MI-MESI 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜

장 성 태†

요 약

본 논문에서는 분리형 트랜잭션 버스를 기반으로한 다중 프로세서 환경하에서 MESI와 I-MESI 캐쉬 일관성 유지 프로토콜의 문제점을 개선한 MI-MESI 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜을 제시한다. 이 프로토콜에서 각 캐쉬 블럭은 여섯개의 캐쉬 상태 즉, Modified-shared, Invalid-by-other, Modified, Exclusive, Shared 및 Invalid 상태중의 하나를 유지하여, 기존의 MESI와 I-MESI 캐쉬 일관성 유지 프로토콜에서 발생하는 불필요한 메모리 모듈의 갱신과 메모리 모듈에서의 접근 충돌을 크게 줄여서 빠른 메모리 접근 시간을 제공할 수 있다.

MI-MESI Write-invalidate Snooping Cache Coherence Protocol

Seong Tae Jhang†

ABSTRACT

In this paper, we present MI-MESI write-invalidate snooping cache coherence protocol which addresses several significant drawbacks of MESI and I-MESI write-invalidate snooping cache coherence protocols under the split transaction bus based multiprocessor environment. In this protocol, each cache block maintains one of six cache states which represent Modified-shared, Invalid-by-other, Modified, Exclusive, Shared and Invalid states. By using these cache states, our protocol reduces both the access contention and unnecessary updates for the memory modules significantly, and thus providing the fast memory access time.

1. 서 론

공통 버스를 기반으로한 공유 메모리 다중 프로세서 시스템은 다음과 같은 두 가지의 근본적인 문제점들[1, 2]을 내재하고 있다. 즉, 다수의 프로세서 모듈이 동시에 공통 버스에 접근(access)하려고 할 때 발생하는 버스 충돌(bus contention)과 다수의 프로세서 모듈이 하나의 메모리 모듈에 동시에 접근하려고 할 때 발생하는 메모리 충돌(memory contention)이 그것이다. 이러한 문제들은 메모리 접근 시간(memory access time)을 증가시켜서, 결국 다중 프로세서 시스템의 수행 속도(execution speed)를 감소시킨다.

이러한 메모리 모듈들과 시스템 버스에서의 접근 충돌을 줄이기 위해서는 분리형 트랜잭션 버스(split transaction bus)[3, 4, 5]를 시스템 버스로 사용하는 것과 각 프로세서 모듈이 자체 캐쉬(private cache)를 유지하는 것이 제시될 수 있다. 그러나, 프로세서 모듈들 사이에 공유된 블럭들에 대한 캐쉬 일관성(cache coherence)이 유지되어야 하며, 이러한 캐쉬 일관성 문제를 해결하기 위해 여러 캐쉬 일관성 유지 프로토콜들

† 정 회 원:수원대학교 전자계산학과 전임강사
논문접수:1995년 4월 12일, 심사완료:1995년 10월 20일

[6, 7]이 제안되어 왔다. 최근에는 상용화된 버스를 기반으로한 다중 프로세서 시스템에 MESI 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜[8, 9, 10]이 널리 사용되고 있으며, 앞서의 논문을 통해서 분리형 트랜잭션 버스 환경하에서 이러한 MESI 캐쉬 일관성 유지 프로토콜의 단점을 개선한 I-MESI 캐쉬 일관성 유지 프로토콜[5]이 제시되었다.

본 논문에서는 앞서 제시한 I-MESI 캐쉬 프로토콜을 개선한 MI-MESI(Modified-shared, Invalid-by-other, Modified, Exclusive, Shared, Invalid) 캐쉬 일관성 유지 프로토콜을 제시한다. 이 프로토콜은 I-MESI 캐쉬 프로토콜에서 발생하는 불필요한 메모리 갱신을 방지하는 동시에 프로세서 모듈들 사이의 캐쉬-대-캐쉬 통신을 더욱 잘 활용하여 분리형 트랜잭션 버스 환경하에서 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 더욱 줄일 수 있다.

본 논문의 2장에서는 분리형 트랜잭션 버스 환경하에서 기존의 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜의 문제점과, 이러한 문제점을 해결하기 위해 제시된 I-MESI 캐쉬 프로토콜에 대해 간략히 기술한다. 3장에서는 MI-MESI 캐쉬 프로토콜의 기본 개념과 동작을 기술하고, 앞서 제시했던 MMESSII[11] 및 I-MESI 캐쉬 프로토콜과의 비교를 수행하며, MI-MESI 캐쉬 프로토콜이 순차적 일관성(sequential consistency)을 유지함을 보인다. 4장에서는 시뮬레이션을 위해 가정한 시스템의 구조와 사용된 작업 부하 모델 및 시뮬레이션을 위해 구현한 캐쉬 시뮬레이터에 대해 기술하며, 제시한 MI-MESI 캐쉬 프로토콜과 기존의 MESI 및 I-MESI 캐쉬 프로토콜의 시뮬레이션 결과를 제시한다. 5장에서는 결론을 맺는다.

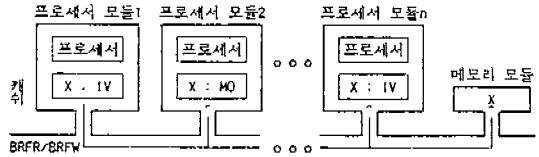
2. 기존 쓰기-프로토콜들에 대한 고찰

2.1 기존 쓰기-프로토콜의 문제점

기존의 쓰기-무효화 스누핑 방식으로 동작하는 캐쉬 프로토콜들은 한 프로세서 모듈이 공유된 상태(SH)로 자체 캐쉬에 저장된 블럭을 갱신하려고 할 때, 먼저 다른 프로세서 모듈들 내의 캐쉬들에 공유된 상태로 저장된 해당 블럭들을 모두 무효화된 상태(IV)로 변환시킨 후에 자체 캐쉬내에 저장된 그 블럭을 갱신된 상태(MO)로 변환하는 동시에 그 블럭을 갱신함으로써 공유된 블럭들에 대한 캐쉬 일관성을 유지한다.

따라서, 쓰기-무효화 스누핑 방식으로 동작하는 캐쉬 프로토콜들에서는 한 프로세서가 무효화된 상태로 자체 캐쉬에 저장된 블럭에 대해 읽기/쓰기 요청을 했을 경우에 "무효화 캐쉬 실패(invalidation cache miss)"가 발생하며, 이러한 "무효화 캐쉬 실패"는 프로세서 모듈의 수가 증가하고, 각 프로세서 모듈내의 캐쉬와 캐쉬 블럭의 크기가 증가함에 따라 캐쉬 실패의 상당 부분[12. 13]을 차지한다고 알려져 있다. 그러나, MESI 캐쉬 프로토콜을 포함한 기존의 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜에서 각 프로세서 모듈은 "무효화 캐쉬 실패"에 대해 그 블럭이 다른 프로세서 모듈내에 갱신된 상태로 저장되어 있는지 여부를 알 수 없기 때문에 (그림 1)에서와 같이 읽기 실패의 경우에는 그 블럭에 대한 브로드캐스트 읽기를 위한 읽기(BRFR:Broadcast Read For Read) 요청, 쓰기 실패의 경우에는 브로드 캐스트 쓰기를 위한 읽기(BRFW:Broadcast Read For Write) 요청을 다른 프로세서 모듈들과 메모리 모듈로 전송한다.

