□ 기술해설 □

병렬처리 시스템의 캐쉬 일관성

병렬처리 시스템에서 캐쉬 일관성에 대한 고찰

서강대학교 박병섭* · 김성천**

목 차

1. 서 론
2. 하드웨어를 이용한 캐쉬 일관성 프로토콜
2.1 스누피 캐쉬 프로토콜
2.2 디렉토리 기법
2.3 캐쉬 일관성 유지 네트워크
3. 소프트웨어에 의한 방법
3.1 공유데이타의 구분
3.2 캐쉬 일관성 유지 기법
4. 캐쉬 일관성의 구현 예
4.1 Stanford DASH
4.2 SCI(Scalable Coherence Interface)
5. 결 론

1. 서 론

병렬처리 컴퓨터는 기존의 단일 프로세서 시스템으로는 해결하지 못했던 작업을, 동일한 성능의 프로세서를 상호연결하여 성능을 향상시키고 있으며, 빠른 속도와 많은 계산이 요구되는 분야에서 기존의 단일 프로세서 시스템이 수행할 수 없는 일들을 처리할 수 있게 되었다. 이러한 병렬처리 컴퓨터의 중요한 요소중의 하나가 프로세서와 메모리의 효율적인 데이타의 교환이다. 이러한 데이타에 대한 효율적인 접근 및 관리가 전체 시스템의 성능을 좌우하게 됨은 당연하다. 일반적으로 병렬처리 컴퓨터 시스템은 메모리 시스템의 논리적, 실제적 형태에 따라 분산시스템과 공유메모리 시스템으로 구분할 수 있다[1, 2]. 분산시스템에서는 각각의 프로세서들이 자신의 전용 메모리로 접근하고 다른 프로세서들과는 메시지 전달방법을 통하여 상호 통신한다. 이러한 형태의 시스템을 소결합 시스템(loosely-coupled system)이라고 한다. 공유메모리를 사용하는 다중프로세서에서의 통신은 공유변수를 통하여 프로세서들간에 정보를 교환함으로써 구현된다. 이러한 병렬 프로그램의 용이성 때문에 공유메모리 다중프로세서는 현재 상용 시스템으로 널리 사용중에 있으나, 데이타 접근과 관련하여 다음과 같은 문제점이 야기될 수 있다. 즉, 하나의 메모리는 한번에 하나의 메모리 요청만을 처리할 수 있기 때문에 여러 처리기들로부터 메모리 접근 요청이 있는 경우 메모리 충돌(memory contention)이 일어나고 이는 순차적으로 처리된다. 상호연결네트워크에 있는 개별적인 연결 링크에 대한 충돌 현상은 통신 집중화(communication contention)가 발생하여 다른 메모리 모듈로의 접근을 지연시킨다. 또한, 많은 수의 프로세서를 갖는 시스템은 매우 복잡한 상호연결네트워크을 필요로 한다. 따라서, 이러한 네트워크 구조에서는 메모리로의 접근 요청이 지연되는 메모리 접근 시간 지연(memory latency time)이 발생한다.

이상과 같은 문제를 해결하기 위한 하나의

* 준회원
** 종신회원

방법으로 캐쉬 메모리(cache memory)를 사용한다. 캐쉬는 각각의 프로세서들에게 전용으로 연결되어 매우 빠른 속도로 메모리의 일부내용의 지역적 사본(copy)을 저장하여 관리함과 동시에, 프로세서의 속도만큼 빠른 메모리 접근 속도로써 데이타 및 명령어를 제공한다. 따라서 캐쉬를 사용함으로 인하여 메모리 접근 시간을 단축시키며, 또한 메모리 참조에 따른 지역성(locality of memory reference)의 특징도 효과적으로 이용할 수 있다[3-5]. 이러한 구조를 갖는 시스템의 캐쉬들은 하나나 또는 몇개의 적은 수의 프로세서에 의해서 참조되기 때문에 종종 "공유"의 개념과 구분하여 '사유' 캐쉬(private cache) 또는 개별캐쉬라고 한다.

공유메모리를 사용하는 다중프로세서는 프로세서간에 데이타의 공유를 통하여 상호통신하는 방식으로 인하여 동일 메모리 블럭의 여러 복사본들이 동시에 하나 이상의 캐쉬에 저장될 수 있다. 그러나 동일한 자료에 대해서 여러개의 사본이 존재한다는 사실은 결국 사유 캐쉬들간에 일관성을 유지하기 위한 부가적인 작업을 요구한다. 이것을 캐쉬 일관성유지 문제(cache coherence problem)라고 한다. 소결합 시스템에서는 캐쉬의 사용이 별문제가 되지는 않지만 밀결합 시스템에서는 캐쉬와 공유메모리간에 일관성이 이루어져야하기 때문에 이러한 시스템에서의 캐쉬 설계는 보다 복잡해 지고 상호연결네트워크와도 밀접하게 연결되어 있다[3,5,7].

본 고에서는 공유, Read-Write 자료 구조들을 사용할 때 고려되는 캐쉬 일관성유지 문제와 이를 해결하기 위하여 제안된 방법들에 대하여 살펴보기로 한다. 현재 사용되고 있는 방법들은 일관성 유지문제를 전적으로 하드웨어가 유지관리하는 프로토콜에서부터 공유되는 갱신가능한 데이타의 중복 사본이 존재하지 않도록 소프트웨어를 이용하는 프로토콜까지 다양하게 구현되어 있다. 따라서 본 고에서는 현재 상용되고 있거나 제안된 프로토콜과 정책들이 각각의 일관성 유지 관련 문제들을 어떻게 처리하는가에 대하여 살펴보기로 한다.

2. 하드웨어를 이용한 캐쉬 일관성 프로토콜

하드웨어 프로토콜을 사용하는 캐쉬 메모리 시스템에서 캐쉬와 메모리 사이의 정보교환의 단위는 일정한 양의 메모리 워드들로 구성된 블럭(block)이다. 이러한 동일한 블럭은 동시에 여러개의 사본이 전체 시스템 내에 존재하도록 허용하여 각각 다른 캐쉬에 저장될 수 있다. 이렇게 여러 캐쉬에 동시에 존재하는 메모리 블럭들 사이의 일관성을 유지하기 위해서 사용하는 정책에는 기록-무효화(write-invalidate) 정책과 기록-갱신(write-update) 정책이 있다[3,6].

기록 무효화 정책은 여러벌의 동일한 사본이 존재하여 읽기-요청을 지역적으로 처리할 수 있으나, 어떤 프로세서가 블럭의 내용을 갱신하면, 다른 캐쉬들에 저장되어 있는 모든 사본을 무효화하는 방법이다. 예를 들어 그림 2.1(a)는 임의의 블럭 X가 시스템내에 동시에 네개(메모리 하나와 캐쉬에 3개)의 사본으로 존재하는 상황이며, 그림 2.1(b)는 프로세서 1에 의하여 블럭 X의 내용이 갱신되어 블럭 X'이 되고 이외의 다른 모든 사본이 무효화된 상황이다(그림에서 I로 표시되었음). 이때, 프로세서 2가 블럭 X'에 있는 데이타에 대하여 읽기-요청을 하면, 프로세서 1에 연결된 캐쉬는 해당 데이타를 공유메모리로 복사시키고 이를 프로세서 2가 읽어들이도록 한다.

기록-갱신 정책은 나머지 모든 사본들의 내용을 같은 내용으로 갱신하는 방법이다. 그림 2.1(c)는 기록-갱신 정책을 사용하는 경우, 네트워크은 블럭 X의 갱신후 자동적으로 나머지 사본들도 갱신하였음을 보여준다. 나머지 사본을 갱신하는 방법은 여러가지 방법으로 구현될 수 있다. 캐쉬 일관성 문제를 하드웨어로 구현하는 기법은 시스템 자체가 캐쉬의 일관성을 보장하여 주기 때문에 매우 효율적인 해결책을 제공하는 반면, 복잡한 하드웨어로 인하여 많은 비용이 필요하다. 하드웨어를 이용한 프로토콜은 다음과 같이 크게 3가지로 분류된다. 즉, 1) 스누피 프로토콜(Snoopy protocol)은 주로 공유버스를 사용하는 다중프로세서에서 사용하는 프로토콜이고, 2) 디렉토리 기법(Directory scheme)은 다단계 상호연결네트워크로 연결된 다중프로세서에서 사용하는 프로토콜이다. 그리고 3) 캐쉬 일관성네트워크 구조(Cache coherence network archite-

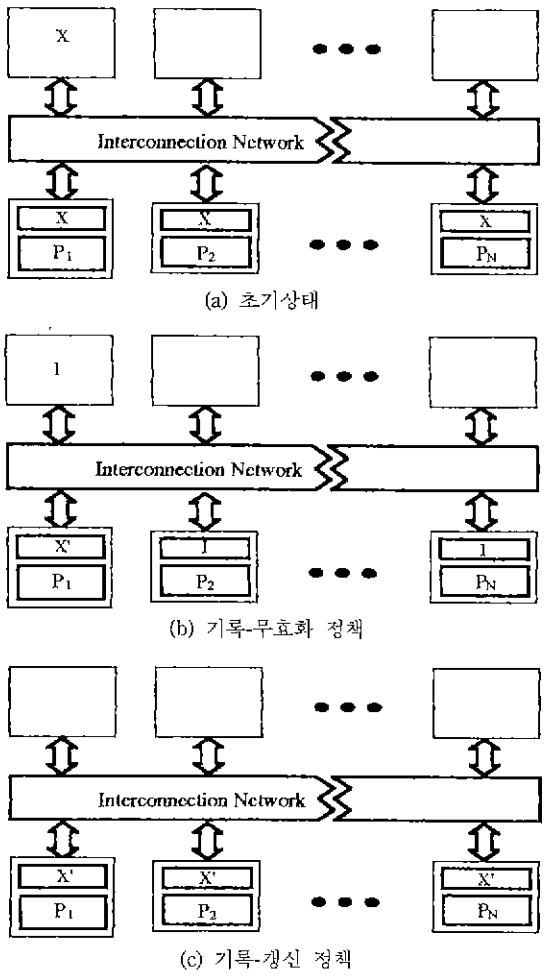


그림 2.1 캐쉬 일관성 프로토콜 (IEEE, Computer, June, 1990)

cture)는 공유버스 구조와 다단계 상호연결 네크워크의 단점을 보완하려는 목적으로 제안된 것으로 캐쉬 일관성을 효율적으로 유지하기 위해 변형된 네트워크 구조를 이용하여 캐쉬 일관성 문제를 해결하려는 방법이다.

