상호 연결망을 통하여 전송한다. 반면에 공유되지 않는 전용 블록이면 갱신된 내용을 전송하지 않는다. 기존의 프로토콜[2]과 본 논문에서 소개되는 새로운 프로토콜을 시뮬레이션을 통하여 성능을 비교함으로써 새로운 프로토콜이 슬롯링을 상호 연결망으로 하는 공유 메모리 다중 처리기 시스템에서 보다 효과적으로 동작할 수 있음을 확인한다.

2. 슬롯링의 구조

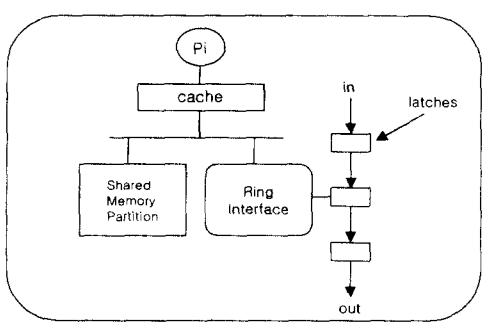
공유 메모리 다중 처리기 시스템의 상호 연결망으로서 상업적으로 가장 널리 채택된 것은 처리기들간에 공유되는 버스이다. 버스를 이용하는 시스템은 버스 자체가 가지고 있는 위상(Topology)으로 인한 문제점[3] 때문에 대역폭(Bandwidth)에 한계가 있으며, 더구나 현대의 비약적인 반도체 기술의 발달로 인한 고성능 처리기의 등장으로 공유 버스를 기반으로 하는 다중 처리기 시스템은 한계점에 도달하게 되었다[3].

이러한 공유 버스가 가지는 문제점을 극복하기 위하여 여러 가지 상호 연결망이 제안되었다. 그 중의 하나가 점대점 단방향(Point-to-Point Unidirectional) 상호 연결망으로서 슬롯링은 가장 간단한 형태의 하나이다.



(그림 1) 슬롯링의 구성

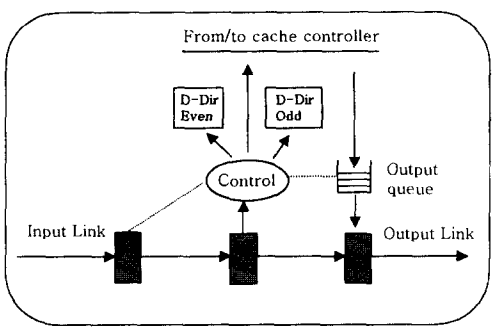
슬롯링은 (그림 1)과 같이 노드들로 구성되며 한 방향으로만 자료를 전송한다. 각각의 노드는 (그림 2)와



(그림 2) 처리기 노드 상태도

같이 지역캐쉬(Local Cache)에 연결된 처리기, 메모리 모듈과 링 접속기(Ring Interface) 등으로 구성되어 있다. 또한 링에서의 전송은 동일한 클럭에 의해서 동기화 되며, 각각의 처리기 연산에 대해서는 비동기적이다.

(그림 3)은 링 접속기를 나타낸 것이다. 링 접속기는 지역캐쉬와 슬롯링간에 메세지 송수신과 링을 통하여 전송되는 일관성 유지에 관련된 명령어들을 감지하는 스누핑(Snooping) 기능을 담당한다. 유용한 정보들을 갖는 메세지는 패킷 슬롯에 채움으로써 링에 삽입되며, 각각의 패킷 슬롯을 비어 있는 상태로 표시함으로써 링에서 제거된다. 링 접속기는 메세지의 첫번째 비트를 검사해서 링으로부터 제거할 것인지 다음 노드로 전송할 것인지를 빠른 속도로 결정할 수 있고 이러한 작동 원리는 슬롯링 상호 연결망 구조에서 성능에 영향을 미치는 매우 중요한 사항이다.



(그림 3) 링 접속기의 상태도

각 링 클럭 사이클(Cycle)에서 64비트 길이의 패킷 슬롯(Packet Slot)이 파이프라인(Pipeline)의 한 단계에서 다음 단계로 이동한다. 슬롯링은 3 * P단계의 원형 파이프라인으로 구성된다. 따라서 링에는 총 3 * P개의 슬롯이 순환하게 된다. 여기서 P는 처리기 노드의 수이다.

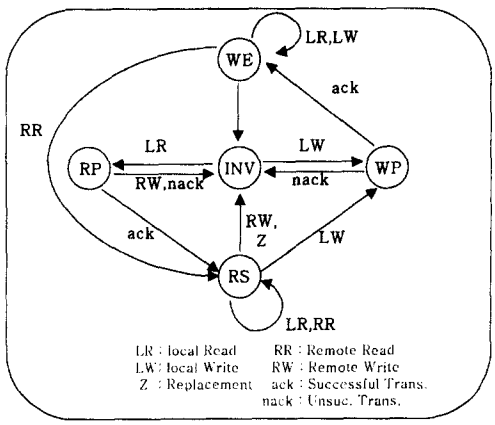
(그림 2)에서 볼 수 있듯이 공유 메모리는 링에 있는 각각의 노드에 분산되어 있으며, 시스템에 있는 모든 처리기들은 하나의 동일한 주소영역을 사용하고, 블록이 사상(Mapping)되는 노드를 홈 노드(Home Node)라 한다. 하나의 더티 비트(Dirty Bit)가 현재 메모리 블록이 가장 최근 것인지의 여부를 나타내기 위하여 사용된다. 노드에 도착된 메세지는 지역 캐쉬 디렉토리의 복사본을 가지고 있는 홀수와 짝수로 나누어진 2-방향 인터리브 이중 디렉토리(2-Way Interleaved Dual Directory)에 의해서 처리된다. 이 때 스누핑과 캐쉬에 대한 처리기의 접근은 서로 간섭이 일어나지 않는다. 각각의 지역 캐쉬는 홈 노드의 메모리 블록이나 다른 노드의 메모리 블록을 유지할 수 있다.