그러나, "무효화 캐쉬 실패"가 발생할 경우에 생성되는 이러한 BRFR 요청이나 BRFW 요청에 대한 응답은 요청된 블럭을 갱신된 상태로 저장하고 있는 프로세서 모듈이 그 블럭을 메모리 모듈로 축출한 경우를 제외하고는 항상 그 블럭을 갱신하여 저장하고 있는 프로세서 모듈, 즉 (그림 1)의 경우 프로세서 모듈 2에 의해 제공된다. 따라서, "무효화 캐쉬 실패"에 의한



(그림 1) MESI 프로토콜의 예
(Fig. 1) The example of MESI protocol

BRFR나 BRFW 요청은 메모리 모듈로 하여금 궁극적으로 다른 프로세서 모듈에 의해 제공될 블럭에 대한 메모리 읽기 동작을 수행하도록 하기 때문에, 분리형 트랜잭션 버스를 기반으로한 다중 프로세서 시스템 환경하에서 메모리 접근 충돌과 실질적인 메모리 접근 시간(effective memory access time)을 증가시켜서 시스템의 성능을 크게 저하시키는 요인으로 작용한다[5, 11].

2.2 I-MESI 캐쉬 일관성 유지 프로토콜

I-MESI 캐쉬 프로토콜의 기본 개념은 특정 블럭에 대한 "무효화 캐쉬 실패"가 발생하고, 다른 프로세서 모듈이 그 블럭을 Modified 상태로 자체 캐쉬에 저장하고 있으면, BRFR나 BRFW 요청 대신에 앞서의 MMESSII 캐쉬 프로토콜을 통해 제시했던 프로세서 모듈들 사이의 캐쉬-대-캐쉬(cache-to-cache) 통신을 사용하여 MESI 캐쉬 프로토콜보다 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 줄이는데 있다. 또한, MMESSII 캐쉬 프로토콜에서 필요로 했던 각 캐쉬 블럭당 유지해야 하는 ID 영역과 같은 특별한 하드웨어의 지원 없이도 기존의 다중 프로세서 시스템에 쉽게 이식되어 사용될 수 있도록 하는데 있다.

이를 위해 I-MESI 캐쉬 프로토콜은 각 캐쉬 블럭을 위해 MESI 캐쉬 프로토콜과 동일한 의미를 갖는 Modified 상태와 Exclusive 상태 및 Shared 상태 외에 2개의 무효화된 캐쉬 상태 즉 Invalid-by-other 상태와 Invalid 상태를 유지하며, 캐쉬 상태 전이를 위해 shared 신호선과 dirty 신호선을 필요로 한다. Invalid 상태는 해당 블럭을 자체 캐쉬내에 Modified 상태로 저장하고 있는 프로세서 모듈이 없음을 의미하는 반면에, Invalid-by-other 상태는 다른 프로세서 모듈이 그 블럭을 Modified 상태로 저장하고 있음을 의미한다. 따라서, I-MESI 캐쉬 프로토콜에서는 "무효화 캐쉬 실패"가 발생했을 경우에 해당 블럭이 다른 프로세서 모듈내에 Modified 상태로 저장되어 있는지 여부를 쉽게 알 수 있으며, 읽거나 쓰기 요청된 블럭의 상태가 Invalid-by-other의 경우에는 메모리 모듈이 참여할 필요가 없는 캐쉬-대-캐쉬 읽기를 위한 읽기(CRFR:Cache-to-cache Read For Read) 요청이나 캐쉬-대-캐쉬 쓰기를 위한 읽기(CRFW:Cache-to-cache Read For Write) 요청을 전송한다. 하지만, 4개의 상태를 유지하기 위해 각 캐쉬 블럭당 2 비트의 상태 비트를 필요로 하던 MESI 캐쉬 프로토콜에 비해 I-MESI 캐쉬 프로토콜은 5개의 상태를 유지하기 위해 각 캐쉬 블럭당 3 비트의 상태 비트가 필요하고, 앞에서 기술한 것처럼 CRFR나 CRFW의 새로운 버스 트랜잭션이 필요하다.

〈표 1〉은 I-MESI 캐쉬 프로토콜의 동작을 간략히 기술한 상태 전이표로서, MO는 Modified 상태, EX는 Exclusive 상태, SH는 Shared 상태, IV는 Invalid상태, IO는 Invalid-by-other 상태, sl은 shared 신호선, dl은 dirty 신호선, NULL은 해당 블럭이 저장되어 있지 않음을 의미하며, N/C는 캐쉬 상태에 변화가 없음을 의미한다.

다음의 (그림 2)는 Invalid-by-other 상태의 블럭에 "무효화 캐쉬 실패"가 발생한 경우에 대한 I-MESI 캐쉬 프로토콜의 동작을 설명하는 예이다.

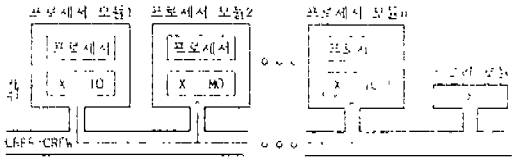
3. MI-MESI 캐쉬 일관성 유지 프로토콜

3.1 MI-MESI 캐쉬 일관성 유지 프로토콜

I-MESI 캐쉬 프로토콜은 읽기/쓰기 요청된 블

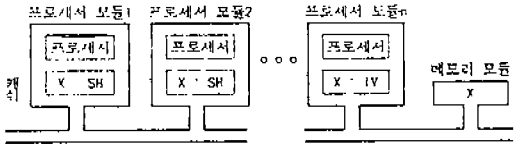
〈표 1〉 I-MESI 캐쉬 프로토콜의 상태 전이표
〈Table 1〉 State transition table of I-MESI cache protocol

조 건	요구한 프로세서 모듈			다른 프로세서 모듈			메모리 모듈
	동 작	블럭상태		동 작	블럭상태		
		FROM	TO		FROM	TO	
블럭축출	MM으로 축출	MO	NULL	상태 변환	IO	IV	메모리 갱신
읽기적중	읽기 수행	MO,SH,EX	N/C	.....	...	...	.....
읽기실패	BRFR요청전송	IV,NULL	EX	.....	...	...	블럭제공
		IV,NULL	SH	블럭제공,dl=1	MO	SH	dl=0이면
				상태변화	IO	IV	블럭제공
				상태변화,sl=1	EX,SH	SH	
	CRFR요청전송	IO	SH	블럭제공	MO	SH	.....
				상태변환	IO	IV	
쓰기적중	쓰기수행	MO,EX	MO	.....	...	...	.....
	무효화요청전송	SH	MO	무효화 수행	SH,IV	IO	.....
쓰기실패	BRFW요청전송	IV,NULL	MO	블럭제공,dl=1	MO	IO	dl=0 이면
				상태변환	IV	IO	블럭제공
				무효화수행	EX,SH	IO	
	CRFW요청전송	IO	MO	블럭제공및무효화수행	MO	IO	.....