2.1 스누피 캐쉬 프로토콜

버스를 사용하는 스누피 캐쉬 프로토콜에 의한 시스템은 일관성 유지에 관련된 명령들을 모든 캐쉬들에게 방송하고, 버스를 통하여 들어오는 모든 일관성 유지 관련 명령들을 일일이 검색하여 캐쉬 일관성을 유지하는 절차를 수행하여야 한다. 이 경우 크게 두가지 프로토콜을 따른다.

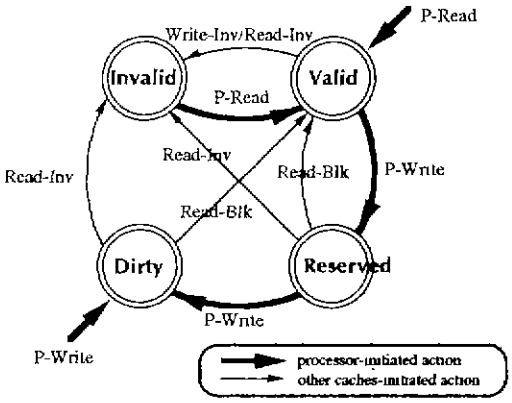
2.1.1 Write-once 프로토콜

Write-once 프로토콜은 스누피 캐쉬 프로토콜 방식을 기반으로 기록-무효화 정책을 구현한 것으로 Goodman[3,6]에 의하여 제안되었다. Write-once 프로토콜에서는 각 사본들의 상태를 다음과 같이 네가지로 구분한다. 즉, 블럭 내에 원하는 자료가 존재하지 않는 상태인 Invalid, 블럭내에 원하는 자료가 존재하고 메모리 블럭의 내용과 같은 상태인 Valid, 자료의 내용이 단 한번 갱신되었으며 갱신후 메모리의 내용을 갱신시켰기 때문에 여러 사본들 중에서 유일하게 메모리의 내용과 일치하는 Reserved 및 블럭이 캐쉬로 복사된 후 그 내용을 여러번 갱신하였으며 최근의 갱신 내용이 메모리로 재 복사되지 않은 상태인 Dirty 상태 등이다.

Write-once 프로토콜은 블럭이 캐쉬 내에 머물러 있는 동안은 연속해서 갱신되고 (이러한 상태를 dirty 상태라고 한다.), 메모리로 대체되어 나가는 경우에는 전체 블럭의 내용이 메모리로 재 복사되는 방법이다(copy-back update). 그림 2.2는 일관성 유지 명령에 의하여 사본의 상태가 변환되는 과정을 나타낸 것이다.

읽기-실패와 기록-성공, 그리고 기록-실패에 대한 각각의 처리 내용은 다음과 같다.

- 읽기-실패시: Dirty 사본이 없는 경우, 메모리로부터 한 블럭을 복사한다. Dirty 사본이 있는 경우에는 그 사본의 내용을 복사하고 메모리 내용을 갱신시킨다. 모든 처리 후의 사본들은 Valid 상태가 된다.
- 읽기-성공시: Dirty 혹은 Reserved 상태에 있는 경우, 지역적 갱신 작업을 수행하고 Dirty 상태가 된다. Valid 상태에 있는 경우에는 무효화 명령을 방송하여 다른 모든 사본들의 내용을 무효화시키고, 자신은 갱신 작업 후 Reserved 상태가 된다.
- 기록-실패시: 읽기-실패의 경우와 비슷한 작업을 수행하며 모든 작업 처리후, Dirty 상태가 된다. 이때, 다른 사본들은 Invalid 상태가 된다.



- Read-Blk: 블럭의 자료를 읽어 들인다.
- Write-Blk: 블럭의 자료를 갱신한다.
- Write-Inv: 블럭의 다른 모든 사본들을 무효로 만든다.
- Read-Inv: 블럭을 읽어 들이고, 다른 모든 사본들을 무효로 만든다.

그림 2.2 Write-once 프로토콜의 상태 전이도(IEEE, Computer, June, 1990)

- 블럭 대체 : Dirty 상태에 있다면 모든 사본의 내용이 메모리로 재 복사 된다. 그렇지 않으면 아무런 작업도 수행되지 않는다.

기록-무효화 정책을 사용하는 시스템으로는 Symmetry 다중프로세서(Sequent Computer System)와 Alliant FX(Alliant Computer System) 등이 있다.

2.1.2 Firefly 프로토콜

Firefly 프로토콜은 Digital Equipment의 Firefly 다중처리기 워크스테이션을 위하여 제안된 것으로서 기록-갱신 정책을 구현한 것이다[3,6]. 사본들의 상태는 다음과 같이 세가지로 구분된다. 즉, 메모리 내용과 일치하며 유일한 사본임을 나타내는 Valid-exclusive, 메모리의 내용과 일치하며 다른 일관성 있는 사본들이 존재함을 의미하는 Shared 및 유일한 사본이지만 메모리 내용과 일치하지 않는 Dirty상태이다.

Firefly 프로토콜에서는 사유 블럭에 대하여 copy-back-update 정책을 사용하고 공유 블럭에 대해서는 write-through 정책을 사용한다. 공유 또는 사유의 개념은 수행 당시에 결정된다. "공유라인 (shared line)"이라는 전용버스는 해당

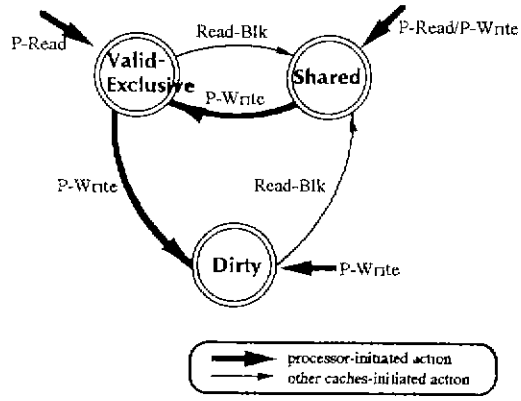


그림 2.3 Firefly 프로토콜의 상태 전이도(IEEE, Computer, June, 1990)

사본의 상태가 Shared인지 아닌지를 빨리 알려주는 장치이다. 그림 2.3는 Firefly 프로토콜에서의 상태 전이도이다.

- 읽기-실패: Shared 사본이 있는 경우 그 사본의 내용을 제공받는다. Dirty 상태인 경우, 그 내용을 제공하고 메모리 내용을 갱신한 다음 shared 상태로 상태 변화를 한다. 다른 사본이 존재하지 않는 경우, 메모리로부터 블럭을 복사하고 Valid-exclusive 상태로 상태 변환한다.
- 기록-성공: Dirty 혹은 Valid-exclusive 상태인 경우, 모든 처리를 지역적으로 처리한 후, Dirty 상태가 된다. Shared 상태인 경우, 모든 처리를 지역적으로 처리한 후, Dirty 상태가 된다. Shared 상태인 경우, 다른 사본들의 내용을 갱신한다. Shared 상태가 아닌 경우에는 모든 처리가 끝나고 나면 Valid-exclusive 상태로 상태 변환한다.
- 기록-실패: 메모리 혹은 다른 캐쉬로부터 블럭의 내용을 제공받는다. 메모리의 내용을 복사하는 경우, 모든 처리가 끝나고 나면 Dirty 상태로 변환한다. 다른 캐쉬들의 내용도 갱신하고 Shared 상태로 상태 변환한다.
- 블럭 대체: Dirty상태인 경우, 모든 사본의 내용이 메모리로 재 복사되며, 그렇지 않은 경우에는 아무런 작업도 수행하지 않는다.

스누피 캐쉬 프로토콜을 구현할 때, 필요한 장치는 다음과 같다.

- 캐쉬 제어기: 그림 2.3과 같이 정의된 규칙을 구현한 유한 상태 기계이다.
- 캐쉬 디렉토리: 각각의 블럭들의 상태 정보들을 저장하기 위한 장소로서 각 블럭에 대하여 2비트씩의 장소가 필요하다.
- 버스 제어기 : 버스를 모니터 하는 기능을 가진 것으로서 스누피 기법을 구현한다. 대개 스누피 기법은 디렉토리에 대한 집중화 현상을 발생시키기 때문에 캐쉬 디렉토리를 중복 복사하여 사용한다.

이외에도 스누피 프로토콜을 효과적으로 구현하기 위해서는 전용버스선이 필요하며 그 예로서 "공유선" 등이 있다[3]. 또한, 표준 버스 구조로서 IEEE의 FutureBus(IEEE 표준 P896.1)가 있다[2].

메모리 참조로 인한 평균 버스의 교통량(traffic bandwidth)은 캐쉬의 접근 실패 비율과 실패가 발생하였을 때 옮겨야 하는 메모리 단어의 수(블럭의 크기)의 곱으로 표현된다. 결국 블럭의 크기가 큰 경우, 접근 실패율이 감소되어도 버스 트래픽은 감소되지 않는다. 기록-무효화 정책을 사용하는 경우, 캐쉬 접근 실패는 다른 프로세서로부터의 무효화 명령에 의하여 발생하며, 이 때문에 캐쉬 블럭의 크기는 버스의 트래픽의 감소에 전혀 도움을 주지 않는다. 따라서 일반적으로 버스를 사용하는 다중프로세서 시스템에서는 작은 크기의 캐쉬 블럭을 사용한다[3].