3. 슬롯링에서의 캐쉬 일관성 프로토콜

공유 메모리 다중 처리기 시스템에서 여러 캐쉬들에 의해서 공유되는 블록들간에 일관성을 유지하기 위해서 하드웨어로 구현되는 프로토콜은 크게 두 종류로 나누어진다. 하나는 공유 버스를 기반으로 하는 시스템과 같이 메모리 참조 요구를 공유 버스를 통하여 방송(Broadcasting)하며, 일관성은 각각의 캐쉬가 버스를 스누핑함에 의해서 유지되는 스누피 프로토콜(Snoopy Protocol)[14]이다. 다른 하나는 상대적으로 방송이 불리한 다단계 상호 연결망(Multistage Interconnection Network)에서와 같이 일관성을 유지하기 위하여 공유되는 캐쉬 블록에 대한 정보를 메모리에 별도로 유지하는 디렉토리를 기반으로 하는 방법(Directory-based Scheme)[15]이다.

슬롯링 구조에서는 방송이 가능하고 메모리 접근 지연시간 측면에서 유리하며, 효율성이 좋고 간단한 디렉토리 유지로 인해서 메모리 낭비가 없는 스누핑 프로토콜이 적합하다[3]. 슬롯링을 상호 연결망으로 하는 시스템에서 채택된 기존의 스누핑 프로토콜[2]은 지연 쓰기와 기록 무효화를 바탕으로 한다. 시스템에서 동일한 메모리 블록의 사본들간에 일관성을 유지하기 위하여, 캐쉬 블록은 Invalid(Inv), Read Shared(RS), Write Exclusive(WE) 상태 중의 하나를 유지한다. 블록은 실제 응용 프로그램이 실행되는 동안에 각 처리기의 메모리 참조 요구에 따라 일관성을 유지하며 상태를 변환한다.

(그림 4)는 기존의 슬롯링 프로토콜에서 캐쉬에 있는 블록들의 상태 변환을 나타내고 있다. 일관성을 위한 동작들은 공유 메모리 관점에서 일관성을 유지하기 위하여 프로토콜에 의해서 수행되는 일련의 절차들이다. 일관성 동작은 읽기 실패(Read Miss), 쓰기 실패(Write Miss), 무효화(Invalidation), 블록 대치(Replacement) 등의 경우에 대해서 필요하다.



(그림 4) 캐쉬블록의 상태 전이도

4. 새로운 프로토콜

4.1 개요

본 논문에서는 기록 무효화 정책의 무효화로 인한 캐쉬 실패와 기록 갱신 정책의 블록 갱신시 마다 갱신 내용을 전송하기 위한 상호 연결망에서의 교통량 증가로 야기되는 오버헤드(Overhead)를 효과적으로 줄일 수 있는 새로운 프로토콜을 제안한다. 새로운 프로토콜은 별다른 하드웨어 추가 없이 기록갱신과 기록무효화 캐쉬 일관성 정책의 단점을 효율적으로 보완할 수 있으며 기본적으로는 기록 갱신 정책과 지연 쓰기를 바탕으로 한다.

이 프로토콜은 블록이 갱신되는 시점에서 해당 블록이 여러 캐쉬들간에 공유되는 블록이면 블록을 갱신한 후에 갱신된 내용을 링을 통하여 방송한다. 반대로 해당 블록이 공유되지 않는 전용 블록이면 갱신 후에 갱신된 내용을 방송하지 않는다. 이러한 프로토콜을 구현하기 위해서는 처리기가 캐쉬 블록에 대해 갱신을 하고자 하는 경우에 시스템에 있는 다른 캐쉬가 동일한 블록을 공유하고 있는지에 대한 여부를 알아야 한다. 이러한 정보는 슬롯링 매커니즘(Mechanism)하에서 간단하게 얻을 수 있으며 별다른 하드웨어 추가 없이 위에서 지적한 두 가지 캐쉬 일관성 정책의 단점을 효율적으로 보완할 수 있다.

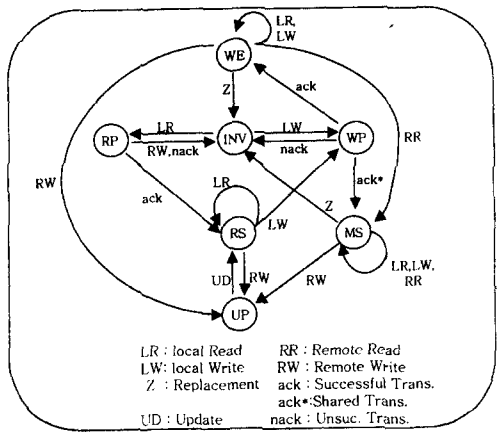
4.2 일관성 유지를 위한 상태 분석

응용 프로그램이 실행되는 동안에 일관성 동작은 읽기 실패, 쓰기 실패, 쓰기 적중(Write Hit), 블록 대치의 경우에 대해서 필요하다. 메모리 블록의 복사본은 다음의 상태중 하나를 유지하며 처리기의 메모리 참조 요구에 따라서 적절하게 상태를 변환함으로써 일관성을 유지한다.

① Invalid(INV) : 일관성이 없거나 캐쉬에 존재하지 않는 상태.
② Read Shared(RS) : 일관성이 있으며 동일한 사본이 다수 존재할 수 있는 상태.
③ Write Exclusive(WE) : 해당 블록은 수정되어 메모리 블록과 일관성이 없으며, 시스템 내에서 유일하게 정확성을 유지하는 상태.
④ Modified Shared(MS) : 해당 블록을 수정되어 메모리 블록과 일관성이 없으며, 캐쉬들간에 공유되어 있는 상태

공유 버스를 상호 연결망으로 하는 시스템에서는 블록들간에 일관성을 유지하기 위한 상태 변환시에 버스에 중재기(Arbitrator)가 있어서 일관성 유지를 필요로 하는 사건(Event)들 사이에 중재를 담당하므로 충돌이 일어나지 않는다. 그러나 슬롯링을 상호 연결망으로 하는 다중 처리기 시스템에서는 공유 버스를 기반으로 하는 시스템과는 달리 메모리 참조 요구를 가진 여러 개의 메세지가 특정한 시간 내에 동시에 링을 순환할 수 있다. 이 때에 하나 이상의 노드가 동시에 같은 블록에 대해서 메모리 참조 요구를 할 경우 충돌이 일어나게 되며, 이를 해결할 수 있는 적절한 방법이 요구된다.

