수 평형 이진트리를 이용한 디렉토리 캐쉬 일관성 유지 기법

서 대 화†

요 약

디렉토리 기반의 캐쉬 일관성 유지 기법은 대규모 공유메모리 다중처리기에서 캐쉬 일관성 문제를 해결하기 위한 방법이다. 이제까지 기존 기법들은 디렉토리를 위해서 많은 메모리 요구와 긴 무효화 시간, 네트웍 통신량의 집중, 그리고 낮은 확장성 등의 문제점들을 가지고 있다. 본 논문에서는 이런 문제점을 해결하면서 확장성을 가진 새로운 디렉토리 기반의 캐쉬 일관성 유지 기법을 제안하였다. 이 기법에서는 새롭게 제안한 수 평형 이진 트리를 사용하였다. 수 평형 이진 트리는 디렉토리에 있는 노드 수에 따라 모양이 일정하고, 최대 깊이가 $\lfloor \log_2 n \rfloor$이며, 같은 노드 개수를 가진 이진 트리 중에서 가장 작은 단말 노드 수를 가지는 특성이 있다. 이러한 특성은 캐쉬 디렉토리의 메모리량, 무효화 시간 및 네트웍 통신량을 최소로 해 주고, 다중처리기의 높은 확장성을 보장해 준다.

Directory Cache Coherence Scheme using the Number-Balanced Binary Tree

Dae-Wha Seo†

ABSTRACT

The directory-based cache coherence scheme is an attractive approach to solve the cache coherence problem in a large-scale shared-memory multiprocessor. However, the existing directory-based schemes have some problems such as the enormous storage overhead for a directory, the long invalidation latency, the heavy network congestion, and the low scalability. For resolving these problems, we propose a new directory-based cache coherence scheme which is suitable for building scalable, shared-memory multiprocessors. In this scheme, each directory entry for a given memory block is a number-balanced binary tree(NBBT) structure. The NBBT has several properties to efficiently maintain the directory for the cache consistency such that the shape is unique, the maximum depth is $\lfloor \log_2 n \rfloor$, and the tree has the minimum number of leaf nodes among the binary tree with n nodes. Therefore, this scheme can reduce the storage overhead, the network traffic, and the invalidation latency and can ensure the high scalability in the large-scale shared-memory multiprocessors.

1. 서 론

공유메모리 다중처리기는 캐쉬를 포함하는 여러 프로세서들과 공유메모리 모듈들이 상호 연결망에 접속된 형태이다. 이러한 시스템은 프로세서들 사이에 코드와 자료의 공유가 용이하다. 하지만 하나의 자료를

† 정 회 원:경북대학교 전자전기공학과
논문접수:1996년 4월 8일, 심사완료:1997년 2월 24일

여러 프로세서가 자신의 캐쉬에 복제를 두고 사용하기 때문에 캐쉬 일관성 문제를 야기시킬 수 있다. 캐쉬에 있는 복제들간의 일관성을 유지시켜 주기 위해서는 한 프로세서가 자신의 캐쉬에 있는 복제를 수정할 때, 메모리 블록과 다른 캐쉬에 있는 복제를 더 이상 사용할 수 없게 하거나, 수정된 내용이 수정되는 즉시 이를 변경시켜 주는 방법이 필요하다.

캐쉬 일관성 유지 기법은 크게 스누프 캐쉬 일관성 유지 기법[8]과 디렉토리 기반 캐쉬 일관성 유지 기법[1, 4, 9, 11]으로 나누어진다. 대부분 버스 기반 다중처리기에서는 스누프 캐쉬 일관성 유지 기법을 사용한다. 그러나 버스를 사용한 다중처리기에서는 프로세서들간 통신이 방송 방식으로 시스템의 프로세서 수가 최대 20~30개로 제한된다. 수백 개 이상의 프로세서들로 구성되는 공유메모리 다중처리기인 경우, 방송 방식은 캐쉬 사용을 통한 네트웍 대역폭 축소가 불가능하고, 구현 시 많은 추가 경비가 요구된다.

따라서 다단계 상호연결망 구조의 공유메모리 다중처리기에서는 방송 방식을 사용하지 않는 캐쉬 일관성 유지 기법이 요구된다. 이러한 배경에서 디렉토리 기반의 캐쉬 일관성 유지 기법이 제안되었다. 이 기법에서는 메모리의 특정 영역인 디렉토리가 각 메모리 블록의 복제가 있는 캐쉬를 포함하는 프로세서의 위치와 각 복제의 상태 정보를 가지게 하여, 복제의 수정이 일어 났을 때, 복제를 가진 캐쉬에만 무효화 메시지나 수정된 값을 보낸다.

기존의 디렉토리 기반의 캐쉬 일관성 유지 기법들은 크게 완전사상(full-mapped) 디렉토리 기법[2, 11], 제한(limited) 디렉토리 기법[3, 4], 연결(chained) 디렉토리 기법[5, 6, 11]으로 분류할 수 있다. 이들 기법들은 효용성 측면에서 여러 문제들을 가지고 있다. 완전사상 디렉토리 기법과 제한 디렉토리 기법에서는 디렉토리가 메모리에 구현 되기 때문에 모든 복제 요구가 메모리에서 순차화되고 상호연결망의 통신이 메모리에 집중되는 현상이 있다. 특히 완전사상 디렉토리 기법에서는 프로세서 수가 증가함에 따라 디렉토리로 사용되는 메모리량도 증가된다. 연결 디렉토리 기법은 디렉토리 수정 등 캐쉬 일관성 유지를 위한 프로토콜이 복잡하며 쓰기 실패로 인한 무효화 처리에 많은 시간이 소요되어 시스템 성능을 저하시킨다.