기록-갱신 정책을 사용하는 다중프로세서의 경우, 일관성 유지 작업으로 인한 캐쉬 접근 실패가 발생하지 않기 때문에 캐쉬 블럭의 크기는 큰 문제가 되지 않는다. 전역적인 갱신의 발생율도 블럭 크기에 무관하다. 기록-갱신 정책을 사용하는 경우, 자료들이 대체되어 나갈 때까지 캐쉬 내의 자료들은 무효화되지 않는다. 따라서 캐쉬 블럭의 크기가 크면, 접근 실패의 발생 횟수가 줄어들고 이로써 버스의 트래픽을 감소시킨다. 그러나, 전역적인 갱신에 의하여 실제로 사용되지 않는 캐쉬의 내용까지도 갱신되는 문제점이 발생한다.

스누피 캐쉬 프로토콜에서 기록-갱신 정책을 사용하는 경우의 가장 중요한 문제는 무효화 실패의 발생율을 최소화하는 것이며, 기록-갱신 정책을 사용하는 경우는 버스의 트래픽을 줄이기 위하여 공유 자료의 양을 줄인다. 이를 위하여 읽기방송(read broadcasting) 과 경쟁적 스누핑(competitive snooping) 등의 방법을 이용한다. 읽기방송 방법은 읽기실패가 발생하여 새로운 블럭 내용을 복사할 때 다른 무효가 된 사본을 가지고 있는 캐쉬들에게도 같은 내용의 블럭을 방송하는 방법이다. 이 방법은 읽기 실패율을 줄이고자 하는 것으로서 Rudolph와 Segall[8]에 의하여 제안되었다. 경쟁적 스누핑 방법은 Eggers와 Katz[9,10]에 의하여 제안된 방법으로 통신에 드는 비용과 무효화 실패를 처리하기 위하여 드는 비용을 초과하는 점을 채산점(breake-even point)으로 두는 방법이다.

기록-갱신 정책을 채용하고 있는 시스템으로는 Firefly(DEC)와 Dragon 워크스테이션(Xerox P-ARC) 등이 있다[6].

2.2 디렉토리 기법

단일 버스를 사용하는 다중프로세서는 버스의 용량으로 인하여 연결할 수 있는 프로세서의 수가 한정되어 있다. 그러나 다단계 연결네트워크를 사용하는 다중프로세서에서는 많은 수의 프로세서들을 연결할 수 있다.

모든 캐쉬들에게 갱신 명령을 방송하는 스누피 캐쉬 프로토콜은 일반적인 상호 연결네트워크의 성능을 급격하게 저하시키는 현상을 발생시키기 때문에 일관성 유지 명령을 같은 블럭의 사본들을 가지고 있는 캐쉬들에게만 보내는 방법을 사용하여야 한다. 이와 같이 한 블럭의 사본들이 어느 캐쉬에 저장되어 있는 가에 대한 정보를 사용하는 일관성 유지 방법을 디렉토리 기법(directory scheme)이라고 한다. 디렉토리 기법을 사용하는 프로토콜은 디렉토리의 정보를 유지, 관리하는 방법과 디렉토리내에 저장하는 정보의 종류에 따라 구분된다[3,11,12].

2.2.1 디렉토리 관리에 의한 분류

Tang[11]이 제안한 방법은 중앙에 모든 캐쉬의 디렉토리를 중복하여 저장하는 방식으로 이 방법에서는 특정 블럭에 대한 사본을 찾기 위하

여 캐쉬 제어기가 모든 중복 디렉토리를 탐색해야 한다. 이 방법은 구현할 때 가장 적은 수의 비트가 필요한 장점이 있으나, 디렉토리를 중앙 관리함으로 인하여 집중화 현상이 발생하며, 또한 해당 블럭의 사본들을 찾기 위하여 모든 디렉토리의 내용을 탐색하여야 하는 단점이 있다.

Censier 등[12]은 각 메모리 블럭마다 존재 플래그 벡터(presence flag vector)라고 불리는 비트 벡터를 할당하여 블럭의 사본들이 존재하는 캐쉬들의 위치를 빠르게 찾을 수 있도록 한다. 또한 각 사본들의 현재 상태를 나타내는 몇개의 비트가 필요하며 그 내용과 크기는 사용하는 캐쉬 일관성 유지 정책에 따라 다르다. Stenstrom [13]은 메모리 블럭마다 상태 정보와 존재 플래그 벡터를 두는 Censier 등의 방법과는 달리 캐쉬의 사본과 연관된 디렉토리 정보를 사용한다.

Censier 기법과 Stenstrom 기법은 메모리 혹은 캐쉬 모듈에 디렉토리 정보들을 분산시켜 두기 때문에 디렉토리에 대한 집중화 현상을 감소시킬 수 있다. 또한, 존재 플래그 벡터를 사용하여 Tang 기법에서 문제점으로 지적되는 매번 발생하는 모든 사본들에 대한 탐색을 제거할 수 있다. 물론, Censier 기법을 구현할 때 필요한 비트의 수는 메모리 크기와 비례하고, Stenstrom 기법은 캐쉬의 크기와 비례하는 부담을 가진다. 또한 Stenstrom 기법에서는 현재 메모리의 사용자가 누구인지를 구분하여야 한다. 일반적으로, Censier 기법과 Stenstrom 기법은 메모리의 사용자가 누구인지를 구분하여야 한다. 여기서 기록-무효화 정책을 사용하고, 시스템 내에 Dirty사본이 단하나 존재한다고 가정할 때, Censier 기법과 Stenstrom 기법에서 읽기 실패와 기록 성공에 따른 처리 절차는 다음과 같다.

Censier 기법

- 읽기 실패: 메모리 제어기는 메모리 접근 요청을 Dirty 사본을 가지는 캐쉬에게 보내어 Dirty 사본의 내용을 메모리로 재복사하도록 한다. 메모리 내용의 갱신이 완료된 후, 블럭의 내용을 읽기 요청을 보낸 캐쉬 내로 복사한다.
- 기록 성공: 캐쉬의 내용을 갱신시킨 후, 이를 메모리 제어기에 알린다. 메모리 제어기는 존재 플래그 벡터에 표시되어 있는 캐쉬들에게 무효화 명령을 방송한다.

Stenstrom 기법

- 읽기 실패: 메모리 접근 요청을 받은 메모리 제어기는 Dirty사본을 가진 캐쉬에게 사본의 내용을 읽기를 요청한 캐쉬에게로 직접 복사하여 주도록 한다.
- 기록 성공: 존재 플래그 벡터에 표시되어 있는 캐쉬들에게 무효화 명령을 캐쉬가 직접 방송한다.

표 2.1 제안된 3가지 기법의 비트 오버헤드(IEEE, Computer, June, 1990)

기 법	오버헤드(No. of bits)
Tang	CB
Censier	M(B+N)
Stenstrom	C(B+N) + MlO

표 2.1은 위에서 설명한 3가지 기법을 정보를 저장하는데 필요한 비트의 수 관점에서 구현비용을 나타낸 것이다. 여기서 M은 메모리 블럭, C는 캐쉬 라인, N은 캐쉬 수, 그리고 B는 각 캐쉬 블럭에대한 상태 정보를 표현하는데 필요한 비트수이다.

2.2.2 디렉토리에 저장하는 정보에 따른 분류

분류디렉토리 기법의 가장 중요한 장점은 일관성 유지를 위한 명령을 블럭의 사본을 가지고 있는 캐쉬들에게만 보내어 처리하도록 제한할 수 있는 점이다. 갯수에 제한 없이 캐쉬의 사본들에 대하여 정보를 저장하는 방식을 완전-사상 디렉토리 기법(full-map directory schemes) 이라고 한다. 이 기법은 다중프로세서에서 프로세서의 수가 많아질 수록 구현할 때 많은 비용이 필요하다[3,11].

디렉토리의 크기를 줄이기 위한 것으로서 실제 캐쉬의 수보다 캐쉬 포인터의 수를 적게 하여 저장하는 제한 디렉토리 기법(limited directory

표 2.2 각 기법의 포인터를 위한 비트 오버헤드(IEEE, Computer, June, 1990)

기 법	오버헤드 (No. of bits)
Full map	$M\log_2 N + CN$
Limited	$iM\log_2 N$
Chained	$M\log_2 N + C\log_2 N$

schemes)이 있다. 캐쉬의 수가 N이고 각 디렉토리 항목에 있는 포인터의 수가 $i<(iN)$ 만큼 있을 때, 각 메모리 블럭에 대한 정보를 저장하기 위해서는 $i\times\log_2^N$만큼의 비트가 필요하다. 이 기법을 사용할 때 가장 중심이 되는 문제는 i개 이상의 캐쉬가 블럭의 사본을 요청한 경우의 처리 문제이다. 대개, i 이상의 요청들을 무시하는 방법과, i 이상의 사본이 존재하면 방송을 사용하는 방법으로 처리한다.

디렉토리를 위하여 필요한 비트의 수를 줄이기 위하여 제안된 방법으로 디렉토리 내에는 사본을 가지는 최초의 캐쉬에 대한 정보만을 저장하고 각 캐쉬들은 다음 캐쉬에 대한 포인터를 가지는 연결 디렉토리 기법(chained directory schemes)이 있다. 완전 사상 디렉토리 기법과 비교하여 더 많은 시간이 소요되며, IEEE SCI(IEEE P 1596)가 연결 디렉토리 기법을 제안하고 있다[14].

표 2.2는 위 3가지 방법에 대해, 캐쉬 포인터를 위해 필요한 비트 오버헤드를 보여주고 있다.