링 프로토콜에서는 Read Pending(RP)과 Write Pending (WP)의 두 가지 부가적인 상태로서 이러한 충돌을 해결한다. 노드간에 메모리 참조 요구가 충돌이 일어날 경우는 어느 한 노드만 상태 변환, 즉 메모리 참조 요구가 허락되며 나머지 참조 요구들은 상태 변환이 허락되지 않으며 참조 요구를 재시도 해야만 한다. 또한 본 논문에서 소개하는 새로운 프로토콜은 Update Pending(UP)상태를 추가한다. 이는 RS 상태에 있는 블록이 원격 노드로부터 블록 갱신 요구를 받았을 경우에 해당 블록이 공유되어 있음을 알리며 차후에 갱신된 내용을 전송 받기 위한 캐쉬 블록 상태이다. 이러한 Pending 상태는 실제 캐쉬 디렉토리에는 존재하지 않는 상태이며, 스누퍼(Snooper)에 있는 레지스터에 블록의 참조 요구가 Pending 상태에 있다는 정보만 유지하게 된다.



(그림 5) 새로운 프로토콜의 캐쉬블록 상태 전이도

Pending상태에 있는 캐쉬 블록은 그 블록의 참조 요구에 대한 응답을 아직 받지 않은 상태이며 블록 미스에 대한 응답은 블록을 포함하고 있는 메세지이다. 만약 Pending 상태에 있는 블록들이 요구에 대한 긍정 응답(Acknowledgment)을 받지 못하면 상태 변환을 포기하고 그 블록에 대한 참조 요구를 재시도해야만 한다. 그림 5는 새로운 슬롯링 프로토콜에서 캐쉬에 있는 블록들의 상태 변환을 나타내고 있다.

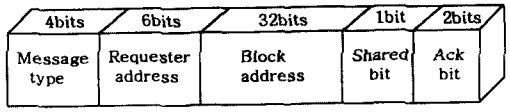
4.3 일관성 유지를 위한 메시지

슬롯링에서의 프로토콜은 일관성을 유지하기 위해서 슬롯링을 통하여 전달되는 메세지의 집합으로 구현될 수 있다. 각 노드의 물리적 주소공간에 사상되는 캐쉬 블록 미스는 지역 미스(Local Miss)라고 한다. 이 경우에 읽기 실패에 대해서는 해당 블록의 메모리가 변경되지 않았으면 일관성을 유지하기 위한 별도의 메세지를 필요로 하지 않으며, 그 밖의 실패나 쓰기 적중, 블록 대치는 일관성을 유지하기 위하여 링을 통하여 메세지를 전달하게 된다.

4.3.1 메시지 형태

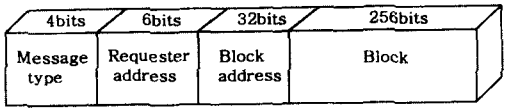
메세지 형태는 다음과 같은 2가지 종류로 나누어진다.

① 조회(Probe) 메세지 : 메모리 참조 요구나 일관성 유지를 위한 짧은 형태의 메세지로서, Read-block, Write-block, Write-hit 등이 있다. 다음은 내부 구성 형태이다.



(그림 6) 조회 메세지 구성도

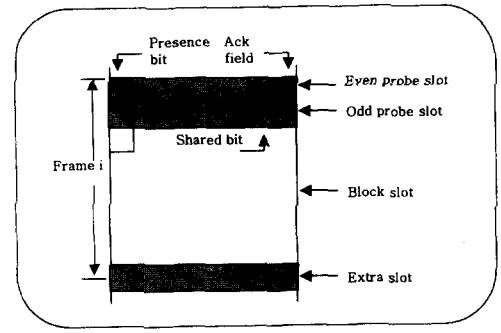
② 블록(Block) 메세지 : 조회 메세지에 대한 응답을 하기 위해서 블록을 포함하고 있는 긴 형태의 메세지이다. Send-block 과 Shared-update 메세지가 여기에 해당된다.



(그림 7) 블록 메세지 구성도

본 논문에서 소개하는 새로운 프로토콜의 가장 핵심적인 사항인 공유되는 블록에 대한 정보를 얻기 위하여 조회 메세지에 공유 비트 필드를 추가함으로써 조회 메세지를 확장한다. 이러한 방법은 슬롯링 구조에서 블록을 갱신하고자 할 때 반드시 홈 노드나 더티 노드에 의해서 긍정응답을 받아야 하며 조회 메세지는 반드시 링을 한 번 순환한다는 개념에서 착안된다.

조회 메세지는 슬롯링을 통해 전송하기 위해서 단지 하나의 64bits 패킷 슬롯(Packet Slot)과 하나의 파이프라인 단계를 필요로 한다. 반면에 블록 메세지는 5개의 연속적인 패킷 슬롯과 파이프라인 단계를 필요로 한다. 이러한 두 종류의 메세지는 링에서 조회 슬롯(Probe Slot)과 블록 슬롯(Block Slot)에 실리게 되며, 슬롯들이 모여서 하나의 프레임(Frame)을 구성하게 된다.



(그림 8) 프레임의 구성도

각각의 프레임은 짝수 주소 블록에 대한 짝수 조회

슬롯(Even Probe Slot)과 홀수 주소 블록을 위한 홀수 조회 슬롯(Odd Probe Slot), 블록 슬롯, 그리고 노드간에 인터럽트(Interrupt)신호를 위한 하나의 부가적인 슬롯(Extra Slot)으로 구성된다. 그림 8은 프레임의 구성을 나타낸 것이다.

4.4 일관성 유지를 위한 명령어와 응답

<표 1>은 일관성 유지에 관련된 명령어와 그에 대한 응답을 나타낸 것으로 명령어와 응답은 메세지 형태로 전송된다.

〈표 1〉 일관성 명령과 관련된 응답 메세지

사 건	명 령	응 답
읽기 실패	Read-block	Send-block
쓰기 실패	Write-block	Send-block None(non-shared) Shared-update
쓰기 적중	Write-hit	None(non-shared) Shared-update
블록 대치	Send-block	None

4.5 충돌에 대한 해결

여러 개의 노드로부터 동시에 같은 블록에 대한 참조 요구는 한 노드에 대해서만 허락함으로써 충돌에 따른 문제점을 해결한다. 이러한 과정은 조회 슬롯에 있는 <Ack>필드를 통한 긍정응답으로 이루어진다. 블록 미스에 대한 긍정응답은 블록 참조 요구를 받았으며 곧 해당 블록을 전송할 것이라는 의미이다. 갱신요구에 대한 긍정응답은 특정한 시간에 유일하게 그 블록을 수정할 수 있는 노드로 지정 받기 위함이다. 프레임 i의 짝수(홀수) 조회 메세지에 대한 긍정응답은 바로 다음에 연속적으로 이어지는 프레임 i+1의 짝수(홀수) 조회 슬롯의 <Ack>필드를 통하여 전송된다. 이와 같이 긍정응답을 같은 조회 슬롯에 피기백(Piggy-back)하지 않고 다음에 이어지는 연속적인 프레임의 슬롯에 있는 <Ack>필드를 이용함으로써 인터리브 이중 디렉토리에 대해서 충분한 디렉토리 조회시간을 제공한다.