(그림 2) I-MESI 프로토콜의 예 1
(Fig. 2) The example 1 of I-MESI protocol

력이 Invalid-by-other 상태일 때 메모리 모듈이 참여할 필요가 없는 CRFR나 CRFW 요청을 전송하여 MESI 캐쉬 프로토콜에 비해 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 줄일 수 있었다. 하지만, 한 프로세서 모듈이 어떤 블럭 X에 대한 CRFR 요청을 전송했을 경우에, 블럭 X를 Modified 상태로 저장하고 있는 프로세서 모듈이 그 블럭을 제공하고 그 블럭의 상태를 Shared 상태로 변환한다. 이때, 공유 메모리 모듈도 항상 해당 블럭을 항상 갱신한다. 따라서, 블럭 X를 Invalid-by-other 상태로 저장하고 있는 모든 다른 프로세서 모듈은 전송된 CRFR 요청에 대해 블럭 X의 상태를 Invalid 상태로 변환한다. (그림 3)은 (그림 2)에서 프로세서 모듈 2가 요청된 블럭 X를 전송한 후의 I-MESI 캐쉬 프로토콜을 구동하는 시스템의 상태를 예시한다.



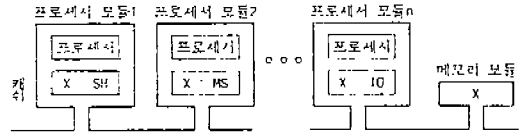
(그림 3) I-MESI 프로토콜의 예 2
(Fig. 3) The example 2 of I-MESI protocol

이와 같이, I-MESI 캐쉬 프로토콜에서도 MESI 캐쉬 프로토콜에서와 같이 불필요한 메모리 모듈 갱신이 자주 발생할 뿐만 아니라, 블럭 X의 상태를 Invalid-by-other에서 Invalid 상태로 변환한 다른 프로세서 모듈이 다시 블럭 X에 접근하려할 때 BRFR나 BRFW 요청을 사용할 수 밖에 없으며, 이로 인해 메모리 모듈에서의 접근 충돌이 여전히 많은 빈도수로 발생할 수 있다.

MI-MESI 캐쉬 프로토콜의 중심 개념은 I-MESI 캐쉬 프로토콜에서 발생하는 이러한 불필요한 메모리 갱신을 줄이고, MMESSII 캐쉬 프로토콜을 통해 제시한 메모리 모듈이 참여할 필요가 없는 캐쉬-대-캐쉬 통신 개념을 더욱 효율적으로 활용하여 분리형 트랜잭션 버스 환경하에서 메모리 모듈에서의 접근 충돌과 접근 지연 시간을 줄이는 데 있다.

이를 위해 MI-MESI 캐쉬 프로토콜에서는 각 캐쉬 블럭을 위해 I-MESI 캐쉬 프로토콜에서 사용한 5개의 캐쉬 상태 외에 Modified-shared 상태를 하나 더 유지한다. Modified-shared 상태는 Modified 상태처럼 해당 블럭이 자체 캐쉬내에서 갱신되었으나 메모리 모듈내에 저장된 그 블럭은 아직 갱신되지 않았음을 의미하며, Modified 상태와는 달리 다른 프로세서 모듈이 그 블럭을 Shared 상태로 저장하고 있을 수 있음을 의미한다. 이러한 Modified-shared 상태의 정의에 따라 Invalid-by-other 상태는 I-MESI 캐쉬 프로토콜에서와는 달리 해당 블럭을 Modified 상태나 Modified-shared 상태로 저장하고 있는 프로세서 모듈이 존재함을 의미하며, Shared 상태는 그 블럭을 다른 프로세서 모듈이 Shared 상태나 Modified-shared 상태로 저장하고 있을 수 있음을 의미한다.

앞의 예에 대해 MI-MESI 캐쉬 프로토콜에서는 블럭 X를 Modified 상태로 저장하고 있는 프로세서 모듈이 캐쉬 실패가 발생한 프로세서 모듈에게 블럭 X를 제공하는 동시에 블럭 X의 자체 캐쉬내에서의 상태를 Modified-shared 상태로 변환하지만, 공유 메모리 모듈은 갱신하지 않는다. 따라서, 블럭 X를 Invalid-by-other 상태로 저장하고 있는 모든 다른 프로세서 모듈들은 블럭 X의 상태를 변환할 필요가 없으며, 블럭 X에 대해 또다른 캐쉬 실패가 계속해서 발생했을 경우에 그 프로세서 모듈들은 CRFR나 CRFW 요청을 사용하여 블럭 X를 Modified-shared 상태로 저장하고 있는 프로세서 모듈로부터 요청한 블럭을 제공받기 때문에 I-



(그림 4) MI-MESI 프로토콜의 예
(Fig. 4) The example of MI-MESI protocol

MESI 캐쉬 프로토콜에서 발생하는 불필요한 메모리 접근을 방지할 수 있다. (그림 4)는 (그림 2)에서 프로세서 모듈 2가 요청된 블럭 X를 전송한 후의 MI-MESI 캐쉬 프로토콜을 구동하는 시스템의 상태를 예시한다.

이제 본 논문을 통해 제시하려는 MI-MESI 캐쉬 프로토콜의 동작을 블럭 대체, 읽기 적중, 읽기 실패, 쓰기 적중, 쓰기 실패의 관점에서 좀더 자세히 기술하면 다음과 같다,

○ 블럭 대체:블럭 대체를 위해 선정된 블럭이 Modified 상태나 Modified-Shared 상태일 경우에 그 블럭을 해당 메모리 모듈로 축출한다. 이때 축출되는 블럭을 Invalid-by-other 상태로 저장하고 있는 프로세서 모듈들은 그 블럭의 캐쉬 상태를 Invalid로 변환한다.

○ 읽기 적중:읽기 요청된 블럭의 캐쉬 상태가 Modified, Modified-shared, Shared 또는 Exclusive인 경우로서 프로세서는 상태 변화없이 즉시 자체 캐쉬로부터 해당 읽기 동작을 수행한다.