그 밖에 일관성 유지 명령어를 방송을 통하여 처리하는 방식이 있다. 이것은 프로세서의 수가 늘어나도 디렉토리의 구조를 갱신시키지 않고 쉽게 처리할 수 있다는 장점이 있으나 심각한 네트워크 트래픽을 유발시킬 수 있다.

이상에서 살펴본 바와 같이 디렉토리 기법은 연결 네트워크의 트래픽과 디렉토리의 크기에 대한 trade-off가 존재한다는 특성을 가지게 된다.

2.3 캐쉬 일관성 유지 네트워크

계층 버스 구조를 가지는 다중프로세서는 공유

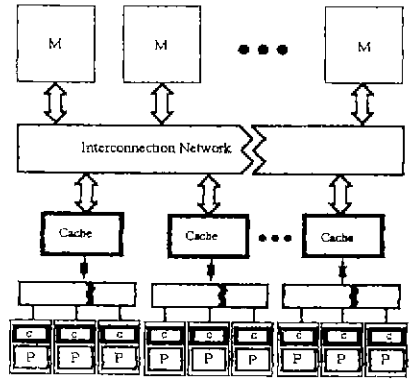


그림 2.4 확장가능한 다중프로세서를 위한 계층적 캐쉬/버스 구조 (Proc. 14th Int'l Symp. on Comp. Arch., 1987)

버스의 한계와 다단계 연결네트워크의 문제점을 보완하기 위하여 제안된 것이다. 계층 버스 구조는 디렉토리 기법의 복잡성으로 인하여 구현에 어려움을 겪지 않는 계층적 캐쉬 일관성 유지 프로토콜을 사용함으로써 연결네트워크의 트래픽을 감소시킨다. 계층적 버스 구조를 가지는 다중프로세서로는 Wilson이 제안한 계층적 캐쉬/버스 구조[15], Goodman에 의하여 제안된 Wisconsin 다중 큐브[16] 및 Swedish Institute of Computer Science에서 설계한 Data Diffusion Machine[17]이 있다.

2.3.1 계층적 캐쉬/버스 구조

계층적 캐쉬/버스 구조(Hierarchical Cache/Bus Architecture)는 Wilson[15] 이 제안한 구조로써, 그림 2.4와 같이 단말노드가 프로세서(P_j)와 그에 연결된 사유캐쉬(C_{1j})인 다중레벨 트리(Multilevel tree)구조이다. 각 클러스터는 클러스터 버스(Cluster Bus)로 연결되어 있으며, 클러스터간의 통신은 클러스터간 버스(Intercluster Bus)를 통해서 이루어진다. 레벨-2 캐쉬(C_{2j})는 각 클러스터 버스와 클러스터간 버스사이에 사용된다. 각 레벨-2 캐쉬는 그 아래 레벨에 연결된 모든 레벨-1 캐쉬용량의 합보다 더 큰 크기의 기억용량을 가진다. 그리고 각 하나의 클러스터는 단일버스 시스템처럼 동작하며, 같은 클러스터에 속한 레벨-1 캐쉬사이의 일관성 유지는 스누피 버스 프로토콜을 사용한다.

그림 2.5은 계층적 캐쉬/버스 구조에서 블럭에 관한 요청을 예로 설명한 것이다. 이 그림에서 프로세서 P_1이 기록요청을 발생했다고 가정하면, 기록-요청은 최상위 레벨까지 전달되어 모든 사본을 무효화 시킨다. 결과적으로 C_{20}, C_{22}, C_{16}, C_{18} 캐쉬 사본이 모두 무효화된다. C_{20}과 같은 상위 레벨의 캐쉬는 그 아래 레벨의 수정된 블럭의 상태를 유지하고 있어야 한다. 이 동작이 끝난 후에 P_7에 의한 연속적인 읽기요청은 더이상 해당 블럭에 대한 복사본이 존재하지 않기 때문에 계층의 상위레벨로 전달되어, 다른 클러스터인 원격 클러스터의 상위레벨 캐쉬인 C_{20}에 읽기요청이 도착하게 된다. 이 요청이 도착되면 C_{20}은 C_{11}로 flush요청을 보내고, 수정된 복사본을 P_7에 공급한다.

여러 사본들간의 일관성을 유지하기 위하여, 기록 무효화 프로토콜의 확장된 형태의 프로토콜을 사용한다. 같은 계층에 저장되어 있는 사본들 간의 일관성 유지는 전통적인 스누피 캐쉬 프로토콜과 같은 방법으로 처리된다. 그러나, 무효화는 다른 모든 사본들을 무효화하기 위하여 수직 계층간으로 전파된다. 높은 계층의 캐쉬들로 전파된 갱신 요청은 역시 높은 계층의 캐쉬가 받아들여 관련 사본들의 내용을 모두 무효화한다.

레벨-2 캐쉬는 각 지역 클러스터에서 상위레벨 클러스터로의 일관성을 확장하는데 사용된다. 상위레벨 캐쉬는 각 클러스터 사이에 다른 레벨의 공유메모리를 형성하고, 주 메모리 모듈은 클러스터간 버스를 통하여 연결된다. 상위 계층의 캐쉬들은 일관성 유지 행위에 대한 일종의 여과 작용을 한다. 즉, 대부분의 메모리 요청은 하위레벨 캐쉬에서 만족되므로 상위레벨로의 트래픽을 현저히 줄일 수 있고, 또한 클러스터간의 캐쉬 일관성은 두번째 레벨 캐쉬들 사이에서 제어되므로, 하위레벨로의 트래픽 전파를 막을 수가 있다.

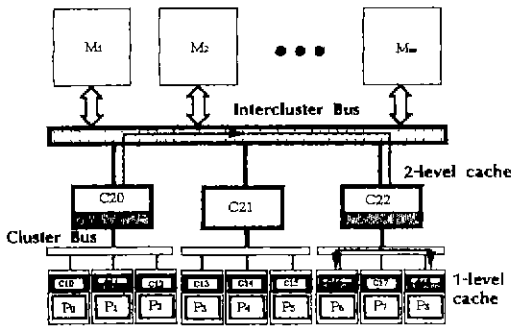


그림 2.5 계층적 캐쉬/버스 구조에서 블럭 요청 예(Proc. 14th Int'l Symp. on Comp. Arch., 1987)

2.3.2 Wisconsin 다중큐브

Wisconsin[16] 다중 큐브는 그림 2.6에서와 같이 격자 무늬 버스 구조를 이루고 있다. 이러한 격자 무늬의 버스는 스위치마다 프로세싱 소자를 가지고 있으며, 열(column) 버스에는 메모리 모듈이 연결되어 있다. 하나의 프로세싱 소자는 캐쉬와 프로세서, 그리고 행(row) 버스와 열 버스에 연결된 스누피 캐쉬 제어기로 구성되어 있다.

계층적 캐쉬/버스 구조 시스템에서와 같이 기록-무효화 프로토콜을 사용하여 일관성을 유지한다. 무효화 명령은 행 버스를 통하여 방송된다. 또한 행 버스를 이용하여 방송되는 읽기요청은 블럭의 사본을 가진 캐쉬에 대한 정보를 가지고 있는 캐쉬 제어기에 의하여 열 버스를 거쳐, 사본이 저장되어 있는 캐쉬로 경로 조정된다.

2.3.3 Data Diffusion Machine

DDM(Data Diffusion Machine)[17]은 계층적

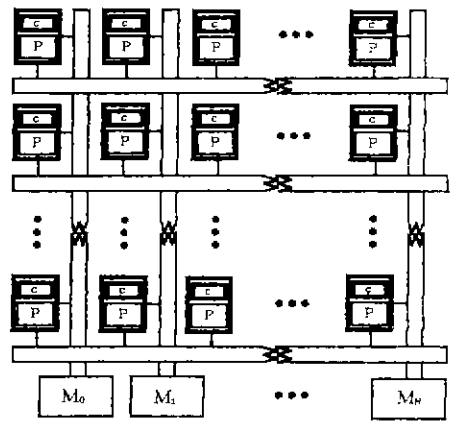


그림 2.6 Winsconsin 다중 큐브 (Proc. 15th Int'l Symp. on Comp. Arch., 1988)

캐쉬/버스 구조와 흡사한 계층적 캐쉬 일관성 유지 구조이다. DDM은 시스템 메모리 형태로 사용되는 큰 규모의 프로세서 캐쉬를 가장 하위 계층에 두고 계층적 버스 구조를 형성한다. 계층적 기록 무효화 프로토콜을 사용하여 캐쉬 일관성을 유지, 관리한다. 그림 2.7에 DDM에 대한 그림이 보여지고 있다. DDM은 계층적 캐쉬/버스 구조와는 달리 상위계층의 메모리는 단지 상태 정보만을 포함하고 있는 디렉토리이기 때문에 많은 메모리 부담을 줄일 수 있다. 그러나, 중간 단계의 디렉토리들은 요청에 대해 안전성을 보장하지 않기 때문에 읽기요청들을 root에게 보낸 다음, 다시 leaf까지 되돌아 와야 하는 문제점이 있다.

DDM에서는 전역적 메모리를 각 프로세서들에게 분산한다. 따라서 사본의 수에 제한이 없다. 자료들은 원칙적으로 장소가 없기 때문에 또한, 자료들에 대한 기본적인 장소(home location)가 없기 때문에 메모리 모듈의 내용을 요구하는 곳이면 어디든지 확산한다. 이러한 구조와 완전 사상 디렉토리를 비교하여 보면, 아직 충분하게 구현된 것은 아니나, 메모리 부담의 측면에서 경제적이다. 또한, 대규모의 공유메모리 다중프로세서를 위하여 버스를 기본으로 하는 구조이며, 가장 적은 비용이 소요된다.