이러한 문제들을 해결하기 위하여 새로운 디렉토리 기반의 캐쉬 일관성 유지 기법인 수 평형 이진트리(number-balanced binary tree, NBBT)를 이용한 BIND 기법이 제안되었다[7]. BIND 기법에서는 디렉토리가 메모리와 캐쉬에 분산되어 있으며, 각 디렉토리 엔트리는 디렉토리를 효율적으로 관리해 줄 수 있도록 수 평형 이진트리 구조를 가지고 있다. n개의 노드를 가지는 수 평형 이진트리는 한가지 모양만 존재하며, 트리의 깊이는 $\lfloor \log_2 n \rfloor$이고, 반완전 이진트리(semi-full binary tree)이다. 이 구조의 디렉토리는 메모리량과 통신량 집중을 줄일 수 있고, 확장성이 용이하여 대규모 다중처리기의 캐쉬 관리를 쉽게 해 준다.

본 논문의 2장에서는 기존의 디렉토리 기반의 캐쉬 일관성 유지 기법의 장단점을 비교하고, 3장에서 BIND 기법의 기본 모델과 디렉토리 관리를 위해 본 논문에서 제안한 수 평형 이진트리의 특성에 대해서 기술한다. 4장에서는 BIND 기법과 기존 기법과의 비교 분석 결과를 기술하고, 5장에서 결론과 향후 연구 과제에 대하여 기술한다.

2. 디렉토리 기반 캐쉬 일관성 유지 기법들의 비교

디렉토리 기반의 캐쉬 일관성 유지 기법은 Tang이 처음 제안하였다[9]. 이 기법에서는 메모리가 모든 캐쉬 상태 정보의 복제를 가진다. 즉, 메모리 모듈이 복제되는 단위인 각 메모리 블록의 복제를 가진 캐쉬를 찾아 준다. 이후 이 기법을 개선하는 여러 디렉토리를 기반으로 하는 기법들이 제안되었다.

완전사상 디렉토리 기법은 각 디렉토리 엔트리에 시스템에 있는 각 캐쉬에 대응하는 존재 비트(presence bit)를 두고, 메모리 블록의 복제가 어느 프로세서의 캐쉬에 저장되어 있는지를 나타낸다. 디렉토리의 각 엔트리에는 특정 프로세서에게 복제를 수정할 수 있는 권한이 부여된 것을 나타내는 수정 비트가 있다. 이 비트가 1로 되어 있으면 오직 하나의 존재 비트만이 1이 되고, 그 프로세서만 복제를 수정할 수 있다. 이 기법은 성능 측면에서 다른 기법들보다도 우수하지만 디렉토리를 구현하기 위해서 많은 양의 메모리가 필요하며, 네트웍의 통신량이 집중되는 현상이 있다. 그리고 프로세서의 수가 증가하게 되면 공유메모리량도 비례해서 증가하기 때문에, 디렉토

리 엔트리의 크기는 프로세서 수에 비례해서 증가하게 된다. 이러한 단점으로 인해서 완전 사상 디렉토리 기법은 많은 수의 프로세서들로 구성되는 시스템에서는 사용하지 않고 있다.

제한 디렉토리 기법은 메모리 블록의 복제 수를 제한하여 디렉토리 크기를 일정하게 유지하는 기법이다. 각 메모리 블록에 대한 디렉토리 엔트리가 i개의 포인터를 가지고, 각 포인터가 블록의 복제를 가지고 있는 캐쉬를 지정한다. 이 기법은 대부분의 병렬 응용에서 특정 메모리 블록에 대한 여러 프로세서의 읽기 요청 수가 제한적이라는 사실에 근거한다. 이 기법은 디렉토리 엔트리 당 메모리량은 일정하고, 전체 디렉토리를 위한 메모리량은 프로세서 수에 비례하고, 포인터 수가 고정되어 확장성이 뛰어나다. 그러나 포인터 수보다 많은 읽기 요청이 있으면 시스템 성능 저하를 야기시키므로 제한 디렉토리 기법의 성공 여부는 자료의 공유 정도 즉, 하나의 자료를 공유하는 프로세서 수에 달려 있다.

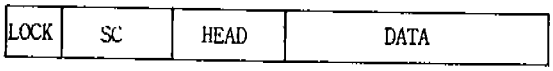
연결 디렉토리 기법은 복제를 가진 캐쉬들을 포인터로 연결시키는 방법으로 프로세서가 메모리 블록의 읽기를 원하면, 그 프로세서 주소를 연결 리스트에 추가로 연결한다. 그리고 프로세서가 복제에 쓰기 요청을 하면, 같은 복제를 가지고 있는 연결 리스트에 있는 모든 캐쉬 엔트리를 무효화시킨다. 포인터의 크기는 시스템에 있는 프로세서 수의 대수(logarithmic)비로 증가하므로 확장성이 뛰어 나다. 하지만 연결 리스트 유지를 통한 일관성 유지 프로토콜이 복잡하고, 쓰기 실패가 발생했을 때 연결 리스트를 차례로 찾아가며 무효화시켜야 되므로 시간이 많이 걸린다. 연결 리스트를 사용한 사례로 SCI(scalable coherent interface)와 SDD(Stanford distributed-directory protocol)가 있다. SCI는 이중연결 리스트를 사용한 기법을 사용하고, ANSI와 IEEE의 표준으로 채택되었다[3]. SDD는 단일연결 리스트를 사용한 기법으로 각 캐쉬엔트리가 하나의 포인터를 가지고 있다.

최근에는 연결 디렉토리 기법의 변형으로 트리 구조를 이용한 디렉토리 구성 방안에 대한 연구가 많이 진행되고 있다. AVL 이진트리를 이용한 Wisconsin STEM(tree merging extension to SCI)도 트리 구조의 한 예이다[5]. 하지만 이 트리구조 기법은 새로운 엔트리의 삽입이나 캐쉬 대체 시에 AVL 이진트리의 유지를 위한 추가부담이 너무 크고, 프로토콜 또한 매우 복잡한 단점을 가지고 있다. 특히 캐쉬 엔트리를 대체할 때 많은 시간이 걸리므로 시스템 성능 저하가 야기되는 문제가 있다.