○ 읽기 실패:읽기 요청된 블럭이 자체 캐쉬내에 저장되어 있지 않거나 Invalid 상태로 저장되어 있는 경우에 읽기 실패가 발생한 프로세서 모듈은 시스템 버스를 통해 그 블럭에 대한 BRFR 요청을 전송하며, 그 요청에 대해 해당 블럭을 Exclusive 상태나 Shared 상태로 저장하고 있는 프로세서 모듈은 shared 신호선에 논리적 1의 값을 인가한다. 해당 블럭을 Modified-shared 상태나 Modified 상태로 저장하고 있는 프로세서 모듈은 dirty 신호선에 논리적 1의 값을 인가하여 자신이 해당 블럭을 제공할 것임을 알리는 동시에 요청된 블럭을 준비하여 그 블럭을 제공하며, 그 블럭의 자체 캐쉬내에서의 상태를 Modified-shared로 유지/변환한다. 메모리 모듈은 dirty 신호선을 조사하여 요청된 블럭을 자신이 제공할 지 여부를 결정한다. 읽기 요청을 전송한 프로세서 모듈은 shared 신호선이나 dirty 신호선에 논리적 1의 값이 인가되었을 경우에 제공받을 블럭을 Shared 상태로, 그 이외의 경우에는 Exclusive 상태로 자체 캐쉬에 저장하고 해당 읽기 동작을 수행한다.

읽기 요청된 블럭이 자체 캐쉬내에 Invalid-by-other 상태로 저장되어 있을 경우에 프로세서 모듈은 메모리 모듈이 참여할 필요가 없는 그 블럭에 대한 CRFR 요청을 전송한다. 이때, 그 블럭을 Modified-shared 상태나 Modified 상태로 저장하고 있는 프로세서 모듈이 요청된 블럭을 제공하는 동시에 그 블럭의 자체 캐쉬내에서의 상태를 Modified-shared 상태로 유지/변환한다. 요청한 블럭을 제공받은 프로세서 모듈은 그 블럭을 Shared 상태로 자체 캐쉬에 저장하고 해당 읽기 동작을 수행한다.

○ 쓰기 적중:쓰기 요청된 블럭의 캐쉬 상태가 Modified나 Modified-shared, Shared 또는 Exclusive인 경우로서, 해당 블럭이 Modified 상태나 Exclusive 상태일 경우에 프로세서 모듈은 그 블럭의 상태를 Modified로 유지/변환하고 쓰기를 수행한다. 그 이외의 경우에 프로세서 모듈은 그 블럭에 대한 무효화 요청을 시스템 버스를 통해 전송하여 무효화를 수행한 후에 그 블럭을 Modified 상태로 변환하며, 그 블럭에 대한 쓰기 동작을 수행한다. 이때 해당 블럭을 공유하고 있는 다른 모든 프로세서 모듈들은 그 블럭의 자체 캐쉬내에서의 상태를 Invalid-by-other로 변환한다.

○ 쓰기 실패:쓰기 요청된 블럭이 자체 캐쉬내에 저장되어 있지 않거나 Invalid 상태로 저장되어 있는 경우에 쓰기 실패가 발생한 프로세서 모듈은 시스템 버스를 통해 해당 블럭에 대한 BRFW 요청을 전송하며, 그 요청에 대해 해당 블럭을 Modified-shared 상태나 Modified 상태로 저장하고 있는 프로세서 모듈은 dirty 신호선에 논리적 1의 값을 인가하며, 요청된 블럭을 준비하여 그 블럭을 제공하는 동시에 자체 캐쉬내의 그 블럭의 상태를 Invalid-by-other로 변환한다. 메모리 모듈은 dirty 신호선을 조사하여 요청된 블럭을 자신이 제공할 지 여부를 결정하며, 해당 블럭을 공유하고 있는 다른 모든 프로세서 모듈들은 그 블럭의 자체 캐쉬내에서의 상태를 Invalid-by-other로 변환한다. 요청한 블럭을 제공받은 프로세서 모듈은 그 블럭을 Modified 상태로 자체 캐쉬에 저장하고, 해당 쓰기 동작을 수행한다.

쓰기 요청된 블럭이 자체 캐쉬내에 Invalid-by-other 상태로 저장되어 있을 경우에 프로세서 모듈은 메모리 모듈이 참여할 필요가 없는 그 블럭에 대한 CRFW 요청을 전송한다. 이때, 그 블럭을 Modified-shared 상태나 Modified 상태로 저장하

고 있는 프로세서 모듈이 요청된 블럭을 제공하는 동시에 그 블럭의 자체 캐쉬내에서의 상태를 Invalid-by-other로 변환한다. 요청한 블럭을 제공받은 프로세서 모듈은 그 블럭을 Modified 상태로 자체 캐쉬에 저장하여 쓰기 동작을 수행하며, 그 블럭을 공유하고 있는 다른 프로세서 모듈들도 자체 캐쉬 내에 저장된 그 블럭을 Invalid-by-other 상태로 변환한다.

다음의 〈표 2〉는 앞에서 기술한 MI-MESI 캐쉬 프로토콜의 상태 전이표이다.

〈표 2〉 MI-MESI 캐쉬 프로토콜의 상태 전이표
〈Table 2〉 State transition table of MI-MESI cache protocol

조 건	요구한 프로세서 모듈			다른 프로세서 모듈들			메모리 모듈
	동 작	블럭상태		동 작	블럭상태		
		FROM	TO		FROM	TO	
블럭옮출	MM으로 축출	MO,MS	NULL	.....	...	...	업데이트 갱신
읽기적중	읽기 수행	MO,MS, SH,EX	N,C	.....	...	...	.....
읽기실패	BRFR요청전송	IV,NULL	EX	.....	...	...	d1=0이면 블럭제공
		IV,NULL	SH	블럭제공,d1=1	MO,MS	MS	
		IV,NULL	SH	상태변화,d1=1	EX,SH	SH	
	CRFR요청전송	IO	SH	블럭제공	MO,MS	MS	.....
쓰기적중	쓰기수행	MO,EX	MO	.....	...	...	.....
	무효화요청전송	MS,SH	MO	무효화 수행	MS,SH,IV	IO	
쓰기실패	BRFW요청전송	IV,NULL	MO	블럭제공,d1=1 무효화수행	MS,MO EX,SH,IV	IO IO	d1=0 이면 블럭제공
	CRFW요청전송	IO	MO	블럭제공무효화수행	MS,MO,EX	IO	.....

다음의 〈표 3〉은 본 논문에서 제시한 MI-MESI 캐쉬 프로토콜과 MMESSII 및 I-MESI캐쉬 프로토콜과의 차이점에 대해 요약한 것이다.