3. 소프트웨어에 의한 방법

하드웨어를 이용하여 캐쉬 일관성 유지 문제를 해결하는 방법은 연결 네트워크의 통신량을 효과적으로 감소시킨다. 그러나 보통 하드웨어를 사용하는 프로토콜은 처리 방법이 매우 복잡하여 프로세서의 수가 많은 다중프로세서인 경우, 구현하기가 매우 어렵다. 이러한 문제를 해결하기 위한 것으로서 안전한 때에만 자료들을 캐쉬에 저장하는 방법이 있다. 즉, 프로그램을 분석하여 캐쉬에 저장할 수 있는 자료와 그렇지 않은 자료를 미리 구분하는 방법이다. 가장 간단한 방법으로 모든 공유 변수들을 캐쉬에 저장하지 않는 방법이 있으나, 공유 자료 구조가 한 프로세서에 의해서만 접근이 가능하거나 혹은 상당한 시간 동안 read만 가능하게 되기 때문에 효율이 급격하게 감소한다는 단점과 복잡한 컴파일러나 프리프로세서가 필요한 단점이 있다[18].

좀 더 효율적인 방법으로 제안된 것으로서 컴파일러에 의하여 공유 변수가 캐쉬에 저장될 안전한 시기를 분석하여 결정하는 방법이 있다. 결정된 안전한 기간 동안 변수는 캐쉬 내로 저장이 가능한 자료로 표시되며, 그 기간이 끝나는 순간에 캐쉬에 있는 자료는 메모리의 내용과 일치하여야 한다. 기간이 지난 후에는 모든 캐쉬에 있는 자료들을 무효화하여 접근이 불가능하도록 한다. 이때, 컴파일러가 자료들을 구분하는 방법과 자료들을 무효화시키는 방법에 따라 다음과 같이 소프트웨어를 사용하는 캐쉬 일관성 유지 기법들을 구분할 수 있다.

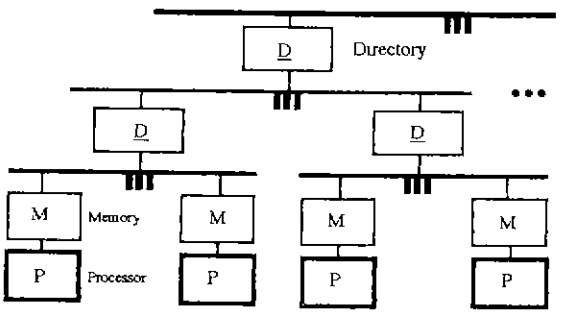


그림 2.7 Data Diffusion Machine (Proc. PARLE89, vol 1, Springer-Verlag, 1989)

3.1 공유데이타의 구분

프로그램을 정적인 계산 단위(computational unit)로 나누어 공유변수에 참조 표시 (reference marking)을 할 수 있다. 하나의 계산 단위에서의 공유변수에 대한 접근은 다른 계산 단위에서와 다를 수도 있다. 예를 들어, 다음과 같은 형태의 접근 방식들이 존재할 수 있다;

(1) 임의의 수의 프로세스가 read 가능.
(2) 꼭 하나의 프로세스만이 read-write 가능, 그 외의 프로세스는 읽기만 함.
(3) 오직 하나의 프로세스가 read-write 가능.
(4) 임의의 수의 프로세스가 read-write 가능.

각 프로세스는 서로 다른 프로세서에서 수행된다고 가정하면, (1)에 해당하는 접근은 그 변

수를 캐쉬에 저장할 수 있음을 나타낸다. (2)의 변수는 read와 write가 가능한 프로세스의 캐쉬로만 저장이 가능하고 메모리의 내용과는 write-through 정책을 사용하여 일관성을 유지할 수 있다. (3)의 변수는 모든 캐쉬에 저장될 수 있으며 copy-back 방식을 이용하여 갱신할 수 있다. 마지막으로 (4)의 변수들은 어떠한 캐쉬로도 저장이 불가능하다. 예를 들면, 상호 배제에 의한 동기화나 장벽(barrier) 동기화에 사용되는 변수들이 이 범주에 속한다.

동기화는 대개 하나의 계산 단위의 범위를 정하는 역할을 하는 경우가 있기 때문에, 어떤 변수의 일관성을 유지하기 위하여 다른 계산 단위에 속한 변수들은 각각 다른 접근 방식을 적용할 수 있다. 한 계산 단위 내에서 캐쉬에 저장되었던 공유변수들은 다음 계산 단위가 수행 시작되기 전에 모두 무효화되어야 하며, 메모리의 내용도 write-through 정책이나 copy-back 방식을 사용하여 갱신이 완료되어야 한다.

3.2 캐쉬 일관성 유지 기법

첫번째 방식은 Cheong과 Veidenbaum[18]에 의해 제안된 것으로 하나의 계산 단위 내에서 참조되는 모든 공유변수는 모두 캐쉬에 저장되거나 모두 저장되지 않는다는 똑같은 형태로 처리된다. 또한 메모리가 항상 최근에 갱신된 내용을 포함하고 있도록 write-through 정책을 사용한다. 그 밖에 Cache-On, Cache-Off, Cache-Invalidate 등의 세가지 캐쉬 명령어를 사용한다. Cache-On은 모두 읽기만 가능한 접근 방식을 사용하거나 (1의 형태), 하나의 프로세스에 의해 배타적으로 접근되는 경우 (3의 형태)의 모든 공유변수에 대해 캐쉬에 적재가 가능함을 표시한다. Cache-Off는 모든 공유변수에 대한 접근을 캐쉬에서 이루어지지 않고, 메모리에서 이루어지도록 한다.

하나의 계산 단위가 수행된 다음에는 Cache-Invalidate 명령을 사용하여 모든 캐쉬의 사본들을 무효화하게 한다. 이러한 전체 무효화를 무차별 무효화(indiscriminate invalidation)[18]라 하며, 효율적이고 구현이 용이하다. 그러나, 이 방법의 사용은 불필요하게 캐쉬의 접근 실패율을 높이는 원인이 된다.

선택적 무효화(selective invalidation)는 높은 캐쉬의 접근 실패율의 결점을 개선시킨 것으로 불일치를 초래한 변수만 무효화하는 방식이다. 그러나 이러한 방법을 효율적으로 구현하는 것이 중요한 문제이다. Cheong과 Veidenbaum[19]의 방법은 선택적 무효화의 효율적인 구현을 위한 기법이다. 이 방법에서 하나의 계산 단위 내의 공유변수 접근 형태는 "최신" 또는 "갱신 가능"으로 분류된다. 그리고 캐쉬 명령어로 Memory-Read, Cache-Read, Cache-Invalidate 등이 사용된다. Memory-Read는 캐쉬의 사본이 갱신된 것일 수도 있다는 것을 의미하고, Cache-Read는 캐쉬의 사본이 항상 최신의 것임을 보장한다. 또한 이 기법은 write-through 정책을 사용한다.

각 캐쉬 라인과 연관된 갱신 비트가 있어서, Cache-Invalidate 명령에 의하여 모든 갱신 비트는 1(true)로 변한다. 갱신 비트가 1인 캐쉬 블럭에 Memory-Read 명령이 들어오면 그 요청은 메모리로 전달된다. 요구된 블럭이 다시 캐쉬로 적재되면서 갱신 비트는 0(false)이 되고 이후의 접근 요청들은 캐쉬에서 직접 수행된다. 이 기법은 무효화 명령에 의한 접근 실패는 감소시키지만, 또한 단점을 가지고 있다. 여러 개의 변수를 사용하는 병렬 루프가 순차적으로 두 개 있다고 할 때, 앞의 루프에서 갱신된 후 뒤의 루프에서 곧바로 참조되는 경우에는 무효화할 필요가 없다. 즉, 캐쉬 내에 저장되어 있는 갱신된 내용을 그대로 사용할 수 있다. 그러나, 위의 기법은 그런 상황을 고려하지 않고 무조건 하나의 계산 단위가 끝난 후 갱신되었으면 무효화시키므로 불필요한 적재가 한번 더 일어나는 셈이 된다.

세번째 기법은 임시 지역성을 고려하는 것으로 Min과 Baer[20]에 의해 제안된 타임 스탬프(time stamp) 방식이다. 여기서는 벡터나 행렬 같은 각 데이타 구조마다 하나의 계수기가 있어서 각 계산 단위의 마지막 단계에서 자신과 연관된 변수가 갱신이 된 경우에 자신의 내용을 갱신한다. 캐쉬 내의 각 블럭과 관련된 타임 스탬프는 그 블럭이 캐쉬 내에서 갱신될 때 대응

하는 계수기의 값에 하나를 더한 값을 자신의 최종적인 값으로 취한다. 이 타임 스탬프가 연관된 계수기의 값을 초과하는 경우에는 캐쉬를 참조하는 것이 유효하다. 그렇지 않으면 이것은 메모리로부터 직접 자료를 가져와야 한다.

이 기법은 두개의 계산 단위 사이에서 하나의 프로세서에 지역적으로 사용되는 변수가 있는 경우에는 타임 스탬프의 값이 자신의 값을 초과하기 때문에 불필요한 무효화를 막을 수 있다. 그러나, 이를 위해 다수의 계수기 레지스터와 캐쉬 디렉토리 내의 각 캐쉬 라인에 대한 타임 스탬프로 인한 부가적인 하드웨어가 필요하다.

무효화 실패의 관점에서 볼 때 세번째 기법이 하드웨어를 이용한 기록 무효화 프로토콜과 같은 효율을 보인다. 그러나, 이러한 소프트웨어 기법을 제공하기 위한 하드웨어 지원의 측면에서 보면, 무차별 무효화는 캐쉬를 on-off하는 장치만이 필요하고, 선택적 무효화는 각 캐쉬 라인당 갱신 비트만이 필요한 것에 비해 타임 스탬프 방식은 보다 복잡하고 많은 하드웨어가 필요하다.