3. 수 평형 이진트리 기법

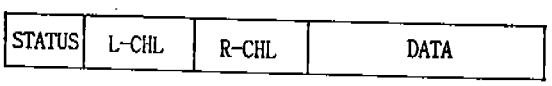
3.1 기본 구조

이 장에서는 수 평형 이진트리를 기반으로 하는 새로운 디렉토리 기반의 캐쉬 일관성 유지 기법인 BIND (Number Balanced bINary-Tree Directory-based scheme) 기법에 대하여 기술한다. BIND 기법은 독자적인 캐쉬를 보유한 프로세서들과 공유메모리로 구성된 다중처리기를 위한 캐쉬 일관성 유지 기법이다. 캐쉬에 복제되는 단위인 메모리 블록은 잠금 비트(L), 공유 계수기(SC) 필드, 헤드 필드 및 자료 필드로 되어 있다(그림 1).



(그림 1) 메모리 블록의 구성
(Fig. 1) Organization of a memory block

잠금 비트는 쓰기나 대체 작업 동안 정확한 동작을 보장하기 위해서 사용하고, SC 필드는 복제된 수를 가지며, 헤드 필드는 루트 캐쉬(root cache)의 위치를 포함한다. 루트 캐쉬는 메모리 블록에 대해서 첫번째 읽기 접근을 시도한 프로세서의 캐쉬를 의미한다. 루트 캐쉬는 쓰기나 대체로 변경된다. 모든 캐쉬는 (그림 2)과 같은 캐쉬 엔트리들의 집합으로 구성된다. 각 캐쉬 엔트리는 상태 필드, 두 개의 자식 필드(L-CHL, R-CHL)와 복제 자료를 저장하고 있는 자료 필드로 구성되며, 자식 필드는 동일한 메모리 블록의 복제를 가지고 있는 캐쉬를 지정하고, 상태 필드는 캐쉬 일관성 유지를 위해 필요한 정보를 포함한다.

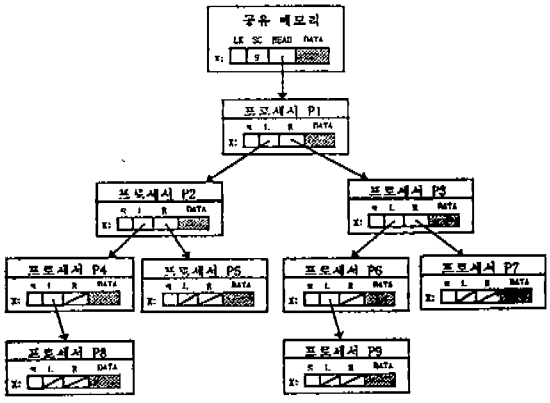


(그림 2) 캐쉬 엔트리 구성
(Fig. 2) Organization of a cache entry

3.2 BIND 기법

BIND 기법은 분산 디렉토리 구조에 디렉토리 정보를 메모리와 캐쉬 엔트리에 분산시켜 관리한다. 메모리 블록의 헤드 필드와 캐쉬 엔트리의 L-CHL과 R-CHL 필드는 블록의 복제를 가지고 있는 캐쉬의 위치를 지정하며 이들의 크기는 $\log_2 N$비트이고, 여기서 N은 시스템에 있는 프로세서의 총 수를 의미한다. 각 메모리 블록의 복제를 가지는 캐쉬 엔트리들은 (그림 3)과 같이 수 평형 이진 트리(NBBT, Number-Balanced Binary Tree) 구조로 유지된다. 수 평형 이진 트리의 특성을 다음 절에 상세하게 기술한다.

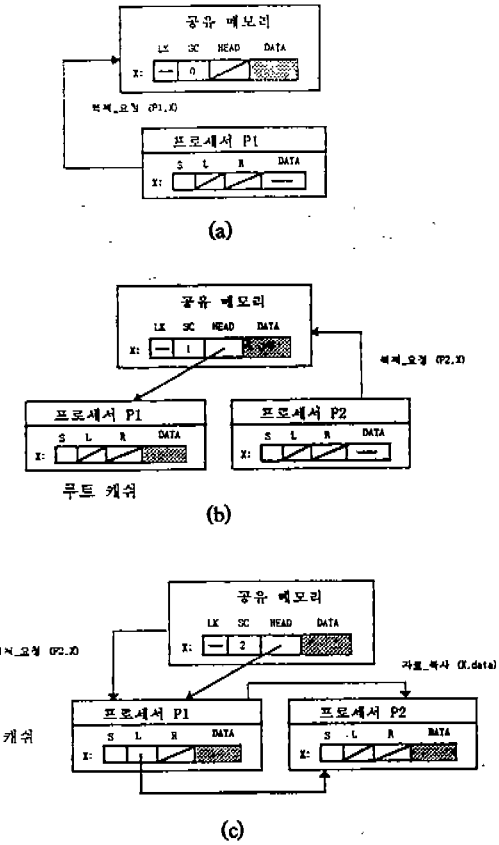
메모리 블록에 있는 $\log_2 N$ 비트의 공유 계수기(SC)는 메모리 블록이 복제된 수로, 새로운 읽기 요청 시 새롭게 복제를 가지게 된 캐쉬가 디렉토리(NBBT)에 등록될 때 등록 위치를 찾는데 사용하고, 쓰기 요청 시 복제 무효화 과정에서 사용한다.