〈표 3〉 MMESSII, I-MESI 캐쉬 프로토콜과의 비교
〈Table 3〉 Comparision among MI-MESI, MMESSII and I-MESI cache protocols

	MMESSII	I-MESI	MI-MESI
캐쉬상태	MO, MS, SS, SH, EX, IV, IO의 7개 상태유지	MO, MS, SS, SH, EX, IV, IO의 6개 상태유지	MO, MS, SS, SH, EX, IV, IO의 6개 상태유지
블럭당 상태 비트수	7개의 상태를 위한 3비트 -ID 영역을 위한 log2(프로세서 모듈수) 비트	6개의 상태를 위한 3비트	6개의 상태를 위한 3비트
확장성	ID 영역으로 인해 제한	제한 없음	제한 없음
기 타	Memory-Inhibit 신호선필요	불필요	불필요

3.2 MI-MESI 캐쉬 일관성 유지 프로토콜의 검증

다음 조건을 만족하는 어떠한 시스템도 순차적 일관성을 갖는다[14]라고 알려져 있다.

(조건 1) 각 프로세서는 프로그램이 명시하는 순서로 메모리 접근 요청을 생성(issue)한다.

(조건 2) STORE 즉 쓰기 동작을 생성한 프로세서는 그 쓰여진 값이 모든 다른 프로세서에 의해 접근 가능하게 될 때까지는 또다른 메모리 접근 요청을 생성하지 않는다.

(조건 3) LOAD 즉 읽기 동작을 생성한 프로세서는 그 읽혀진 값이 모든 다른 프로세서에 의해 접근 가능하게 될 때까지는 또다른 메모리 접근 요청을 생성하지 않는다.

분리형 트랜잭션 버스 환경하에서는 특정 블럭에 대한 접근 요청을 전송한 프로세서 모듈이 해당 블럭을 메모리 모듈이나 다른 프로세서 모듈로부터 제공받기 전에 또다른 프로세서 모듈이 시스템 버스를 통해 그 블럭에 대한 접근이나 무효화 요청을 전송할 수 있다. 따라서, 비 분리형 트랜잭션 버스 환경하에서 순차적 일관성을 갖는 캐쉬 프로토콜들도 분리형 트랜잭션 버스 환경하에서는 순차적 일관성이 유지되지 않는 경우가 발생할 수 있다[5,11].

MI-MESI 캐쉬 프로토콜은 이러한 문제를 해결하기 위해 busy 신호선을 사용한다. 즉 시스템 버스를 통해 특정 블럭에 대한 접근 요청을 전송하였으나 아직 요청한 블럭을 제공받지 못한 프로세서 모듈들은 응답을 기다리고 있는 해당 블럭의 주소와 시스템 버스를 통해 전송되는 접근 또는 무효화 요청의 주소와의 비교를 수행하며, 만일 두 주소가 일치하면 논리적 1의 값을 busy 신호선에 인가한다. 이때 해당 접근 또는 무효화 요청을 전송받은 모든 다른 모듈들은 그 요청을 무시하며, 그 요청을 전송한 프로세서 모듈은 버스 중재 과정을 거쳐 그 요청을 재 전송한다. 따라서, MI-MESI 캐쉬 프로토콜 환경하에서 특정 블럭에 대해 접근 요청을 전송한 프로세서 모듈이 해당 블럭을 제공받기 전까지 다른 프로세서 모듈들로 부터의 그 블럭에 대한 접근 또는 무효화 요청은 처리될 수 없다. 그러므로, MI-MESI 캐쉬 프로토콜은 앞에서 기술한 3 가지 조건을 다음과 같이 만족한다.

첫째, 각 프로세서가 프로그램 순서대로 LOAD와 STORE 요청을 생성하면 MI-MESI 캐쉬 프로토콜은 (조건 1)을 만족한다.

둘째, 프로세서로부터의 쓰기 요청에 대해, 프로세서 모듈은 Shared나 Modified-shared 상태의 블럭에 쓰기 적중인 경우에 먼저 다른 프로세서 모듈들내에 저장되어 있는 그 블럭들을 모두 무효화시킨 후에 해당 블럭에 대한 쓰기 동작을 수행하며, Exclusive 상태나 Modified 상태의 블럭에 쓰기 적중인 경우에 그 블럭을 유일하게 유효한 상태로 저장하고 있기 때문에 쓰기 동작을 즉시 수행한다. 그 이외의 경우에 프로세서 모듈은 다른 프로세서 모듈들내에 유효한 상태로 저장되어 있는 그 블럭을 모두 무효화시킨 후에 다른 프로세서 모듈이나 메모리 모듈로부터 가장 최근에 갱신된 블럭을 제공받아 자체 캐쉬내에 Modified 상태로 저장하고 그 블럭에 대한 쓰기 동작을 수행한다. 그러므로, MI-MESI 캐쉬 프로토콜에서 한 프로세서 모듈이 특정 블럭에 대한 쓰기를 수행할 때, 다른 프로세서 모듈들내의 캐쉬들에 그 블럭이 유효한 상태로 저장되어 있지 않으며, 다른 프로세서 모듈들은 쓰기가 수행된 프로세서 모듈로부터 그 블럭을 제공받는다. 따라서, MI-MESI 캐쉬 프로토콜은(조건 2)를 만족한다.

셋째, MI-MESI 캐쉬 프로토콜에서는 임의의 순간에 하나의 공유된 블럭에 대해 단지 하나의 프로세서 모듈로부터의 쓰기만이 수행될 수 있다. 따라서, 특정 블럭에 대해 접근 요청을 생성한 프로세서들은 모든 다른 프로세서 모듈들에게 접근 가능한 해당 블럭에 대한 읽기 동작을 수행하거나 그 읽기 동작의 수행을 해당 블럭이 갱신되어 모든 다른 프로세서 모듈들에게 접근 가능하게 될 때까지 기다려야만 한다. 그러므로, MI-MESI 캐쉬 프로토콜은 (조건 3)을 만족한다.

4. 시뮬레이션

4.1 작업 부하 모델

시뮬레이션에 사용된 작업 부하(workload)는 각 프로세서에 의한 S 블럭(공유된 블럭)들에 대한 메모리 접근 요청들과 P 블럭(공유되지 않은 블럭)들에 대한 메모리 접근 요청들로 구성된다. S 블럭들에 대한 작업 부하 모델로 참조 국부성(reference locality)을 잘 반영할 수 있는 LRU 스택 모델(Least Recently Used Stack Model)[7, 15]을 사용하였으며, P 블럭들에 대한 작업부하 모델로 캐쉬 적중과 캐쉬 실패 등에 대한 고정된 확률 분포[7, 15]를 사용하였으며, 캐쉬 프로토콜에 무관한 시스템 파라미터들은 모두 동일하다고 가정하고 시뮬레이션을 수행하였다. 시뮬레이션에 사용된 파라미터들과 그 값의 범위는 다음과 같으며, 기존의 논문들[7, 11, 17]에서 보편적으로 사용되고 있는 값들에 근거해서 채택하였다.

