CC-NUMA 다중 프로세서의 캐쉬 일관성 유지를 위한 세그먼트 디렉토리

최 종혁†○ 박 규호‡
† LG 종합기술원 시스템 IC 센터
‡ 한국과학기술원 전기및전자공학과 컴퓨터공학연구실
choix@lgcit.com, kpark@core.kaist.ac.kr

Segment Directory for Cache Coherence of CC-NUMA Multiprocessors

Jong Hyuk Choi†○ Kyu Ho Park‡
† LG Corporate Institute of Technology, System IC Center
‡ Computer Engineering Research Lab., Dept. of Electrical Eng., KAIST
choix@lgcit.com, kpark@core.kaist.ac.kr

요 약

세그먼트 디렉토리는 CC-NUMA (Cache Coherent Non-Uniform Memory Access) 시스템의 전체 벡터와 포인터의 장점만을 가지는 새로운 형태의 디렉토리 요소이다. 포인터가 하나의 프로세서 위치만을 가리키는데 비해, 세그먼트 디렉토리는 복수 개의 프로세서들을 한 번에 가리킬 수 있으면서, 포인터처럼 작은 단위로 사용 가능하다. 본 논문에서는 세그먼트 디렉토리를 제한 디렉토리 방법들에 적용하여 디렉토리 넘침의 횟수를 줄인다. 기존의 방법들이 디렉토리 넘침 후의 효율적인 캐쉬 일관성 유지 기법을 제시했던 것에 비해, 세그먼트 디렉토리는 디렉토리 넘침 자체를 제거하는 최초의 시도이다. 디렉토리 넘침의 제거로 CC-NUMA 시스템 대역폭 요구량이 줄어들고, 프로그램 수행이 가속되며, 디렉토리 제어기 점유가 대폭 감소된다. Tango-Lite 를 사용한 실행 구동 시뮬레이션을 통하여 세그먼트 디렉토리가 약 80% 까지의 디렉토리 넘침을 제거한 것을 확인하였고, 이에 따르는 시스템 성능 향상을 분석하였다.

1. 서 론

최근 CC-NUMA 구조의 다중 프로세서 시스템들이 널리 보급되고 있다. 수십 개 이상의 프로세서를 연결할 수 있는 확장성과 더불어 단일 시스템 이미지 주소 공간을 제공하여 경제적이고 편리한 운용 환경을 제공하는 것이 CC-NUMA의 장점이다. 대표적인 시스템들로는 Stanford의 DASH와 FLASH, MIT의 Alewife, SGI의 Origin, Sequent의 NUMA-Q, HAL의 S1 등이 있고, 국내에서는 KAIST의 Hanbit3가 있다. 대부분의 CC-NUMA는 확장성을 위하여 디렉토리를 사용한 캐쉬 일관성 유지 기법을 채택하고 있다. SMP에서 사용되는 버스 기반의 스누핑 방법이 어드레스 트랜잭션을 방송하여 캐쉬 복사본들 간의 일관성을 유지하는 데 비해, 디렉토리 방법에서는 분산되어 있는 메모리 블록 옆에 디렉토리 블록을 준비하여 캐쉬 복사본들의 위치를 명시적으로 기록한다. 디렉토리에 기록된 복사본들에게만 일관성 유지 메시지들을 전달함으로써 상호 연결망의 포화를 피하고 CC-NUMA의 확장성을 향상시킨다.

디렉토리 프로토콜은 전체 벡터와 포인터 디렉토리를 디렉토리의 구성 요소로 사용한다. 전체 벡터 디렉토리 방법 (FM) [1] 은 메모리 블록마다 노드 수 (N) 비트로 구성된 벡터를 준비하여 시스템의 캐쉬 상태를 정확히 기록한다. 최적의 성능을 얻을 수 있으나, 디렉토리 메모리가 $O(N^2)$로 증가하여 중형 이상 CC-NUMA에서는 사용이 곤란하다.

제한 디렉토리와 연결 리스트 디렉토리 방법은 구성 요소로서 포인터를 사용한다 [3]. 제한 디렉토리 방법은 메모리 블록마다 수 개의 포인터를 준비하여 공유도가 작은 대부분의 경우 위치를 기록한다. 포인터가 모자라는 경우인 디렉토리 넘침을 해결하기 위하여 LimB, LimNB, CV, LimitLESS 방법이 제안되었다. 연결 리스트 디렉토리 방법은 캐쉬 블록들에도 포인터를 준비하여, 메모리 블록을 헤드로 하는 캐쉬 복사본의 연결 리스트를 구성, 관리한다.