4.6 일관성 동작에 대한 분석

다음은 슬롯링에서 일관성을 위한 동작과 관련된 데이타의 이동 블록의 상태 변환을 기술함으로써 보다 상세하게 프로토콜의 일관성 유지를 위한 동작들을 나타낸다. 응용 프로그램이 실행되는 동안에 캐쉬에 대한 각 처리기의 메모리 참조 요구에 대해서 일관성 유지를 필요로 하는 경우는 다음과 같다.

① 읽기 실패의 경우 : 읽기 실패는 Inv상태에 있는 블록을 참조하고자 할 때 발생한다. 이 때 블록을 요구한 노드가 홈 노드이고 메모리 블록이 변경되지 않았으면 바로 블록을 메모리에서 캐쉬로 적재하면 된다. 그렇지 않으면 블록 상태를 Inv에서 RP상태로 변환하고 링에 Read-block 메세지를 전송해서 블록을 요구한다. 이에 대한 응답은 홈 노드가 Send-block메세지로 한다. 그러나 홈 노드의 메모리 블록 상태가 Invalid 이면 그에 대한 응답은 소유권한(Ownership)을 가진 노드(WE, MS)가 Send-block 메세지로 한다. 소유권한 노드가 WE일 경우 해당블록은 더 이상 WE 상태가 아니라 MS상태가 된다. RP 상태는 미스에 대한 응답으로 블록을 포함하고 있는 Send-block 메세지를 받았을 때 RS 상태가 된다. 만약 RP 상태에 대한 메모리 참조 요구가 긍정응답을 받지 못하거나 원격 노드로부터 갱신 메세지를 받으면 RS 상태로 변환을 포기하고 메모리 참조 요구를 재시도 해야만 한다.

② 쓰기 실패의 경우 : 처리기가 Inv 상태의 블록을 참조 요구하는 경우이며 해당 블록의 상태를 즉시 WP로 전환하고 링을 통하여 Write-block 메세지를 전송한다. 이때 Write block 메세지는 2가지 기능을 수행한다. 하나는 미스된 블록을 요구하는 기능이며, 다른 하나는 미스된 블록이 다른 캐쉬에 공유되어 있는지에 대한 여부를 조사하는 기능이다. 시스템에 있는 캐쉬들은 RS 상태에 있는 블록에 대해서 Write-block 메세지를 받으면, 해당 블록을 공유하고 있다는 신호로서 연속적인 다음 프레임의 공유 필드를 셑하고 UP 상태로 전환한다. 이후에 이 블록에 대한 갱신 내용이 방송되면 해당 블록을 갱신하고 UP에서 다시 RS 상태로 전환한다. Write-block 메세지에 대한 응답은 해당 메모리 블록이 Valid이면, 홈 노드가 블록 요구를 받았으며 곧 블록을 전송할 것이라는 의미로 다음 프레임의 <Ack> 필드를 셑하고 이후에 Send-block 메세지로서 이루어진다. 그러나 홈 노드 메모리 블록이 Invalid이면 응답 책임은 소유권한 노드(WE, MS)에 있으며 소유권한 노드는 <Ack> 필드와 공유 비트 필드를 셑하고 Send-block 메세지로 응답한다. 공유 비트 필드를 셑하는 이유는 소유권한 노드의 해당 블록은 더 이상 WE(MS)가 아니며 상태를 UP로 변환해서

차후에 갱신된 내용을 전송 받기 위함이다.

미스된 노드는 Write-block 메세지가 링을 순환한 후 되돌아오면 링에서 제거한 후 다음 프레임의 <Ack> 필드와 공유 필드를 조회해서 긍정응답과 해당 블록의 공유 여부를 결정한다. 공유 필드가 셀되어 있는 경우는 공유 블록의 경우로서 블록을 갱신한 후 Shared-update 메세지를 방송해서 UP 상태의 블록을 갱신하며 이후 캐쉬 블록의 상태는 MS이다. 공유 필드가 셀되어 있지 않으면 공유되지 않은 전용 블록으로서 Shared-update 메세지를 전송하지 않는다. 블록은 WE 상태로 유지되며 노드는 차후에 이 블록에 대한 요구에 대해서 응답의 책임이 있다. 만약 Write-block 메세지에 대해서 긍정응답을 받지 못하면 다른 노드와 그 블록에 대한 참조가 충돌이 난 경우로서 Write-block 메세지를 재전송해야 한다.

③ 쓰기 적중의 경우 : RS, MS 상태에 있는 블록에 대하여 갱신을 하고자 하는 경우이다. 먼저 RS 상태일 경우 Write-hit 메세지를 링을 통하여 전송한다. Write-hit 메세지는 동일한 블록을 다른 캐쉬가 공유하고 있는가에 대한 조회와 다른 캐쉬에 해당 블록을 갱신 하고자 함을 통보하는 기능을 한다. 이 경우에 블록을 채취(Fetch)할 필요는 없으며, 이에 대한 응답은 홈 노드로부터 갱신에 대한 긍정응답만 받으면 된다. 동일한 블록을 소유하고 있는 다른 캐쉬는 Write-hit 메세지가 방송되면 상태를 UP로 변환하고 공유 비트 필드를 셀해서 공유되어 있는 블록임을 알리며 이후에 Shared-update 메세지가 방송되면 블록을 갱신하며 RS 상태가 된다. Write-hit 메세지가 되돌아오면 해당 노드는 링으로부터 메세지를 제거하고 다음 프레임의 <Ack> 필드와 공유 필드를 조회한다. 공유 블록인 경우에는 갱신을 한 후에 Shared-update 메세지를 방송해서 UP 상태에 있는 동일한 블록을 갱신한다. 이후의 블록 상태는 소유권한을 갖는 MS 상태이다. 전용 블록일 경우는 메세지를 방송하지 않으며 해당 블록은 WE이다. MS상태일 경우는 쓰기 요구에 대한 권한을 부여할 수 있는 소유권한을 가지고 있으며 공유된 상태로서 해당 블록을 갱신하고 그 내용을 Shared-update 메세지를 통하여 방송한다.