순수한 하드웨어 기법의 기록 무효화 프로토콜과 비교해 보면, 소프트웨어를 이용한 프로토콜이 비용 면에서 훨씬 효율적이다. 또한 간단한 예의 관찰 결과는 소프트웨어를 이용한 기법들과 하드웨어를 이용한 기법들의 성능이 비슷함을 볼 수 있다.

Smith[5]에 의해 개발된 기법은 임계 구역과 관련된 모든 공유변수를 다음과 같은 방법을 사용하여 선택적으로 무효화한다. 첫째, 임계 구역 내의 모든 공유변수들은 똑같은 페이지로 할당된다. 그리고 1회용 명명자 (One-Time Idenifer: OTI)가 각 페이지와 연관된다. 하나의 캐쉬 블럭이 캐쉬 내로 적재될 때 상응하는 OTI는 Translation-Lookaside-Buffer(TLB)로부터 캐쉬 라인에 상응하는 캐쉬 디렉토리 내로 적재된다. 캐쉬의 접근이 적중하는 동안 적재된 OTI는 TLB 내의 OTI와 같은 값을 가진다. 그리하여, 임계 구역과 관련된 모든 공유변수의 무효화는 단순히 상응하는 페이지에 대한 OTI만 갱신시킴으로써 빠르게 수행할 수 있다. 이 방식의 가장 큰 특징은 빠르면서도 선택적인 무효화에 있다.

이상과 같은 소프트웨어를 이용하는 기법들은 아직 상업적 시스템에서 사용되고 있지 않으나, Illinois 대학의 Cedar 시스템[4] 등에서 활발히 연구중이다.

4. 캐쉬 일관성의 구현 예

4.1 Stanford DASH

DASH(Directory Architecture for SHared memory)는 스탠포드에서 개발한 확장가능한 NUMA구조의 공유메모리 시스템이다[21,22]. 이 시스템은 확장가능하고, 낮은 지연시간을 갖는 상호연결네트워크를 통해서 연결된 다수개의 다중프로세서 클러스터들로 구성되어 있다. 물리적인 메모리는 여러 클러스터안의 처리노드들 사이에 분산되며 분산되어 있는 메모리가 전체 주소공간을 형성한다. 그림 4.1은 DASH시스템의 구조를 보여주고 있다.

DASH 시스템의 캐쉬 일관성 프로토콜에서 볼 수 있는 특징은 클러스터간에는 상호연결네트워크을 통하여 디렉토리 기반의 일관성 유지 프로토콜을 사용하고, 한 클러스터 내에서는 스누핑 방법을 사용한다. 또한 일관성을 유지하기 위해서 방송을 사용하지 않고 직접적인 메시지 전달 방법을 사용한다. 클러스터간의 연결 구조를 살펴보면 그림 4.2와 같이 처리노드(Silicon Graphics POWER Station 4D/240(340))를 4개의 프로세서와 2-레벨 캐쉬로 구성하고 있다. 노드 사이의 연결은 2×2 메쉬로 구성되어 있는데, 하나는 요구 메시지를 위해 사용하고 다른 하나는 응답 메시지를 위해 사용한다. 한 노드가 4개의 프로세서로 구성되어 있기 때문에 노드를 보통 클러스터라고 부른다. 클러스터 내부에서의 일관성 유지 방법은 버스를 기반으로하는 스누핑 프로토콜을 사용하고 노드사이에는 디렉토리를 기반으로하는 프로토콜을 사용한다. 4D/240(340)의 메모리 버스는 파이프라인된 동기화 버스이며 메모리에서 캐쉬로, 또는 캐쉬에서 캐쉬로의 전송이 가능하다. 노드사이의 일관성은 디렉토리 제어기가 담당한다.

DASH에서 캐쉬 일관성은 무효화를 기반으로 하는 소유권(ownership) 프로토콜로 메모리 블

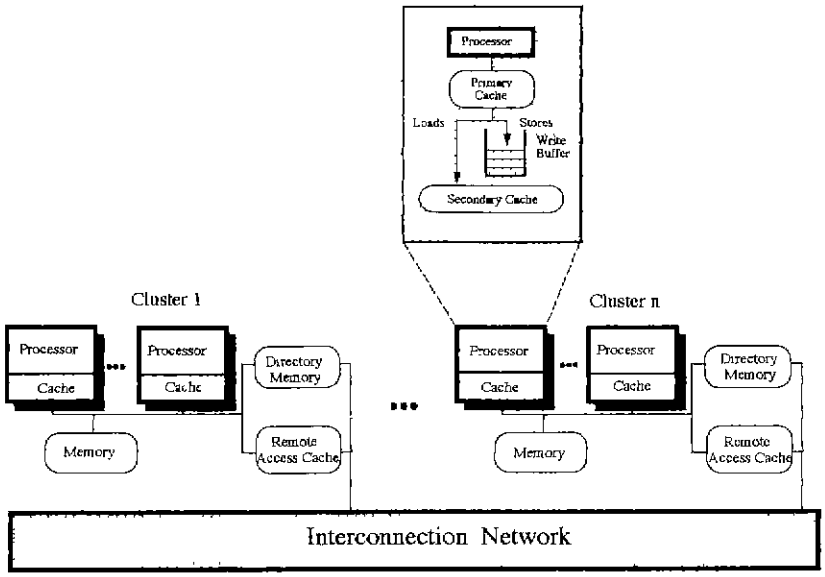


그림 4.1 DASH 시스템의 구조 (Proc. 17th Int'l Symp. Comp. Arch. May, 1990)

럭은 다음과 같은 세가지 상태중에 하나를 갖는다. 즉, 다른 클러스터에 의해 캐쉬되지 않은 Uncached상태, 하나이상의 다른 클러스터에 의해 공유되었으나 갱신되지 않은 Shared상태, 그리고 어떤 다른 하나의 클러스터에 의해 갱신된 Dirty상태가 그것이다. 소유권 프로토콜은 각 블럭의 실질적인 소유자(owner)만이 해당 블럭의 디렉토리 상태를 갱신할 수 있도록 제한하는 것으로, 따라서 소유자가 다른 클러스터에 위치하는 경우에는 명목상으로만 소유자이고, 블럭의 내용을 갱신한 그 클러스터가 실제적인 소유자가 된다. 읽기의 경우, 읽으려는 정보가 그 클러스터에 존재할 경우, 그 블럭이 Shared상태이면 단순히 해당 정보를 읽어오면 되나, Dirty상태이면 그 블럭의 소유자인 캐쉬가 그 내용을 보내고 블럭의 상태를 Shared상태로 바꾸어야하며, 동시에 블럭의 홈(Home)이 그 클러스터이면 메모리에 Write back을 수행하고 아니면, 원격 참조 캐쉬(Remote Access Cache)가 그 블럭을 Shared상태로 가지고 있어야 한다. Dirty상태의 원격 캐쉬 블럭에 대한 읽기 요청에 대한 상태전이도는 그림 4.3(a)과 같다.

그림 4.3(b)는 Shared상태의 원격 캐쉬 블럭에 대한 쓰기동작을 나타내고 있다. 무효화에 근거

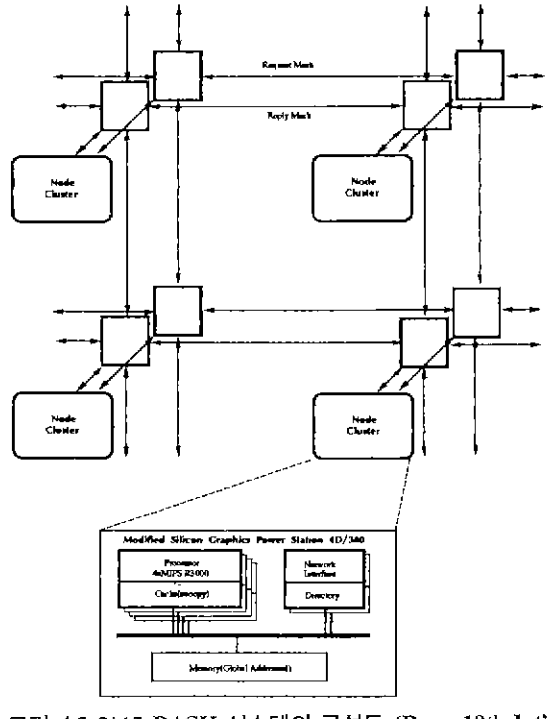
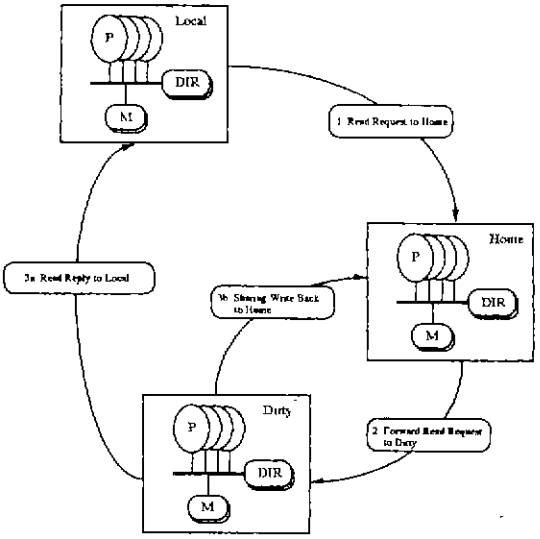
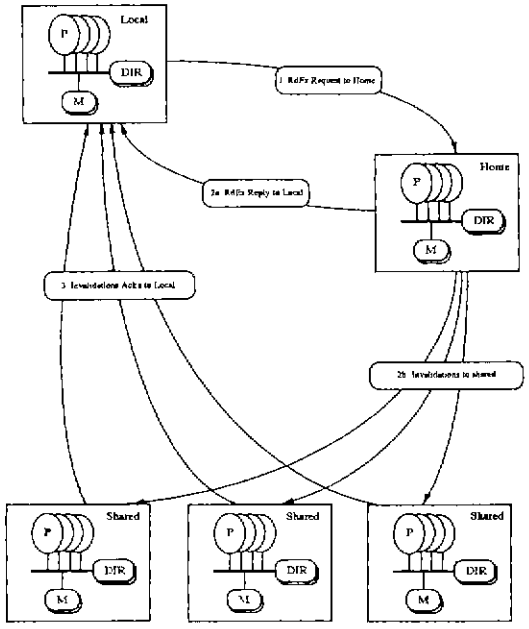