◇ 메모리 접근 요청을 생성할 확률(acc):0.3
◇ S블럭들에 대한 메모리 접근 요청 확률(shd):0.05-0.1
◇ 메모리 접근 요청이 읽기 요청일 확률(rd):0.7-0.9
◇ P블럭들에 캐쉬 적중일 확률(h):0.96
◇ 쓰기 적중인 P블럭이 Modified 상태일 확률:0.96
◇ 축출될 P 블럭이 갱신되어 있을 확률:0.35
◇ S블럭들의 총 수:500개
◇ 블럭의 크기:4 워드
◇ 캐쉬의 크기:128Kbytes
◇ 프로세서 모듈의 수(Npm):2-20개
◇ 메모리 모듈의 수(Nm):2 개
◇ 메모리 모듈당 입력 버퍼의 수:1 개
◇ 버스 싸이클 시간:3 프로세서 싸이클
◇ 버스 중재 시간:1 버스 싸이클
◇ 요구의 전송:1 버스 싸이클
◇ 응답의 전송:1 버스 싸이클
◇ 스누핑에 걸리는 시간:1 버스 싸이클
◇ 스누핑 결과 전송:1 버스 싸이클
◇ 메모리 모듈 접근 시간:4 버스 싸이클
◇ 프로세서 모듈 접근 시간:3 버스 싸이클

4.2 버스 모델

본 논문에서는 앞선 메모리 접근 요청의 실행과 현재의 메모리 접근 요청이나 응답의 전송 및 다음 시스템 버스의 사용자를 결정하는 버스 중재가 동시에 진행될 수 있는 동기화(synchronous) 분리형 트랜잭션 버스 모델 환경하에서 시뮬레이션을 수행하였다. 사용된 버스 모델은

TICOM 다중 프로세서 시스템[8]과 Multimax 다중 프로세서 시스템[10]에 사용된 분리형 트랜잭션 버스와 유사하게 동작하며, 어드레스 버스에 대한 중재와 데이타 버스에 대한 중재가 분리되어 있어서 어드레스 버스와 데이타 버스가 프로세서 모듈들이나 메모리 모듈에 의해 각각 독립적으로 사용될 수 있다고 가정하였다.

4.3 캐쉬 시뮬레이터

캐쉬 프로토콜들 사이의 성능 비교를 위해 개발된 캐쉬 시뮬레이터는 SLAM II 사건 구동 시뮬레이션 도구[16]를 사용하여 구현하였으며, 프로세서와 캐쉬 제어기, 스누프 제어기, 버스 중재기, 시스템 버스 및 메모리 모듈 등과 같은 주요 구성 요소들의 동작들과 기능들을 시뮬레이션하는 사건(event)들로 구성된다. 시뮬레이터의 주 프로그램에서는 시스템 파라미터들을 초기화한 후 제어를 SLAM II 사건 구동 시뮬레이션 도구의 SLAM 프로세스로 이양하며, SLAM 프로세스는 INTLC 사건을 스케줄한다. 이때부터 SLAM 프로세스는 예약된 시간 순서대로 다음과 같은 사건들을 스케줄링하여 시뮬레이션을 수행한다.

○**INTLC 사건**:이 사건은 프로세서 모듈과 메모리 모듈의 수를 결정하며, 각 프로세서 모듈마다 할당되는 LRU 스택과 해당 프로세서의 활용도(utilization)를 측정하기 위해 사용되는 시스템 플래그들을 초기화한다. 또한, 프로세서 수만큼의 CPU 사건과 메모리 모듈 수만큼의 MEMMOD 사건 및 BUSARB 사건들이 time 0에서 스케줄되도록 동작한다.

○**CPU 사건**:각 프로세서의 동작을 시뮬레이션하는 이 사건은 acc의 확률로 메모리 접근 요청을 생성하여 다음 프로세서 클럭에 CCTRL 사건이 스케줄 되도록 동작한다.

○**CCTRL(Cache Controller)사건**:캐쉬 제어기 동작을 시뮬레이션하는 이 사건은 CPU 사건과 SYSBUS 사건에 의해서 구동된다.

○**ARBREQ(Arbitration Request) 사건**:프로세서 모듈이나b 메모리 모듈내에 있는 버스 중재기에 의한 버스 중재 요구 동작을 시뮬레이션하는 이 사건은 CCTRL 사건이나 SCTRL 사건 및 MEMMOD 사건에 의해 구동될 수 있으며, 전송된 메시지를 BUSARB 사건의 입력 큐에 저장한다.

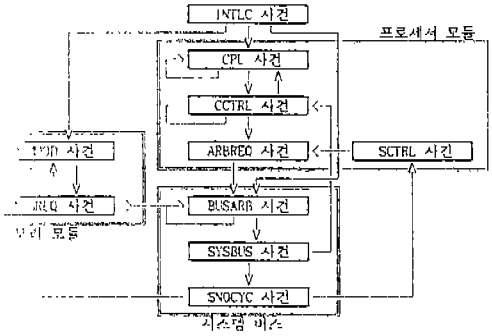
○**BUSARB(Bus Arbitration)사건**:이 사건은 입력 큐를 이용하여 FIFO 방식으로 동작하는 시스템 버스 중재기를 시뮬레이션 한다. 이를 위해 이 사건은 다음 버스 클럭에 BUSARB 사건이 다시 스케줄 되도록 동작하며, 입력 큐가 비어있지 않을 경우에는 입력 큐의 메시지 내용을 읽어 다음 버스 클럭에 SYSBUS 사건이 스케줄 되도록 동작한다.

○**SYSBUS(System Bus)사건**:이 사건은 BUSARB 사건에 의해 구동되며, 시스템 버스를 통한 여러 요청의 전송을 시뮬레이션한다. 이 사건은 BUSARB 사건이 제공한 메시지를 해석하여 그 내용이 앞선 접근 요청에 대한 응답일 경우에는 다음 버스 클럭에 해당 프로세서 모듈의 CCTRL 사건이 스케줄 되도록 동작하며, 그 이외의 경우에는 SNOCYC 사건이 스케줄 되도록 동작한다.

○**SNOCYC(Snooping Cycle)사건**:이 사건은 SYSBUS 사건에 의해 구동되며, 전송된 무효화 요청 및 접근 요청에 대한 스누핑을 시뮬레이션한다. 이 사건은 전달된 메시지의 내용을 해석하여 요청의 종류에 따라 다른 프로세서 모듈들내에 저장되어 있는 해당 S 블럭의 캐쉬 상태들을 갱신하며, 요청된 블럭을 제공할 프로세서 모듈이 존재할 경우에는 다음 버스 클럭에 그 프로세서 모듈의 SCTRL 사건이 스케줄 되도록 동작한다. 이 사건은 또한 요청의 종류에 따라 해당 메시지를 MEMMOD 사건의 입력 큐에 저장하여 MEMMOD 사건이 그 요청을 처리할 수 있게 동작한다.