(그림 3) BIND 기법에서의 디렉토리 구조
(Fig. 3) Directory structure in the BIND scheme

3.2.1 읽기 실패 (read miss)

프로세서가 캐쉬로부터 읽기 동작이 실패하게 되면, 메모리 모듈에 원하는 블록의 복제를 요구하게 된다. 메모리 모듈은 메모리 블록의 헤드 필드가 비어 있으면 비어 있는 헤드 필드에 요청한 캐쉬의 위치를 기록하고 블록의 복제를 요청한 프로세서에게 전송한다. 그렇지 않고 메모리 블록의 헤드 필드가 루트 캐쉬를 지정하고 있으면, 루트 캐쉬가 복제 요구를 처리하도록 하고 복제를 요구한 캐쉬를 이진 트리에 연결시킨다. (그림 4)는 읽기 요청에 따른 수 평형 이진 트리 구조의 캐쉬 디렉토리 형성 과정을 보여 준다. (그림 4(a)) 경우와 같이 메모리 블록 X에 대한 블록의 복제가 없으면 SC의 초기값은 0 이다. 메모리 모듈이 캐쉬 C_1로부터 복제_요청 메시지를 받으면, SC를 1 증가시키고 헤드 필드가 캐쉬 C_1를 지정하도록 한 후 (그림 4(b))처럼 블록 X의 복제를 캐쉬 C_1으로 전송하게 된다. 즉, 캐쉬 C_1이 블록 X의 루트 캐쉬가 되는 것이다. 루트 캐쉬는 이후 발생하는 모든 복제 요청에 대하여 블록의 복제를 제공하게 된다. 메모리 모듈이 캐쉬 C_2로부터 메모리 블록 X의 복제_요청 메시지를 받으면, SC를 1 증가시키고(SC=2), 루트 캐쉬 C_1에게 적재_요청 메시지를 전송한다. 메시지는 블록의 주소, 요청한 캐쉬의 위치 및 경로 정



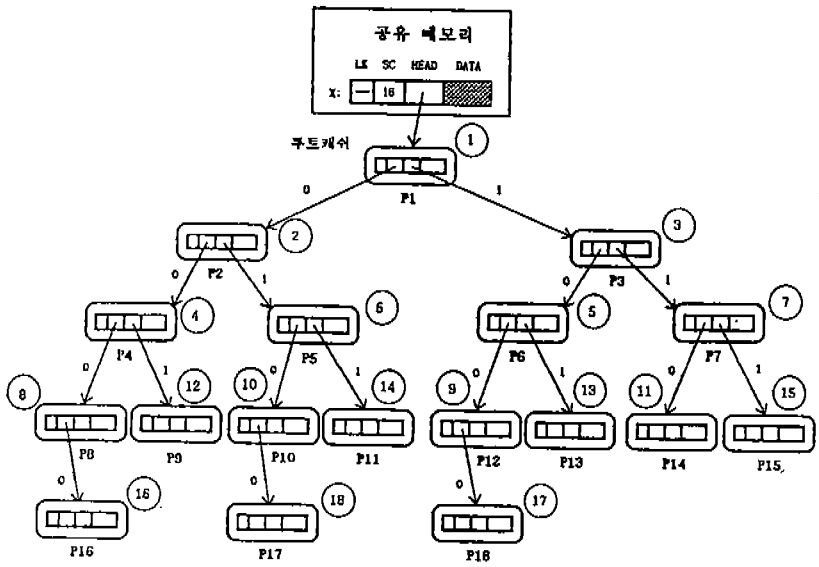
(그림 4) 디렉토리의 구축 과정
(Fig. 4) Construction of the directory

보(routing information, RI)를 가진다. RI는 새로운 복제를 가질 캐쉬가 수 평형 이진 트리에 접속될 위치로 찾아가는 경로 정보로 초기값은 메모리 블록의 SC가 된다. 루트 캐쉬 C_1는 블록 X의 복제를 캐쉬 C_2에게 보내 주고 자신의 캐쉬 엔트리의 L-CHL에 RID를 넣어 준다(그림 4(c)).

캐쉬 C_n으로부터 블록 X에 대한 복제_요청 메시지가 들어오면, 메모리 모듈은 해당 블록의 SC를 1 증가시키고, 적재_요청 메시지를 루트 캐쉬 C_1에게 보낸다. 즉, SC가 18일 때 19(이진수로 00010011)로 만들어 C_1에게 적재_요청 메시지를 보낸다. 루트캐쉬가 적재_요청 메시지를받으면, 블록 X의 복제를 캐쉬 C_n에게 보내고, RI의 LSB에 따라서 다음 자식 캐쉬 노드에 연결 메시지를 보낸다. RI의 초기값은 메모리에서 전송 받은 SC값이 된다. 연결 메시지에는 요청한 블록의 주소, 요청한 캐쉬의 위치와 초기의 RI를 1비트 왼쪽 이동시킨 값인 수정된 RI를 포함한다. RI의 LSB를 사용해 자식 포인터를 선택한다. 만약 선택된 자식포인터가 공백이면 그 포인터가 요청 캐쉬 C_n을 지정하도록 해 준다. LSB가 0 이면 요청 캐쉬가 루트캐쉬의 왼쪽 자손이고, 1 이면 오른쪽 자손이 된다. 읽기 요청에 의해서 만들어지는 이진트리의 노드 접속 순서는 (그림 5)와 같다.

3.2.2 쓰기 요청 (write request)

쓰기 요청인 경우에는 캐쉬의 요구를 들어 주기 전에 이진트리 디렉토리에 있는 모든 캐쉬 엔트리를 무효화 시킨다. 물론 복제 수가 1인 경우 즉, SC가 1인 경우는 쓰기를 위한 추가적인 작업이 필요 없게 된다. 쓰기가 발생하여 메모리 블록을 공유하고 있는 캐쉬 엔트리들 중 하나의 캐쉬 엔트리가 변경(update)이 되면 나머지 복제들은 무효화를 시킨다. 이 때 무효화 메시지를 받은 캐쉬 엔트리는 자신을 무효화하고 자신의 자식 노드들에게 무효화 메시지를 보내주게 된다. 자식이 없는 노드는 자신을 무효화한 후 무효_완료_응답 메시지를 루트캐쉬에게 보낸다. 루트캐쉬는 단말 노드의 수만큼의 무효_완료_응답 메시지를 받으면 무효화 과정을 종료하게 된다. 단말 노드의 수는 SC값으로 계산해 낼 수 있다. 계산 방법은 다음 장에서 자세히 설명할 것이다.