④ 블록 대치 : WE, MS 상태에 있는 캐쉬 블록은 수정되어서 메모리 블록과 일관성을 유지하고 있지 않으며 블록요구에 대한 응답 권한을 가지기 때문에 대치될 경우에는 홈 노드의 해당 메모리 블록을 갱신 해야만 한다. RS 상태에 있는 블록은 메모리 블록과 일관성을 유지하기 때문에 대치되더라도 일관성 유지를 위한 별도의 동작을 필요로 하지 않는다.

5. 프로토콜의 정당성

일반적인 다중 처리기 시스템에서 다수의 프로세스(Process)가 동일한 기억장치의 영역에 접근하므로 프로그램의 정확한 수행을 위하여 사건 순서 모형(Event Ordering Model)이 필요하다. 그 중 가장 엄밀한 모형이 Lamport의 순차 일치성(Sequential Consistency)[5]이며, Scheurich와 Dubois는 캐쉬가 기반이 된 시스템에서 순차 일치성의 충분조건을 다음과 같이 제시하였다[6].

① 모든 처리기는 프로그램에서 명시된 순서대로 기억장치에 접근한다.
② 처리기는 이전의 접근들이 전역적으로 수행되지(Globally Performed) 않으면 새로운 접근을 시작하지 않는다.

여기서 쓰기(Write)가 전역적으로 수행 되었다 함은 수정된 상황이 모든 처리기에게 알려짐을 의미하며, 읽기(Read)의 경우는 돌려 받는 값이 정해져 있고 그 값을 수정한 쓰기가 전역적으로 수행된 것을 의미한다.

본 논문에서 제시된 프로토콜의 경우,

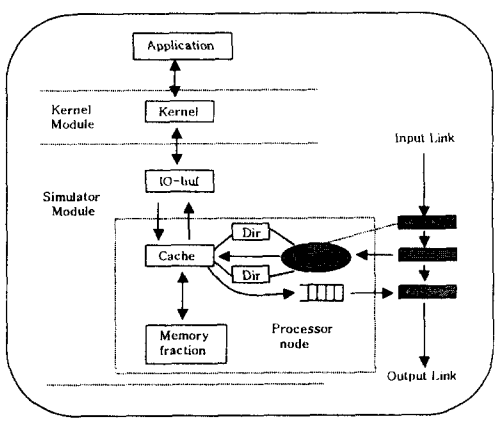
① 모든 처리기는 프로그램에서 명시된 순서대로 기억장치에 접근한다.
② 쓰기의 경우에 블록의 상태가 RS일 때 동작 이전에 갱신 신호를 방송하여 모든 캐쉬에 알리며, 캐쉬에 존재하는 같은 블록을 동시에 갱신한다.
③ 읽기의 경우에도 각각의 캐쉬는 링을 관찰하고 있고, 블록의 상태가 일관성이 있으므로 이전의 쓰기가 MS나 WE로 된 상태의 블록을 읽어가면 만족한다.

이와 같이 순차 일치성을 만족하므로 본 논문의 프로토콜로서 병렬 프로그램을 안전하게 실행함을 확인할 수 있다.

6. 성능평가

6.1 모의실험 환경

본 논문에서는 Limes[16]를 사용하여 시뮬레이션을 수행하였다. Limes는 실행구동(Execution-driven) 방식의 시뮬레이터로서 스레드 기반(Thread-based)의 응용프로그램을 입력으로 받아서 커널(Kernel) 모듈에서 수행한다. 이때에 메모리 참조 등의 사건이 발생하면 시뮬레이터 모듈을 호출하여 사건을 넘겨주게 된다. 시뮬레이터 모듈은 매 사이클 마다 주어진 사건들을 처리하며 그 결과를 커널에 통보한다. Limes는 i486급 이상의 LINUX를 탑재한 PC에서 수행된다. (그림 9)는 시뮬레이션 모듈 구성을 나타낸 것이다.



(그림 9) 시뮬레이터 모듈 구성

6.2 모의실험 변수 및 응용프로그램

시뮬레이션은 2장에서 기술한 시스템 구조와 3, 4장의 프로토콜을 사용한다. 시뮬레이션에 사용되는 시스템 구성 변수는 <표 2>와 같다. 목표 시스템은 중.소형 규모의 다중처리기 시스템을 가정하여 처리기 수를 8, 16, 32개인 경우에 대하여 실험하였다. 처리기수에 따라 링의 크기가 변화하며 링을 순환하는 프레임의 수가 결정된다. 분산 공유메모리 다중처리기 시스템에서는 분산된 메모리 모듈에 대한 참조 비율에 따라 성능에 영향을 준다. 본 논문에서는 각 처리기가 활성화된 후 참조의 50%를 지역 메모리 모듈에서 참조하도록 하였고 나머지는 각각의 모듈에서 무작위로 참조하도록 하였다.

〈표 2〉 시스템 구성 변수

처리기 수	8, 16, 32
캐쉬 사상(mapping)	2-방향 집합 연관 (2-way set-associative)
캐쉬 크기	8 Kbyte
캐쉬 블록 크기	32byte
캐쉬 블록 수	256개
메모리 참조	20 사이클
캐쉬 참조	1 사이클

각 처리기는 요구한 메모리 참조가 만족될 때까지 대기 상태에 있으며 처리기의 요구가 캐쉬에서 만족되었을 경우를 1사이클로 간주 하였다. 본 논문에서 사용하는 응용프로그램은 스레드 기반의 SPLASH2[17]에서 제공하는 프로그램중에서 fft, barnes, lu, water-spatial등 4개를 사용한다. fft는 여섯 단계의 FFT알고리즘을 수행하며, Lu는 밀집행렬을 하위 삼각행렬과 상위 삼각 행렬로 인수 분해하는 응용프로그램이다. 또한 water-spatial은 액체 상태에서의 물분자의 힘과 전위를 측정한다. <표 3>은 시뮬레이션에서 사용된 응용프로그램이 수행되는 동안 요구한 메모리 참조 형태이다.