그림 4.2 2×2 DASH 시스템의 구성도 (Proc. 19th Int'l Symp. Comp. Arch. May, 1992)

한 프로토콜은 기록하기 이전에 프로세서가 캐쉬 블럭에 대한 배타적인 소유권을 획득해야만 한



(a) Dirty 상태의 원격 캐쉬 블럭의 읽기



(b) Shared 상태의 원격 캐쉬 블럭의 쓰기

그림 4.3 DASH에서 디렉토리 기반의 캐쉬 일관성 프로토콜의 두가지 예 (Lenoski and Hennessy, 1990)

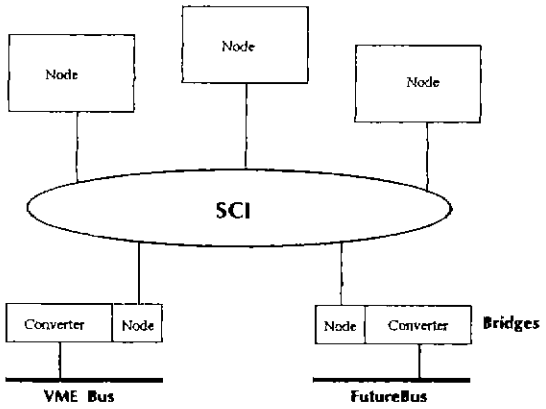
다. 따라서 만일 프로세서가 캐쉬되지 않은 블럭, 또는 Shared상태로 캐쉬된 블럭에 대한 쓰기동작을 요청할 경우, 프로세서는 지역버스에 배타적-읽기(Read-Exclusive:RdEx) 요청(메시지 1)을 발생한다. 이 경우에 다른 캐쉬들은 지역 클러스터안에 Dirty상태의 블럭 엔트리를 가지고 있지 않으므로, RdEx 요청은 홈 클러스터에 보내진다. 앞에서처럼 원격 접근 캐쉬 엔트리는 지역 클러스터에 할당된다. 홈 클러스터에서 Pseudo-CPU는 버스상에 배타적-읽기 요청을 발생한다. 디렉토리는 이 블럭이 Shared상태임을 나타내고 있으므로, 디렉토리 제어기는 지역 클러스터에 RdEx 응답을 보내고, 공유된 클러스터에는 무효화-요청(Inv-Req, 메시지 2b)을 보낸다. 홈 클러스터는 그 블럭을 소유하고 있고, 즉시 디렉토리를 Dirty상태로 갱신한다. 이것은 현재 지역 클러스터가 메모리 블럭의 배타적 사본을 갖고 있음을 나타낸다. RdEx응답 메시지는 응답 제어기에 의해서 지역 클러스터가 수신하게 된다.

4.2 SCI(Scalable Coherence Interface)

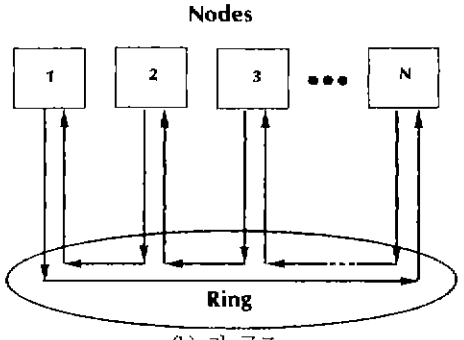
현재의 고성능 병렬처리 시스템의 요구에 부응하여 기존의 버스기반의 백본에서 전이중(full-duplex) 방식의 포인트 투 포인트 인터페이스 명세로 확장하기 위해서 특별한 일관성 상호 연결구조가 필요해졌다. 이러한 인터페이스를 확장가능한 일관성 인터페이스(SCI)라고 부르며, IEEE 표준 1596-1992에 명시되어 있다[14,23]. 현재 SCI는 단방향 포인트 투 포인트 연결만을 지원하고 있으며, 최대 64K개까지의 프로세서, 메모리 모듈, I/O노드들을 연결한 공유된 SCI 상호연결과 효율적으로 인터페이스할 수 있다. SCI에서 캐쉬 일관성 프로토콜은 디렉토리에 근거한 방법을 사용하고 있다. 공유리스트(sharing list)가 참조 목적을 위해 분산된 디렉토리와 서로 연결되어 사용되고 있다.

4.2.1 SCI 상호연결 모델

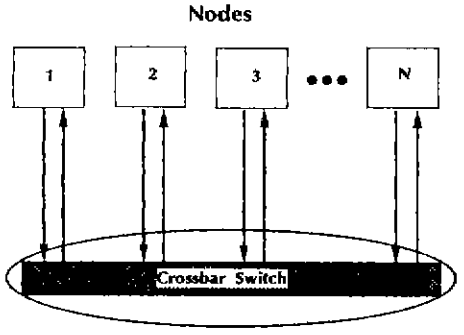
SCI는 노드와 외부 상호연결사이에 인터페이스를 정의하고 있다. 원래 초기의 목적은 링크당 1 Gbyte/s의 대역폭을 갖는 16-bit 링크를 사용하기 위해 개발되었으나, 결과적으로, 백본 버스가 단방향 포인트 투 포인트 링크로 대치하게 되었다. 일반적인 SCI구성이 그림 4.4(a)에 나타나 있다. 각 SCI 노드는 메모리와 I/O장치를 부



(a) 일반적인 SCI 구성



(b) 링 구조



(c) 크로스바 구조

그림 4.4 SCI 상호연결 구조 (IEEE standard 1956~1992)

착한 프로세서가 될 수 있다. 그리고 SCI 상호연결 구조는 그림 4.4(b)와 그림 4.4(c)처럼 링(Ring) 구조와 크로스바 스위치(Crossbar Switch)라고 가정할 수 있다.

각 노드는 SCI 링이나 크로스바와의 사이에 입력링크(input link)와 출력링크(output link)를 가지고 연결되어 있다. 이러한 환경에서는 버스 기반의 전통적인 방송 트랜잭션 개념을 포기하고 있다. 일관성 프로토콜은 요청기(Requester)에 의해서 시작되고 응답기(Responder)에 의해서 종료된다. 링 상호연결 구성은 노드들 사에에 가장 간단한 피드백 연결을 제공한다. 그리고 SCI 링의 대역폭, 중재기법, 주소지정 메카니즘은 백본 버스보다도 월등한 성능을 보여주며, 또한 스누피 캐쉬 제어기를 없앰으로써 가격이 저렴해질 수 있다. 이러한 장점보다 더 중요한 사항은 그 확장성에 있다. 비록 SCI가 확장가능하지만, 캐쉬 디렉토리에 사용된 메모리의 양도 또한 확장이 가능하다. 그러나 공유리스트가 길어지게 되면, 무효화에 많은 시간이 소요되기 때문에 SCI 프로토콜의 성능은 향상되지 않는다. 이러한 문제는 아직 해결해야할 문제점으로 남아 있다.

4.2.2 공유 리스트 구조

공유리스트는 캐쉬 일관성을 위해서 필요한 디렉토리를 체인으로 연결하기 위해서 사용된다. 공유리스트의 길이는 일반적으로 제한이 없고, 각 리스트는 동적으로 생성, 소멸된다. 일관성을 위한 각 캐쉬 블럭은 블럭을 공유하고 있는 프로세서의 리스트상에 등록된다. 프로세서는 지역적으로 저장된 캐쉬에 대하여 일관성 프로토콜을 바이패스하는 옵션을 갖고 있다. SCI는 중앙 디렉토리를 사용함으로써 일어날 수 있는 확장성의 제한을 공유 프로세서들 사이에 디렉토리를 분산하는 방법으로 피할 수 있다. 공유프로세서 사이에 통신은 그림 4.5에서 보여주는 것처럼 매우 밀접하게 공유된 메모리 제어기에 의해 지원된다. 다른 블럭들은 지역적으로 캐쉬되지만, 일관성 프로토콜에서는 보여지지 않는다. 모든 블럭 주소에 대하여, 메모리와 캐쉬 엔트리는 공유 리스트에서 첫 프로세서(head)를 식별하고 이전노드와 다음노드를 연결하는데 사용되는 추가적인 태그 비트를 갖게된다. 각 링크에 양방향 화살표로 표시된 전방향 포인터(forward pointer)와 후방향 포인터(backward pointer)를 갖는. 이중 연결 리스트(double-linked list)가 프로세서 사이에 공유 리스트 형태로 유지될 수 있다. 비일관성(non-coherent)사본은 페이지레벨의 제어에 의해서 일관성 있게 만들 수 있으나, 이러한

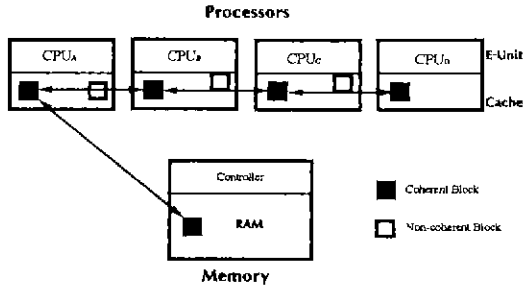


그림 4.5 분산 디렉토리를 갖는 SCI 캐쉬 일관성 프로토콜 (IEEE Computer, June, 1990)

고수준의 소프트웨어 일관성 프로토콜은 현재 발표된 SCI 표준의 범위를 넘어서는 사항이다.