(그림 5) NBBT에서 읽기 공유로 발생되는 캐쉬 엔트리의 연결 순서

(Fig. 5) Connecting sequence of read-sharing caches in the NBBT

3.2.3 대체 (replacement)

. 이진트리 디렉토리에 연결된 캐쉬 엔트리 중 대체가 발생하게 되면 그 엔트리를 최종적으로 연결된 캐쉬 엔트리로 대체함으로써 계속적으로 수 평형 이진 트리를 유지할 수 있게 된다. 최종적으로 연결된 캐쉬 엔트리는 SC를 RI로 사용하면 쉽게 찾을 수 있다.

3.3 수 평형 이진트리의 특성

이 절에서는 캐쉬 일관성 유지를 위해서 새롭게 제안된 수평형 이진트리에 특성에 대해서 살펴보기로 한다. 수 평형 이진 트리는 다음과 같이 순환적으로 표현할 수 있다.

정의 3.1 (NBBT) 공백 트리는 수 평형(number balance)이다. T가 좌우의 부분 트리로 T_L과 T_R을 가지는 공백이 아닌 이진트리일 때, T가 수 평형이면

(i) T_L과 T_R이 수 평형이고

(ii) n_L이 n_R과 같거나 1 크다.(즉, $n_L=n_R+1$ 혹은 $n_L=n_R$) 여기서 n_L과 n_R은 T_L과 T_R의 노드 수이다.

정리 3.1 (유일성) n개의 노드를 가지는 수 평형 이진 트리는 유일(unique)하다.

증명 : $n=1$일 때는 명백하게 성립한다. $i \le n-1$인 i개 노드를 가지는 모든 NBBT에 대해서도 성립한다고 가정하면, $i=n$에 대해서도 성립함을 보이면 된다. T가 n개의 노드를 가지는 NBBT이고 T_L과 T_R이 T의 좌우 부분트리(subtree)라고 하면, T_L과 T_R의 노드 수는 유일한 값을 가진다.

즉, n이 홀수이면 $n_L=n_R=\frac{n-1}{2}$ 이 되고, n이 짝수이면 $n_L=\frac{n}{2}$, $n_R=\frac{n}{2}-1$이 된다. 그리고 정의 3.1에 의해서 T_L과 T_R이 NBBT임을 알 수 있다. 그러므로 귀납법에 의해서 T_L과 T_R은 유일하다. T_L과 T_R가 유일하므로 T_L과 T_R을 좌우 부분 트리로 가지는 NBBT는 역시 유일하다. □

정리 3.1을 이용하면 추가될 캐쉬 엔트리가 NBBT에 연결될 위치를 쉽게 결정할 수 있다. NBBT에 있는 노드 수인 SC의 값을 이미 알고 있기 때문에, 새롭게 연결할 캐쉬 엔트리의 위치를 쉽게 찾을 수 있다.

정리 3.2 (깊이) T가 n개 노드를 가지는 수 평형 이진 트리라고 하면, T의 깊이는 $\lfloor \log_2 n \rfloor$이 된다.

증명 : $n=1$에서는 성립한다. 즉, 1개 노드를 가지는 NBBT의 깊이는 0이다. $i \le n-1$인 i개 노드를 가진 모든 NBBT에서 성립한다고 가정을 하면, $i=n$에서도 성립함을 보여 주면 된다. T가 n개의 노드를 가지고, T_L과 T_R이 T의 좌우 부분트리(subtree)라고 하면, T의 깊이, D_T는 다음과 같이 표현된다.

$$D_T = \max(D_{T_L}, D_{T_R}) + 1$$
$$= \max(\lfloor \log_2 n_L \rfloor, \lfloor \log_2 n_R \rfloor) + 1$$

정의 3.1에 의해서 T_L과 T_R의 노드 수는 특정 값을 가진다. 즉, n이 홀수이면 $n_L=n_R=\frac{n-1}{2}$ 이고, n이 짝수이면 $n_L=\frac{n}{2}$, $n_R=\frac{n}{2}-1$이 된다.

(i) n이 홀수인 경우, $n_L=n_R=\frac{n-1}{2}$이기 때문에,

$$D_T = \lfloor \log_2 \frac{n-1}{2} \rfloor + 1 = \lfloor \log_2 (n-1) - 1 \rfloor + 1$$
$$= \lfloor \log_2 (n-1) \rfloor$$가 된다.

n가 홀수이면 $\lfloor \log_2 (n-1) \rfloor = \lfloor \log_2 n \rfloor$이므로 D_T는 $\lfloor \log_2 n \rfloor$가 된다.

(ii) n이 짝수인 경우에,

$n_R=\frac{n}{2}$ 그리고 $n_R=\frac{n}{2}-1$이기 때문에,

$$D_T = \lfloor \log_2 \frac{n}{2} \rfloor + 1 = \lfloor \log_2 n - 1 \rfloor + 1$$
$$= \lfloor \log_2 n \rfloor$$가 된다.

(i)과 (ii)로부터, D_T는 $\lfloor \log_2 n \rfloor$가 된다. □

정리 3.2에서, 트리가 N개 노드를 가지면 탐색을 $O(\log_2 N)$에 할 수 있다. 동시에 새로운 노드를 추가하거나 노드를 삭제하는데도 $O(\log_2 N)$에 할 수 있다.

역시 메모리 블록의 복제의 변경이 요구될 때, 캐쉬된 모든 복제들의 무효화를 위한 최대 시간이 $O(\log_2 N)$가 된다.