〈표 3〉 응용프로그램 메모리 참조 특성

응용 프로그램	총계 (단위: 10^3)	읽기		쓰기		Lock	
		횟수 (단위: 10^3)	비율 (%)	횟수 (단위: 10^3)	비율 (%)	횟수	비율 (%)
Fft	8.14	4.88	59.93	3.26	40.04	113	0.01
barnes	237.12	166.13	70.06	70.93	29.92	2,540	0.01
Lu	49.11	34.45	70.16	14.64	29.82	294	0.01
Water	74.26	56.29	75.80	17.93	24.15	1,674	0.02

※ 처리기 수가 8개일 경우

6.3 모의실험 결과

우리는 각 응용프로그램의 실행에 적용되는 프로토콜에 따라서 블록의 무효화 횟수와 공유블록을 갱신하는 횟수를 측정하였다. 또한 새로운 프로토콜의 경우 공유되는 정보에 따라서 공유블록을 갱신 함으로서 캐쉬 적중률에 미치는 영향과 그에 따른 처리기 이용률 등을 비교하였다.

6.3.1 처리기 캐쉬 평균 적중률

<표 4>는 각 응용프로그램에 대하여 처리기수를 변화시키면서 기록 무효화 정책을 사용하는 기존의 프로토콜과 새로 제안된 프로토콜의 실행 결과로서 시스템

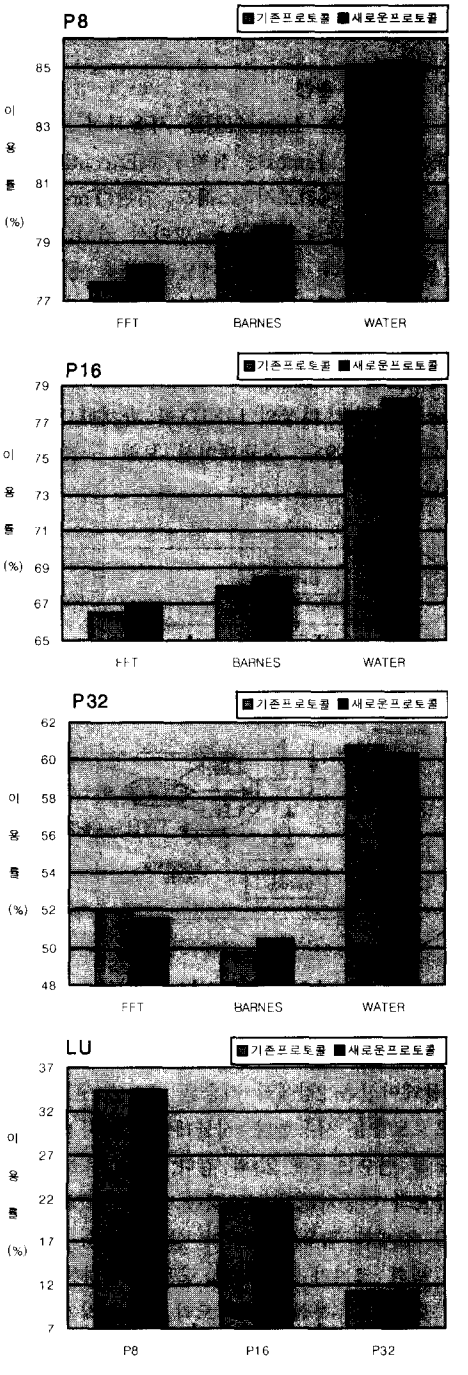
내 처리기 캐쉬들의 평균 적중률을 나타낸 것이다. 각 응용프로그램 별로 차이는 있으나 본 논문에서 제시한 새로운 프로토콜을 사용했을 경우 캐쉬 평균 적중률이 증가 되었음을 볼 수 있다. 캐쉬 평균 적중률의 증가는 처리기수가 적을 때 보다 많을 경우에 많이 증가되었음을 알 수 있다. 각 응용프로그램별 적중률의 증가는 무효화가 많았던 BARNES, WATER 등에서 많이 개선되었음을 볼 수 있다. 전체적으로 블록을 무효화 시키는 기존의 프로토콜 보다는 블록의 공유 정보에 따라서 블록을 갱신하는 새로 제안된 프로토콜이 0.01% 이상의 캐쉬 평균 적중률이 증가되었다. 이는 처리기가 16개인 시스템의 경우 각각의 처리기 캐쉬가 0.01%의 캐쉬 적중률이 향상된 것을 의미하며 결국은 처리기의 메모리 접근시간을 줄여 시스템의 성능을 결정하는 처리기의 이용률이 높아지게 된다. 또한 모의실험에서 사용된 응용프로그램 보다 캐쉬들간에 블록 공유율이 높은 응용프로그램의 경우 보다 높은 캐쉬 적중률 향상이 예상되며, 향후 새로운 병렬 알고리즘의 개발이나 컴파일러의 등장으로 병렬성이 높아지는 경우에 공유되는 블록의 증가로 인해 새로운 프로토콜의 경유가 기존의 프로토콜보다 높은 캐쉬 적중률 향상을 기대할 수 있다.

〈표 4〉 응용 프로그램별 처리기 캐쉬 평균 적중률

응용 프로그램	처리기 수	기존 프로토콜		새로운 프로토콜	
		블록 무효화	캐쉬 평균 적중률(%)	블록 갱신	캐쉬 평균 적중률(%)
FFT	8	121	95.77	1,150	95.78
	16	250	95.72	5,220	95.73
	32	504	96.43	25,383	96.45
BARNES	8	7,958	97.89	15,087	97.91
	16	18,224	97.72	42,035	97.77
	32	35,228	97.63	129,996	97.73
LU (contiguous-blocks)	8	295	99.04	3,384	99.04
	16	613	99.20	16,244	99.20
	32	1,279	99.16	68,051	99.17
WATER (spatial)	8	4,188	99.29	11,747	99.29
	16	9,149	99.11	47,377	99.13
	32	24,036	98.74	244,834	98.78

6.3.2 처리기 이용률

처리기는 응용프로그램 실행시 필요에 따라 메모리 참조 요구를 하게 되며, 요구한 메모리 참조가 만족될 때까지 대기 상태에 있게 된다. 이러한 메모리 참조는