5. 결 론

캐쉬 일관성 유지를 위한 여러가지 기법들은 장단점을 갖고 있기때문에, 구현단계에서 어떤 프로토콜을 사용할 것인가는 여러가지 사항을 고려하여야 한다. 아울러 위에서 살펴본 여러 캐쉬 일관성 정책들은 많은 문제점을 내재하고 있다. 예를 들어 스누피 프로토콜에서 발생할 수 있는 "Ping-Pong Effect"가 대표적인 문제이다. Ping-Pong Effect는 2개이상의 프로세서들이 하나의 공유변수(예, semaphore 또는 lock 변수)에 대해 연속적으로 Read-Modified-Write 사이클을 수행함으로써 다른 프로세서들이 캐쉬의 내용을 무효화 시키고, 그 결과 프로세서들간의 캐쉬 블럭 전송을 빈번하게 발생하여 버스의 데이타 교통량이 급격히 증가하여 연산작업이 지연되는 현상이다. 이러한 현상은 시스템의 utilization을 낮추는 큰 요인이 된다. 따라서 이러한 여러가지 문제를 해결할 수 있는 전략을 강구하여야 한다.

본 고에서 살펴본 캐쉬 일관성 유지를 위한 프로토콜에 대하여 다음과 같은 결론을 내릴 수 있다. 첫째로, 여기 소개된 것들은 스누피 캐쉬 프로토콜을 제외하고는 실제 구현된 것이 없어 실제 시스템에 대한 비교평가가 어렵다는 것이며, 둘째로, 병렬 처리 분야가 이제 막 성숙되어 가는 연구분야이기 때문에 대규모 병렬처리가 가능한 다중 프로세서 시스템이 희소하여 캐쉬 일관성 유지와 관련된 프로토콜이나 전략을 적용하여 실제적인 실험을 하기 어렵다는 점이다. 아울러 캐쉬설계는 아주 크고 복잡하며 많은 trade-off가 존재하는 분야이다.

현재 다중프로세서 시스템에서 캐쉬와 연관되어 진행중인 연구 토픽은 다음과 같다.

(1) 새로운 사유 캐쉬를 지원하는 확장 가능한 구조에 대한 연구.

(2) 효율적인 캐쉬 일관성 유지 문제를 지원하기 위한 하드웨어/컴파일러 trade-off에 관한 연구.

(3) 캐쉬를 사용하는 다중프로세서의 성능 평가.

(4) 신뢰성 있고 효율적인 처리 기간의 동기화 문제에 관한 연구 : 하드웨어와 소프트웨어 동기화 프로토콜의 성능 분석.

(5) 캐쉬를 통한 가상 메모리의 효율적인 구현 방식에 관한 연구 : 가상 메모리 시스템에서, 주소 번역 자체가 캐쉬로 구현되어 있지 않으면, 캐쉬의 속도로 이를 검색할 수 없으므로 캐쉬의 장점을 잃게 된다.

(6) 단순하고 효율적인 병행 모델을 얻기 위한 공유메모리 접근의 순서화 : 각각의 프로세서들이 매우 빠르고, 버스를 사용하지 않은 시스템인 경우, 메모리 접근은 반드시 버퍼를 사용하여야 한다. 그리고, 상호 연결네트워크는 모든 사본들의 단위적인 갱신/무효 등을 지원하지 못한다. 임의의 병행 모델은 하드웨어를 명령어 집합 단계에서 보는 방법을 제공하고, 공유메모리 접근의 전반적인 순서화에 기초한다. 전통적인 병행 모델은 순차적인 일치성을 제공하는 것이며, 다른 모델로는 weak ordering이 정의되어 있다. 이 방법은 기본적으로 명시적인 동기화 기점으로의 접근에 대한 순서를 제한한다.

(7) 프로세서의 효율성을 증가시키기 위한 캐쉬와 그에 관련된 프로토콜의 설계에 관한 연구: 성공률이 비록 높기는 하지만, 공유메모리 접근에 대한 실패와 일관성 유지 행위에 의한 블럭킹 시간, 또는 벌칙에 의하여 프로세서의 효능이 감소하는 경우가 있다. 따라서, 이러한 부담을 최소화하는

캐쉬 프로토콜을 정의하고, 접근이 실패하였을 때 프로세서를 블럭 시키지 않고 여러 번의 접근 실패가 발생한 상태에서도 계속 캐쉬에 접근할 수 있는 블럭킹되지 않는 캐쉬를 설계하는데 중점을 두어 연구가 진행 중이다.

이외에도 기존의 다중프로세서에 대한 관심과 함께, 최근 병렬처리 컴퓨터 구조에 큰 영향을 끼친 고속의 RISC 마이크로 프로세서의 출현과 증가하는 메모리 대역폭은 미래의 다중프로세서 시스템에 점점 더 커다란 부담으로 작용하게 된다. 이러한 이유에 의하여 차세대 다중프로세서에서의 캐쉬에 대한 연구는 병렬처리 컴퓨터 구조 연구 분야에서 더욱 중요한 문제로 부각될 것이다.

참고문헌

[1] Kai Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.

[2] R. Duncan, "A Survey of Parallel Computer Architectures," *IEEE Computer*, Feb., 1990. pp. 5~16.

[3] Per Stenstrom, "A Survey of Cache Coherence schemes for Multiprocessors," *IEEE Computer*, Jun., 1990, pp. 12~24.

[4] Daniel Tabak, *Multiprocessors*, Prentice Hall 1990. pp. 3~22.

[5] Alan Jay Smith, "Cache Memories," *ACM Computing Surveys*, Vol. 14, No. 3, Sep., 1982, pp. 473~530.

[6] James Archibald and Jean-Loup Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. of Computer Systems*, Vol. **4**, Nov., 1986, pp. 273~298.

[7] 전주식, "Bus-oriented 다중 처리 시스템 및 Cache Coherence 문제 해결 방법," 병렬처리 시스템 연구회 하계 단기 강좌, 1990. 7월, pp. 7-1-7-25.

[8] L. Rudolph and Z .Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Architectures," *Proc., 11th Int'l Symp. Computer Architecture*, 1984, pp. 340~347.

[9] S. Eggers and R. Katz, "Evaluating the Performance of Four Snooping Cache Coherence Protocols," *Proc, 16th Int'l Symp. Computer Architecture*, 1989, pp. 2~15.

[10] A. Karlin *et al.*, "Competitive Snoopy Caching", *Proc., 27th Ann. Symp. Foundation of Computer Science*," 1986, pp. 244~254.

[11] A. Agarwal *et al.*, "An Evaluation of Directory Schemes for Cache Coherence," *Proc., 15th Int' l Symp. Computer Architecture*, 1988, pp. 280~289.

[12] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems", *IEEE Trans. on Computer* C-27, Dec., 1978, pp. 1112~1118.

[13] P. Stenstrom, "A Cache Consistency Protocol for Multiprocessors with Multistage Networks," *Proc., 16th Int'l Symp. Computer Architecture*, May, 1989. pp. 407~145.

[14] IEEE, *Scalable Coherent Interface IEEE P1596-SCI Coherence Protocol*, Mar., 1989.

[15] A. W. Wilson, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessor," *Proc., 14th Int'l Symp. Computer Architecture*, 1987, pp. 244~252.

[16] J. Goodman and P. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor," *Proc., 15th Int'l Conf. Computer Architecture*, 1988, pp. 422~431.

[17] S. Haridi and E. Hagersten, "The Cache Coherence Protocol of the Data Diffusion Machine," *Proc., PARLE89*, Vol 1, Springer-Verlag, 1989, pp. 1~18.

[18] H. Choeng and Alexander V. Veidenbaum. "Compiler-Directed Cache Management in Multiprocessors", *IEEE Computer*, Jun., 1990. pp. 39~47.

[19] H. Cheong and A. Veidenbaum, "A Cache Coherence Scheme With Fast Selective Invalidation", *Proc., 15th Int'l Symp. Computer Architecture*, 1988. pp. 299~307.

[20] 민상렬, "A Timestamp-based Cache Coherence Scheme", 병렬처리 시스템 초청 강연 요약집, 한국 정보 과학회, 1990, 11월, pp. 5~33.

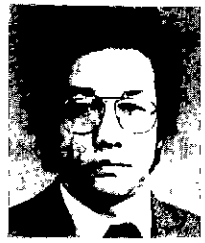
[21] D. L. Lenoski and J.Laudon, *et al.*, "The Directory-Based Cache Coherence Protocol for the

DASH Multiprocessor," *Proc., 17th Int'l Symp. Computer Architecture*, 1990. pp. 148~159.

[22] D. L. Lenoski and J. Laudon, *et al.*, "The DASH Prototype: Implementation and performance," *Proc., 19th Int'l Symp. Computer Architecture*, 1992. pp. 92~103.

[23] D. V. James, A. T. Laundrie, *et al.*, "Scalable Coherent Interface," *IEEE Computer*, Jun., 1990, pp. 74~77.

김 성 천



1975 서울대학교 공과대학 공업교육학(전기전공) 학사
1976 ~1977 동아컴퓨터(주) Sys. Eng.
1977 ~1978 스페리 유니백 Sales Rep.
1982 ~1984 캘리포니아주립대 조교수
1984 ~1985 금성반도체(주) 책임연구원
1961 ~1989 서강대학교 공과대학 전자계산소 부소장
1989 ~1991 서강대학교 공과대학 전자계산학과 학과장
1985 ~현재 서강대학교 공과대학 전자계산학과 교수
1989 ~현재 한국정보과학회 병렬처리시스템 연구회 위원장
대한전자공학회 및 한국통신학회 논문지 편집위원
관심 분야 : 병렬처리 시스템

박 병 섭

1989 충북대학교 공과대학 컴퓨터공학과 학사
1991 ~서강대학교 공과대학 전자계산학과 석사
1991 ~현재 서강대학교 공과대학 전자계산학과 박사과정
관심 분야 : 병렬처리 시스템