정의 3.2 (반포화 이진트리) 깊이 D인 이진트리 T에서, 깊이 D에 있는 모든 노드를 없앤 트리가 포화 이진트리(full binary tree)가 되면, 이진트리 T를 반포화 이진트리(semi-full binary tree)라고 한다.

이진트리가 항상 포화 이진트리로 유지된다면 평균 및 최대 탐색 시간을 최소화 할 수 있다. 반포화 이진트리도 이와 동일한 특징을 가진다.

보조 정리 3.1 T를 수 평형 이진트리라고 하면, T는 반포화 이진트리이다.

증명: $n \geq 1$에 대해서, n개 노드를 가지는 NBBT의 깊이는 $\lfloor \log_2 n \rfloor$이라는 사실은 NBBT가 반포화 이진트리임을 의미한다. □

정리 3.3 n개 노드를 가지는 수평형 이진트리는 n개 노드를 가지는 반포화 이진트리 중에서 가장 작은 단말 노드를 가진다.

증명: $n=1$일 때는 명백하게 성립된다. $i \leq n-1$인 i개의 노드를 가지는 모든 NBBT에 대해서 이 정리가 성립된다고 가정하면, $i=n$에서도 이 가정이 성립함을 보이면 된다. $n-1$개 노드의 NBBT T'에 노드 v를 추가해서 n개 노드를 가지는 NBBT T를 만들었다고 하면, 가정에 의해서 T'은 $n-1$개 노드를 가지는 반포화 이진트리 중에서 가장 작은 단말 노드 수를 가진다. T가 NBBT로 되기 위해서는, 깊이 $\lfloor \log_2 n \rfloor - 1$에 단말 노드가 있으면 v가 단말 노드 중의 하나의 자식 노드로 연결되어야 하고, 아니면 v가 깊이 $\lfloor \log_2 n \rfloor - 1$에 있는 비단말 노드에 연결되어야 한다. 즉, T'의 깊이 $\lfloor \log_2 n \rfloor - 1$에 단말 노드가 있으면 T의 단말 노드 수 NF_T는 T'의 단말 노드 수 $NF_{T'}$와 같고, 아니면 $NF_T = NF_{T'} + 1$이다. 그러므로 T의 단말 노드 수는 n개의 노드를 가지는 반포화 이진트리 중에서 가장 작다. □

정리 3.4 T가 n개 노드를 가지는 NBBT라고 하면, 단말 노드 수 NF_T는 다음과 같다.

$$NF_T = \begin{cases} 2^{\lfloor \log_2 n \rfloor - 1} & \text{if } n \leq \frac{3}{2}\ 2^{\lfloor \log_2 n \rfloor} - 1, \\ n - 2^{\lfloor \log_2 n \rfloor + 1} & \text{if } n > \frac{3}{2}\ 2^{\lfloor \log_2 n \rfloor} - 1. \end{cases}$$

증명: T가 n개 노드를 가지는 반포화 이진트리들 중에서 가장 최소의 단말 노드를 가지므로, 깊이 $\lfloor \log_2 n \rfloor$에서 노드 수가 깊이 $\lfloor \log_2 n \rfloor - 1$에서 노드 수보다 크면 NF_T는 깊이 $\lfloor \log_2 n \rfloor - 1$에서 노드 수와 같고, 크지 않으면 NF_T는 깊이 $\lfloor \log_2 n \rfloor$에 있는 노드 수와 같다. 깊이 $\lfloor \log_2 n \rfloor$에서 노드 수는 $n - (2^{\lfloor \log_2 n \rfloor} - 1)$이 되며, 깊이 $\lfloor \log_2 n \rfloor - 1$에서 노드 수는 $2^{\lfloor \log_2 n \rfloor - 1}$이 된다.
그러므로, $n - (2^{\lfloor \log_2 n \rfloor} - 1) \leq 2^{\lfloor \log_2 n \rfloor - 1}$이면 $NF_T = 2^{\lfloor \log_2 n \rfloor - 1}$이고, 아니면 $NF_T = n - (2^{\lfloor \log_2 n \rfloor} - 1)$이다. □

정리 3.3과 정리 3.4를 이용하면 메모리 모듈은 모든 복제의 무효화를 위한 무효_완료_응답 메시지의 수를 쉽게 계산할 수 있다. 메모리 모듈은 모든 무효_완료_응답 메시지가 도착하면 쓰기 요청을 한 캐쉬에게 쓰기_허용 메시지를 보내 주게 된다. 그래서 무효화 절차를 위해서 메모리 모듈은 SC 값만 관리하면 된다.

4. 기존 디렉토리 기법과의 비교

이 장에서는 기존의 디렉토리 기반의 캐쉬 일관성 유지 기법인 완전사상 디렉토리 기법, 제한 디렉토리 기법, 연결 디렉토리 기법(SCI), 그리고 트리 구조의 디렉토리를 사용하는 Wisconsin STEM 기법[5]과 BIND 기법을 메모리 요구량, 네트웍 통신량, 무효화 시간, 확장성, 프로토콜의 복잡도 측면에서 비교한다. 메모리 요구량은 디렉토리 구축을 위해 요구되는 메모리 량을 의미하고, 네트웍 통신량은 쓰기 실패를 처리하기 위해서 생성되는 메시지 수로 나타낸다. 무효화 시간은 프로세서가 캐쉬에 있는 복제를 변경하려고 할 때 다른 모든 복제들을 무효화시키는데 필요로 하

는 시간을 의미한다.