(그림 10) 응용프로그램별 처리기 이용률

먼저 캐쉬에 요구하게 되며, 캐쉬에서 만족되지 않았을 경우 시스템내에 있는 메모리 모듈에 요구하게 된

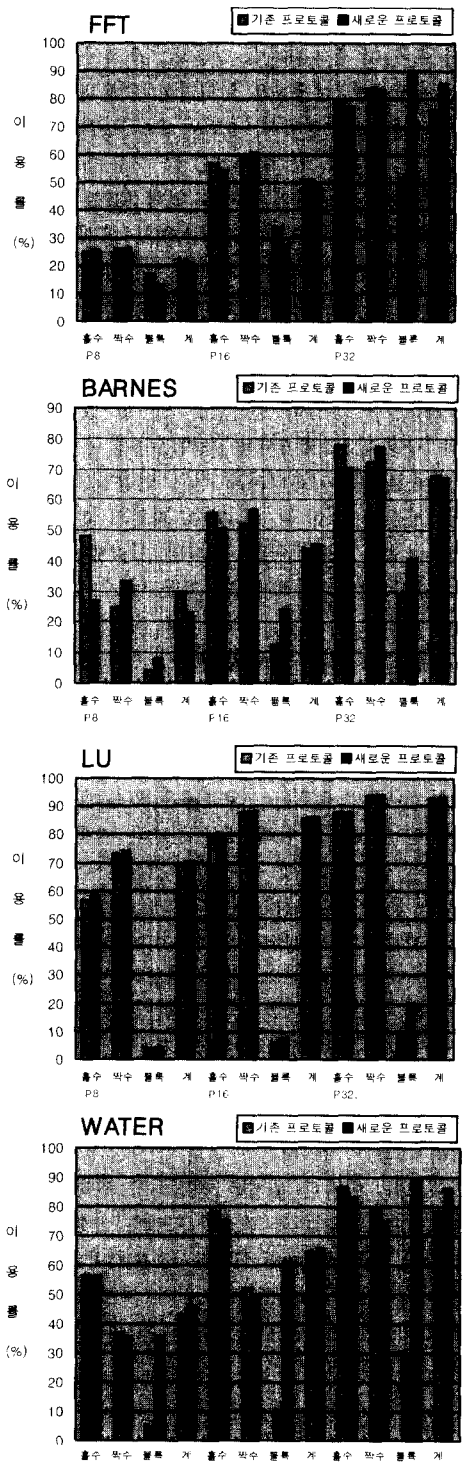
다. <표 2>와 같이 참조요구가 캐쉬에서 적중할 경우에 1사이클 소요되며 메모리에서 참조할 경우 최소 20사이클 이상을 소요하게 된다. 따라서 캐쉬 적중률은 처리기의 대기시간을 결정하는 매우 중요한 요소이다. 처리기 이용률은 처리기 전체 실행 시간중 메모리 참조 대기시간을 제외한 시간의 비율이며 (그림 10)은 응용프로그램별 처리기 이용률을 나타낸 그림이다.

(그림 10)의 결과에서 처리기의 수가 8,16개일 경우에 모든 응용프로그램에서 새로운 프로토콜이 기존 프로토콜과 비교하여 처리기의 이용률이 같거나 향상 되었음을 보여주고 있다. 이는 새로운 프로토콜이 캐쉬 적중률 증가로 인한 메모리 접근지연 시간의 짧아져 처리기의 대기시간이 줄어든 결과이다. 따라서 캐쉬 적중률은 시스템의 성능을 결정하는 매우 중요한 요소이며 작은 적중률의 증가라도 그 이상의 성능이 향상될 수 있음을 의미한다. 처리기의 수가 32개일 경우는 BARNES의 경우를 제외하고는 오히려 처리기의 이용률 감소하는 것을 볼 수 있으며 이는 새로운 프로토콜이 처리기의 수가 증가할 경우 갱신 내용을 전송하기 위한 요구가 급격히 증가하기 때문이며 자세한 사항은 다음 장에서 논한다.

6.3.3 링 이용률

본 논문에서 제안된 프로토콜에서 블록의 공유여부 조사는 최초의 쓰기 요구시 한번만 시도하여 블록의 공유 여부를 결정하기 위한 비용을 최소화 한다. 이는 한번 공유된 블록은 이후에도 계속 공유될 가능성이 매우 크기 때문이다. 따라서 최초의 쓰기 요구시 공유되는 블록으로 간주될 경우 이후의 모든 쓰기에 대해서도 공유되는 것으로 간주하여 갱신시 마다 갱신된 내용을 방송한다. 이렇게 하는 주된 이유는 블록의 갱신시 마다 블록의 공유 여부를 조사할 경우 그로 인한 지연으로 인해 성능이 오히려 감소하기 때문이다.

<표 4>와 (그림 10)의 결과를 분석하여 보면 새로운 프로토콜을 사용할 경우 캐쉬 적중률이 증가하는 만큼 처리기 이용률이 증가하지 않는 경우를 볼 수 있다. 그것은 처리기의 수가 증가함에 따라 공유되는 블록의 수가 증가하는 것이 아니라 공유블록을 소유하는 처리기의 수의 증가로 인하여 공유블록을 갱신하기 위한 요구가 급격히 증가하기 때문이다. (그림 11)은 처리기의 참조요구와 관련한 링 슬롯의 이용률을 나타낸 것으로 홀수 및 짝수 슬롯은 메모리 참조요구 및 공유



(그림 11) 응용프로그램별 링 슬롯 이용률

블록 조사와 관계가 있으며 블록 슬롯은 참조 요구가 실패한 블록 및 갱신 내용을 전송하기 위하여 사용된다.

짝홀수 슬롯의 이용률은 처리기의 수에 관계없이 대부분의 응용프로그램에서 새로운 프로토콜이 기존 프로토콜보다 적음을 알 수 있다. 그 이유는 새로운 프로토콜이 캐쉬 적중률의 증가로 인해 메모리 참조요구가 감소하기 때문이며 공유블록 여부 조사는 메모리 참조 요구시 자연스럽게 이루어짐에 따라 이로 인한 오버헤드도 없음을 알 수 있다. 블록슬롯의 이용률은 응용프로그램중 FFT와 WATER는 처리기의 수가 증가함에 따라 갱신된 내용을 방송하기 위하여 블록 슬롯의 이용률이 급격히 증가함을 알 수 있다. 따라서 이러한 응용프로그램에서는 처리기수가 증가할 경우 캐쉬 적중률이 증가하더라도 상호 연결망상에 교통량 증가로 성능의 감소 될 수 있다. 따라서 본 논문에서 소개한 새로운 프로토콜은 중.대형 시스템 보다는 소형의 시스템에서 원활하게 작동함을 알 수 있다.