○**SCTRL(Snoop Controller)사건**:이 사건은 각 프로세서 모듈내의 스누프 제어기의 동작을 시뮬레이션하며, SNOCYC 사건에 의해 구동된다. 이 사건은 전송된 S 블럭에 대한 접근 요청에 대해 자체 캐쉬내의 해당 S 블럭의 캐쉬 상태를 갱신한다. 데이타 캐쉬로부터 요청된 블럭을 읽는데 걸리는 프로세서 클럭이 경과한 후에 이 사건은 응답을 위한 메시지를 구성하여 ARBREQ 사건이 스케줄 되도록 동작한다.

:MMOD(Memory Module)사건:메모리
동작을 시뮬레이션하는 이 사건은 입력
비어 있을 경우에 다음 버스 클럭에 ME-
) 사건이 다시 스케줄 되도록 동작한다.
큐가 비어있지 않을 경우에는 입력 큐에 저
메시지의 내용에 따라 해당 블럭을 메모리
거나 메모리로부터 읽는데 걸리는 시간이
이 후에 MEMMOD 사건이 스케줄 되도록
하며, 응답의 전송을 위해 BUSARB 사건도
줄 되도록 동작한다.
림 5)는 앞에서 기술한 캐쉬 시뮬레이터의
설명한다.



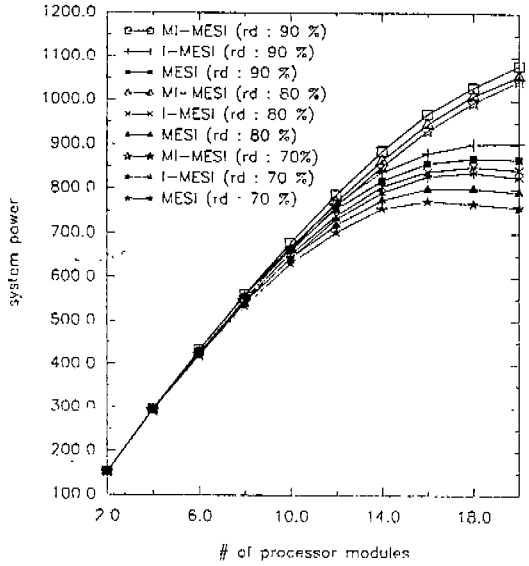
(그림 5) 캐쉬 시뮬레이터의 구조
(Fig. 5) Structure of cache simulator

4.4 시뮬레이션 결과

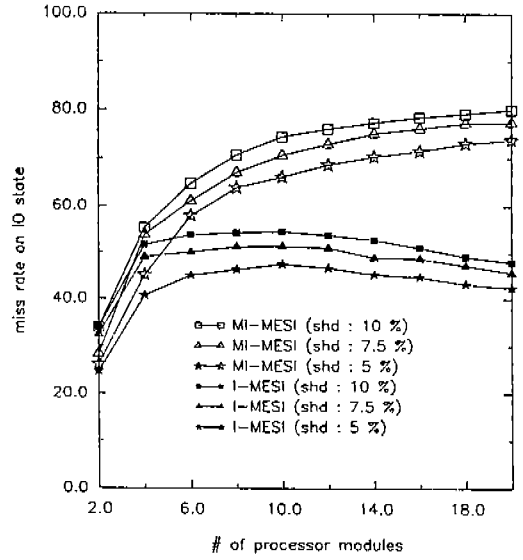
(그림 6)은 공유된 블럭의 비율에 따른 MI-MESI 캐쉬 프로토콜과 MESI 및 I-MESI 캐쉬 프로토콜의 시스템 성능을 비교하기 위해 읽기 요청을 생성할 비율인 rd의 값을 0.8로 고정하고 S 블럭에 메모리 접근 요청을 생성할 확률인 shd를 5%에서 10%까지 2.5%씩 증가시켜가며 시뮬레이션을 수행한 결과로서 시뮬레이션 결과에 표시된 "시스템 성능(system power)"은 시뮬레이션에 사용된 프로세서들(Pi)의 활용도 퍼센트(utilization percentage)들의 합으로 다음의 수식으로 표현된다. (그림 7)은 시뮬레이션에 사용된 작업 부하 모델에 따라 shd를 변화시켰을 때 얻어진 트레이스 데이타의 특성(characteristic) 즉 S 블럭에 대한 메모리 접근 요청을 생성했을 때 그 블럭의 상태가 Invalid-by-other(IO) 상태일 확률을 표시한 그림이다.

$$\text{시스템 성능} = \sum_{i=1}^{Npm} \frac{\text{Pi execution time}}{\text{Pi execution time} + \text{Pi idle time}} \times 100$$

(그림 8)은 S 블럭에 메모리 접근 요청을 생성할 확률인 shd를 10%로 고정하고 읽기 요청



(그림 6) 공유 비율의 변화에 따른 시스템 성능의 비교
(Fig. 6) Comparision of system performace when shd rate increases

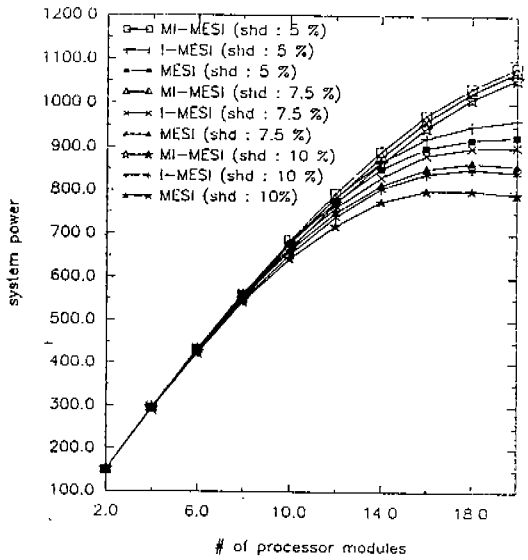


(그림 7) 공유 비율의 변화에 따른 트레이스 데이타의 특성
(Fig. 7) Characteristics of traced data when shd rate increases

율 생성할 비율인 rd의 변화에 따른 MI-MESI 캐쉬 프로토콜과 MESI 및 I-MESI 캐쉬 프로토콜의 시스템 성능을 비교한 시뮬레이션 결과이다.