4.1 메모리 요구량

P개 프로세서, M개 메모리 블록, 각 프로세서 당 C개 캐쉬 엔트리의 경우에 BIND 기법과 다른 기법들과 비교하였다. 〈표 1〉은 각 기법들에서 디렉토리를 위해서 요구되는 비트 수를 나타낸다. 〈표 1〉에서 보면, BIND 기법과 연결 디렉토리 기법이 최소의 메모리를 요구한다. 하지만 완전사상 디렉토리 기법의 경우는 프로세서 수가 많아지면 요구되는 메모리량이 폭증하게 된다. 그리고 제한 디렉토리 기법에서는 각 디렉토리 엔트리가 가지는 포인터 수에 영향을 받게 된다. STEM 기법에서는 주기억장치에서 필요로 하는 양은 줄일 수 있지만 각 캐쉬에는 각 엔트리 당 3개의 포인터와 트리의 높이 균형을 위한 5비트가 추가로 필요하게 된다.

〈표 1〉 메모리 요구량
〈Table 1〉 Storage overhead

	디렉토리를 위한 메모리량
완전사상 디렉토리 기법	MP
제한 디렉토리 기법	$Mi\log_2 P$
연결 디렉토리 기법	$2M\log_2 P+2CP\log_2 P$
STEM 기법	$M\log_2 P+CP(3\log_2 P+5)$
BIND 기법	$2M\log_2 P+2CP\log_2 P$

4.2 네트웍 통신량

프로세서가 복제를 수정하게 되면, 디렉토리는 같은 복제를 가진 모든 캐쉬에게 무효화 메시지를 보내야 한다. 무효화시키기 위해 생성되는 메시지 수는 상호연결망 상의 통신량으로 시스템 성능에 직접적인 영향을 주게 된다. 〈표 2〉는 복제가 수정될 때 모든 복제를 무효화하기 위해서 생성되는 메시지 수를 나타낸다. 완전사상 디렉토리 기법에서는 N개의 무효화 메시지와 N개의 무효_완료_응답 메시지가 필요하다. 하지만 이 기법에서는 메시지들이 특정 채널에 집중되므로 심각한 네트웍 혼잡 현상을 유발시킬 수 있다. 연결 디렉토리 기법에서는 복제를 가진 N개 캐쉬에 보낼 N개의 무효화 메시지와 연결 리스트의 마지막에 있는 캐쉬가 보낼 1개의 무효_완료_응답 메시지가 필요하다. STEM 기법에서는 무효화 메시지 N개가 필요하며, 단말 노드 수 만큼의 무효_완료_응답 메시지가 필요하다.

〈표 2〉 무효화를 위한 메시지 수
〈Table 2〉 The number of messages for invalidation

	무효화를 위한 메시지 수
완전사상 디렉토리 기법	$2N$
연결 디렉토리 기법	$N+1$
STEM 기법	$N+\lfloor N/2 \rfloor$
BIND 기법	$N+\lfloor N/2 \rfloor$

BIND 기법에서도 STEM 기법과 동일하다. 하지만 BIND 기법에서의 단말 노드의 수는 평형 이진트리를 구성하면서도 최소의 단말 노드 수를 가지므로 최소한의 무효_완료_응답 메시지를 필요로 한다. 제한 디렉토리 기법에서는 완전사상 디렉토리 기법과 동일한 개수의 무효화 메시지가 필요하지만, 정해진 포인터 수보다 많은 프로세서가 자료를 공유하고 있으면 모든 프로세서에게 무효화 메시지를 보내고 무효_완료_응답 메시지를 받아야 한다.

4.3 무효화 시간

〈표 3〉은 각 기법에서 걸리는 무효화 시간을 나타낸다. 완전사상 디렉토리 기법에서는 디렉토리가 모든 복제에 대한 정보를 모두 가지고 있기 때문에 멀티캐스트 기법이 지원되는 경우 동시에 모든 복제를 무효화 시킬 수 있다.

〈표 3〉 무효화의 복잡도
〈Table 3〉 The complexity of the invalidation

	시간 복잡도
완전사상 디렉토리 기법	$O(1)$
연결 디렉토리 기법	$O(N)$
STEM 기법	$O(\log_2 N)$
BIND 기법	$O(\log_2 N)$

연결 디렉토리 기법에서는 무효화 시간이 매우 길다. 복제들을 순차적으로 하나씩 무효화시켜야 하기 때문에 무효화 시간 복잡도가 $O(N)$이 된다. 즉, 무효화 시간이 블록의 복제 수에 비례하게 된다. STEM

기법에서는 루트 노드가 각 자식 노드에 멀티캐스팅으로 무효화 메시지를 보내고, 메시지를 받은 노드가 다시 자식 노드들에게 무효화 메시지를 보냄으로써 무효화시켜 간다. 따라서 트리의 깊이가 거의 $\log_2 N$이기 때문에 무효화 시간이 $O(\log_2 N)$가 된다. BIND 기법도 STEM 기법과 동일한 트리 구조를 가지므로 처리 과정이 동일하여 같은 결과가 된다.

4.4 확장성

디렉토리를 구현하기 위해 요구되는 메모리량은 다중처리기의 확장성에 큰 영향을 미친다. 완전사상 디렉토리 기법에서처럼 복제를 가진 모든 캐쉬를 추적하기 위한 비트 백터의 사용은 시스템의 확장을 어렵게 만든다. 이것은 시스템의 메모리 양이 프로세서 수에 선형적으로 비례하기 때문이다. 하지만 BIND 기법, 연결 디렉토리 기법과 STEM 기법은 디렉토리에 있는 포인터 크기가 프로세서 수에 대수적으로 증가하고 각 캐쉬 엔트리나 메모리 블록 당 포인터 개수가 프로세서 수에 무관하기 때문에 많은 프로세서로의 확장성이 뛰어나다.