7. 결론 및 향후 연구방향

우리는 본 논문에서 블록의 공유 정보를 이용하여 효과적으로 캐쉬 일관성 유지를 위한 새로운 프로토콜을 제안 하였다. 새로운 프로토콜은 기록 무효화 정책의 빈번한 무효화로 인해 캐쉬 실패와 기록 갱신정책의 블록의 공유 여부에 관계없이 갱신된 내용을 상호 연결망을 통하여 방송하는 단점을 개선하였다.

새로 제안된 프로토콜은 처리기가 8,16개인 소형 시스템의 다양한 응용 프로그램의 환경하에서 기록 무효화 정책을 사용하는 기존의 프로토콜보다 좋은 성능을 얻을 수 있음을 시뮬레이션 방법을 통하여 확인하였다.

슬롯링을 상호 연결망으로 하는 공유메모리 다중처리기 시스템의 주요한 단점은 처리기 노드수가 증가함에 따라서 접근 지연시간이 선형적(Linearly)으로 증가하는 것이다. 이러한 문제를 해결하기 위하여 슬롯링을 계층적으로 연결하는 연구가 활발히 진행되고 있으며 Hector Multiprocessor[18]와 KSR1[19]은 그중 한 형태이다. 아울러 이러한 슬롯링의 계층적 구조에서 효율적으로 동작할 수 있는 캐쉬 일관성 정책의 연구가 필요하다.

참 고 문 헌

[1] Per Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessor," IEEE Computer, pp. 12-24, Jun. 1990.

[2] L. A. Barroso and M. Dubois, "Cache Coherence on a Slotted Ring," Intl. Conf. on Parallel Processing, pp.I230-I237, 1991.

[3] L. A. Barroso and M. Dubois, "The Performance of Cache-Coherent Ring-based Multiprossors," Proc. 20th Annul. Intl. Symp. on Computer Architecture, pp.268-277, May, 1993.

[4] SQ. Yang, " Performance Analysis of a Cache Coherent Multiprocessor-based on Hierarchical Multiple Buses," Proc. PARBASE-90, pp. 248 - 257, Mar. 1990.

[5] L. Lamport, "How to Make a Multiprocessor Computer that correctly executes Multiprocess Programs," IEEE Trans. on Computers, Vol C-28, No. 9, pp.690-691, Sept. 1979.

[6] C. Scheurich and M. Dubois, "Correct Memory Operation of Cache-based Multiprocessors," The 14th Intl. Symp. on Computer Architecture, pp. 234 - 243, 1987.

[7] Q. Yang, L. Bhuyan, and B. Liu, "Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," IEEE Trans. on Comput, Vol.38, pp.1143-1154, Aug. 1989.

[8] M. H. MacDogall, "Simulating Computer Systems : Techniques and Tools," The MIT Press, 1987.

[9] C. Thacker and L. Stewart, "Firefly: A Multiprocessor Workstation," Comput. Architecture News, Vol.15, pp.164-172, Oct. 1987

[10] E.McCreight, "The Dragon Computer System : An early overview," Xerox Tech. Rep., Xerox Corp., Sept. 1984

[11] M. Dubois, "Cache Architectures in Tightly Coupled Multiprocessors," IEEE Computer, pp.9-11, June, 1990

[12] M. Dubois and F. Briggs, "Effect of Cache Coherency in Multiprocessors," IEEE Tran. on Computer, No.11, pp.1083-1099, Nov. 1982

[13] M. K. Vernon and E. D. Lazowska, "An Accurate and Efficient Performance Analysis Technique for

Multiprocessor Snooping Cache_Consistency Protcols," In Proc. 15th, Ann. Intl. Symp. Comp. Archi. pp.308-316, 1988.

[14] J. Archibald and J. L. Bear, "Cache Coherence Protocols : Evaluation Using a Multiprocessor Simulation Model," Acm. Trans. Comput. Sys. Vol.4, pp.273-298, Nov. 1986.

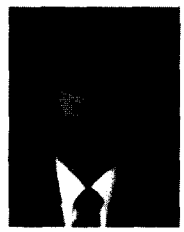
[15] D. Chaiken et al, "Directory-Based Cache Coherence in Large-Scale Multiprocessor," IEEE Computer, pp. 49-57, June. 1990.

[16] Davor Magdic. Limes: A Multiprocessor simulation Environment for PC Platforms (http://galeb.etf.bg.ac.yu/~dav0r/limes)

[17] S. WOO, and J. Singh. The SPLASH2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd Annual Int'l Symp. On Computer Architecture, 43-63, June, 1995.

[18] Z. Vranesic, M. Stumm, D. Lewis and R. White, "Hector : A hierarchically Structured Shared Memory Multiprocessor," IEEE Computer, Vol.24, No.1, pp.72-78, January 1991.

[19] Kendall Square Research, "Technical Summary," Walthan, Massachusetts, 1992.



민 준 식

e-mail : jsmin@nca.or.kr

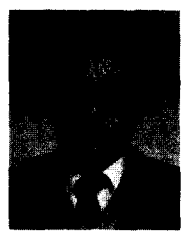
1994년 동국대학교 컴퓨터공학과 졸업(공학석사)

1998년 동국대학교 컴퓨터공학과 박사과정 수료

1994년~1995년 쌍용정보통신 연구원

1996년~현재 한국전산원 주임연구원

관심분야 : 컴퓨터 구조, 병렬처리, 망관리 등



장 태 무

e-mail : jtm@dgu.ac.kr

1977년 서울대학교 전자공학과 졸업(학사)

1979년 한국과학기술원 전산학과 졸업(이학석사)

1995년 서울대학교 컴퓨터공학과 졸업(공학박사)

1979년~1981년 한국전자기술연구소 연구원

1998년 University of Southeastern Lousiana 교환교수

1981년~현재 동국대학교 컴퓨터.멀티미디어공학과 교수

관심분야 : 분산 및 병렬처리, 컴퓨터 구조, 입출력시스템 등