제시된 시뮬레이션 결과들에 있어서 무효화된 캐쉬 상태를 Invalid와 Invalid-by-other로 분리해서 유지하는 I-MESI 캐쉬 프로토콜의 "시스템 성능"이 MESI 캐쉬 프로토콜의 "시스템 성능"보다 우수하게 나타난 원인은 (그림 7)의 트레이스 데이타 특성에서 보듯이 I-MESI 캐쉬 프로토콜이 전체 캐쉬 실패의 상당 부분을 차지하는 "무효화 캐쉬 실패"에 대해 BRFR나 BRFW 요청 대신에 메모리 모듈이 참여할 필요가 없는 CRFR나 CRFW 요청을 사용하여 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 줄일 수 있기 때문이다.

I-MESI 캐쉬 프로토콜에 비해 MI-MESI 캐쉬 프로토콜의 시스템 성능이 보다 더 우수하게 나타난 원인은 앞에서 기술한 것처럼 MI-MESI 캐쉬 프로토콜이 Modified-shared 상태를 유지하여 불필요한 메모리 갱신을 방지할 수 있을 뿐만 아니라, 다른 프로세서 모듈로부터의 BRFR 요청이나 CRFR 요청에 대해 그 블럭을 Invalid-by-other 상태로 저장하고 있는 프로세서 모듈들이 그 블럭의 상태를 변환하지 않음으로 인해 (그림 7)의 트레이스 데이타 특성에서 보듯이 I-MESI 캐쉬 프로토콜을 통해 제시한 캐쉬-대-캐쉬 통신의 개념을 더욱 효율적으로 사용할 수 있어서 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 줄일 수 있기 때문이다.



(그림 8) 읽기 비율의 변화에 따른 시스템 성능의 비교
(Fig. 8) Comparision of system performace when rd rate increases

5. 결 론

본 논문에서는 MESI 캐쉬 프로토콜과 I-MESI 캐쉬 프로토콜에서 발생하는 불필요한 메모리 갱신을 방지하며, 프로세서 모듈들 사이의 캐쉬-대-캐쉬 통신을 이용하여 분리형 트랜잭션 버스 환경하에서 실질적인 메모리 접근 시간 및 메모리 모듈에서의 접근 충돌을 줄일 수 있는 MI-MESI 쓰기-무효화 스누핑 캐쉬 프로토콜을 제시하였다.

본 논문에서는 또한 제시한 MI-MESI 캐쉬 프로토콜과 기존의 MESI 캐쉬 프로토콜 및 I-MESI 캐쉬 프로토콜의 시스템 성능을 비교하기 위해 캐쉬 시뮬레이터를 작성하여 같은 환경하에서 다양한 파라미터 값들에 대해 시뮬레이션을 수행하였다. 시뮬레이션 결과들은 프로세서 모듈의 수와 메모리 블럭들이 실제로 공유되어 있을 확률 및 쓰기 요청이 발생할 확률이 증가할수록 MI-MESI 캐쉬 프로토콜의 시스템 성능이 MESI나 I-MESI 캐쉬 프로토콜의 시스템 성능에 비해 훨씬 우수함을 보여준다.

본 논문을 통해 제시한 MI-MESI 캐쉬 프로토콜은 6개의 캐쉬 상태를 위해 적어도 3개의 상태 비트가 필요하기 때문에 4개의 캐쉬 상태를 유지하는 MESI 캐쉬 프로토콜에 비해 각 캐쉬 블럭당 한 비트가 더 소요되는 단점이 있지만, 분리형 트랜잭션 버스를 사용하는 중대형 컴퓨터 구조에 쉽게 이식되어 활용될 수 있으리라 기대된다.

참 고 문 헌

[1] Michel Dubois and Shreekant Thakkar, "Cache Architecture in Tightly Coupled

Multiprocessors", Computer, pp. 9-11, June 1990.

[2] Per Stenstrom, "Reducing Contention in Shared-Memory Multiprocessors", Computer, pp. 26-37, Nov. 1988.

[3] D.J. Shanin, "The Design and Development of a Very High Speed System Bus-The Encore Multimax Nanobus.", Proc. of COMPCON, pp. 28-36, 1986.

[4] P. Woodbury et al., "Shared Memory Multiprocessors:The Right Approach to Parallel Processing", Proc. of COMPCON, pp. 72-80, Feb. 1989.

[5] 장성태, 김명주, 이재황, 전주식, "개선된 MESI 쓰기-무효화 스누핑 캐쉬 일관성 프로토 콜", 한국 정보과학회 논문집, Vol. 21, No. 11, pp. 2163qr-2173, 1994.

[6] Per Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors", Computer, pp. 12-24, June 1988.

[7] J. Archibald and J. L. Baer, Cache Coherence Protocols:Evaluation Using A Multiprocessor Simulation model", ACM Trans. on Computer Systems, pp. 273-298, Nov. 1986.

[8] 추영복 외, "TICOM 시스템 버스 프로토콜에서 캐쉬간 직접 전송이 가능한 개선된 프로토콜의 구현", '92 봄 정보과학회 학술발표 논문집, pp. 533-536, 1992.

[9] Don Anderson and Tom Shanley, "Pentium Processor System Architecture", MindSha-re Press, 1993.

[10] Daniel Tabak, "Multiprocessors", Prentice Hall, 1990

[11] 장성태, 전주식, "분리형 트랜잭션 버스를 기반으로한 다중 프로세서 시스템을 위해 개선된 쓰기-무효화 스누핑 캐쉬 일관성 유지 프로토콜", 한국 정보과학회 논문지, Vol 21, No. 1, pp. 53-66, 1994.

[12] Susan J. Eggers, Simulation Analysis of Data Sharing in Shared Memory Multiprocessors", Technical Report No. UCB/CSD 89/501, Univ. of California Berkeley, April 1989.

[13] Wolf-Dietrich Weber and Anoop Gupta, Analysis of Cache Invalidation Patterns in Multiprocessors", ASPLOS-IV, pp. 243-256, 1989.

[14] Christoph Scheurich and Michel Dubois, "Correct Memory Operation of Cache-based Multiprocessors", The 14th ISCA, pp. 234-243, 1987.

[15] Michel Dubois, F. A. Brriggs, "Effects of Cache Coherency in Multiprocessors", IEEE Trans. on Computers, Vol. C-31, No. 11, pp. 1083-1099, Nov. 1982.

[16] A. Alan B. Pritsker, "Introduction to Simulation and SLAM II", System Publishing Corporation, 1986.

[17] Anant Agarwal, et al., "An Evaluation of Directory Schemes for Cache Coherence", The 15th ISCA, pp. 280-289, 1988.

장 성 태

1986년 서울대학교 전자계산기 공학과 졸업(학사)
1988년 서울대학교 대학원 컴퓨터공학과 졸업(석사)
1994년 서울대학교 대학원 컴퓨터공학과 졸업(박사)
1994년 ~현재 수원대학교 전자계산학과 전임강사
관심 분야:병렬처리, 컴퓨터 구조, 캐쉬 프로토콜