상호연결망 상의 통신량 측면에서 보면 BIND 기법, 연결 디렉토리 기법, STEM 기법들에서 디렉토리는 프로세서들에 분산되어 있으므로 무효화를 위한 통신량이 분산된다. 그리고 프로세서가 복제의 수정을 요구하는 경우 메모리 모듈은 오직 하나의 무효화 메시지를 루트 캐쉬로 보내고, 나머지에 대해서는 무효화 메시지를 받은 캐쉬가 자식노드인 캐쉬들에게 무효화 메시지를 보내기 때문에 무효화로 인한 통신량은 일반적인 자료 통신량의 범주에서 취급할 수 있다. 하지만 완전사상 디렉토리 기법에서는 디렉토리가 한 곳에 집중되어 있기 때문에 무효화 과정에서 특정 채널에 메시지가 집중되고 특히 프로세서 수가 많아지고 자료의 공유도가 높을수록 메시지 집중 현상이 심화되기 때문에 프로세서가 계속 증가되는데 문제가 있다. 이러한 현상은 제한 디렉토리 기법에서도 발생한다.

4.5 프로토콜 복잡도

완전사상 디렉토리 기법은 대부분을 하드웨어적으로 처리하기 때문에 프로토콜은 다른 기법에 비교해서 간단하다. 그리고 제한 디렉토리 기법은 완전사상 디렉토리 기법과 대부분 유사하지만 정해진 포인터보다 많은 프로세서가 자료를 공유하는 경우의 처리 방법이 추가적으로 필요하게 된다. 연결 디렉토리 기법에서 사용하는 프로토콜은 완전사상에 비교해서는 복잡하지만 트리 구조를 가지는 기법들보다는 간단하다. STEM 기법에서는 BIND와 유사한 평형 이진트리 구조를 가지고 있지만 삽입이나 대체 시 AVL 이진트리를 유지시키기 위한 프로토콜이 매우 복잡하고 더 긴 시간을 요구하고 있다. BIND 기법에서는 삽입 시 이미 구축된 수 평형 이진트리에 추가될 위치를 연결된 총 노드의 수에 의해서 자동적으로 결정되어지기 때문에 프로토콜 자체는 간단하다. 하지만 연결 디렉토리처럼 리스트의 머리 부분에 직접 연결하는 것이 아니라 단말 노드에 접속되기 때문에 추가적인 삽입 시간이 요구된다. 대체 시에도 최종적으로 연결된 노드의 위치를 SC를 이용하여 간단하게 찾아서 대체될 노드와 바꿈으로써 수 평형 이진트리를 유지시킬 수 있다. 하지만 STEM 기법은 대체 시 AVL 이진트리 유지에 필요한, 최종 연결 노드를 찾는 작업이 복잡하고 추가적인 시간을 요구하고 있다.

5. 결 론

공유메모리 다중처리기에서는 메모리 블록의 복제들이 프로세서에 있는 캐쉬에 저장하게 되므로 이들 복제들 간의 일관성을 유지해 주는 기법이 필요하다. 본 논문에서는 이러한 문제를 해결하기 위한 BIND 기법을 제안하였다. 이 기법은 동기화 방법으로는 쓰기 무효화 방법을 사용하고, 디렉토리 유지 방법으로는 분산 디렉토리 방법을 사용하여 확장성 있는 공유메모리의 다중처리기에 적합하다. 또한 BIND 기법의 디렉토리는 수 평형 이진트리 구조로 디렉토리의 구축이나 변경이 용이하고 디렉토리를 위한 메모리 요구량, 쓰기 요청에 따른 복제의 무효화 시간과 네트웍 통신량, 시스템의 확장성 및 프로토콜의 복잡도 측면에서 기존의 디렉토리 구조보다 우수한 능력을 가지고 있다.

향후는 BIND 기법을 적용한 캐쉬 일관성 유지 프로토콜을 개발하고 이것을 실제 시스템에 구현해 보는 연구와 이 기법을 최근 많이 연구되고 있는 분산 공유메모리 컴퓨터에 적용하는 연구가 필요하다.

참 고 문 헌

[1] Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes cr Cache Coherence," in Proc. of 15th Annual Symposium on Computer Architecture, pp. 280-289, 1988.

[2] Archibald, and J. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Trans. on Computer Systems, Vol. 4, No. 4, pp. 273-298, 1986.

[3] D. Gustavson, "The Scalable Coherence Interface and Related Standards Projects," IEEE Micro, Vol. 12, No. 1, pp. 10-22, 1992.

[4] V. James, A. Laundrie, S. Gjessing, and G. Sohi, "Distributed-Directory Scheme: Scalable Coherent Interface(SCI)," IEEE Computer, Vol. 26, No. 6, pp. 74-77, 1990.

[5] R. Johnson, Extending the Scalable Coherenct Interface for Large-Sacle Shared-Memory Multiprocessors., Ph.D Thesis, Univ. of Wisconsin-Madison, 1993.

[6] Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," Technical Report No. CSL-TR-89-404, Stanford University, 1989.

[7] D. W. Seo and J. W. Cho, "Directory-based cache Coherence Scheme using Number-Balanced Binary Tree," Microprocessing and Microprogramming, Vol. 37, pp. 37-40, 1993.

[8] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," IEEE Computer, Vol. 26, No. 6, pp. 12-24, 1990.

[9] C. Tang, "Cache Design in the Tightly Coupled Multiprocessor System," AFIPS Conference Proc., National Computer Conference, pp. 749-753, 1976.

[10] S. Thakkar, M. Dubois, A. Laundrie, and G. Sohi, "Scalable Shared Memory Multiprocessor Architectures," IEEE Computer, pp. 71-73, 1990.

[11] M. Thapar, and B. Delagi, "Distributed-Directory Scheme: Stanford Distributed-Directory Protocol," IEEE Computer, Vol. 26, No. 6, pp. 78-82, 1990.

서 대 화

1981년 경북대학교 전자공학과 졸업(학사)

1983년 한국과학기술원 전산학과(석사)

1993년 한국과학기술원 전산학과(박사)

1981년~1995년 한국전자통신연구소 시스템 S/W연구실 실장

1995년~현재 경북대학교 전자·전기공학부 조교수

관심분야: 병렬/분산 컴퓨터 구조, 병렬운영체계, 멀티미디어 서버