본 논문에서는 전체 벡터와 포인터의 장점을 모두 가지면서 훨씬 더 효율적인 디렉토리 구성 방법인 세그먼트 디렉토리 [2] 를 소개하고 그 응용 사례들을 제시한다.

세그먼트 디렉토리는 전체 벡터의 일부분인 세그먼트 벡터와 전체 벡터 내에서 세그먼트 벡터의 위치를 가리키는 세그먼트 포인터로 구성된다. 본 논문에서는 제한 디렉토리 방법의 포인터를 대신하여 세그먼트 디렉토리를 사용하는 것을 제안한다. 이는 디렉토리 넘침을 줄인 최초의 시도로서 약 80%의 디렉토리 넘침을 제거하였다. SPLASH-2 벤치마크를 Tango-Lite 시뮬레이션 환경에서 수행하여 디렉토리 넘침 제거에 따르는 성능 향상 - 통신량 감소, 프로그램 수행 가속, 제어기 점유 감소 - 을 측정하고 분석하였다.

2. 세그먼트 디렉토리

2.1 전체 벡터 대 포인터

포인터는 전체 벡터에 비해 훨씬 크기가 작아서 적은 크기의 디렉토리를 효율적으로 구성할 수 있으나, 포인터의 정

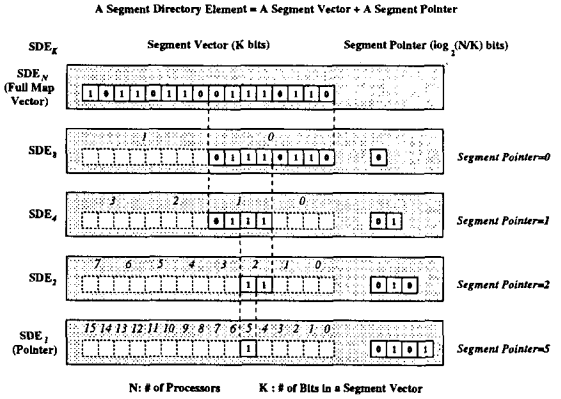


그림 1: 세그먼트 디렉토리의 구성.

적 저장 효율은 전체 벡터 디렉토리의 효율에 비해 매우 작다. 디렉토리 정적 저장 효율을 "하나의 캐쉬 복사본을 가리키는 데 필요한 비트 수"로 정의한다면, 전체 벡터 디렉토리의 정적 저장 효율은 항상 1이다. 반면에, 포인터는 $1+log_2N$ 비트를 가지므로 정적 저장 효율이 $1+log_2N$이다. 하지만, 실제로는, 대부분의 메모리 블록 공유도가 그리 크지 않기 때문에, 일반적으로 전체 벡터 디렉토리의 실제 동적 저장 효율은 포인터보다 낮다.

세그먼트 디렉토리는 전체 벡터와 포인터의 장점을 모두 갖는 새로운 디렉토리 구성 요소이다. 이는 포인터처럼 독립적인 작은 단위로서 사용되는 동시에, 전체 벡터에 필적하는 정적 디렉토리 비트 저장 효율을 가지므로, 결과적으로 전체 벡터나 포인터보다 높은 실제 저장 효율을 갖는다.

2.2 세그먼트 디렉토리의 구조

16 노드 시스템을 위한 세그먼트 디렉토리 구성이 그림 1에 도시되어 있다. 세그먼트 디렉토리 요소는 세그먼트 벡터와 세그먼트 포인터라는 두 개의 부분으로 구성되어 있다. 세그먼트 벡터는 전체 벡터의 일부분인 연속 K 비트 서브 벡터로서, 해당 노드들의 메모리 블록 캐쉬 상태를 나타낸다. 세그먼트 포인터는 전체 벡터 내에서 세그먼트 벡터의 위치를 가리킨다. 본 논문에서는 세그먼트 벡터가 서로 겹치지 않는 (N/K) 위치를 가지는 경우를 다루므로 세그먼트 포인터는 $\log_2(N/K)$ 비트를 가진다. 크기 K인 세그먼트 디렉토리 요소 (SDE_K) 가 차지하는 디렉토리 메모리 크기 $NoB\,(SDE_K)$ 는 $K+\log_2(\frac{N}{K})$ 로 표현된다.

16 노드 CC-NUMA 시스템에서 그림 1에 도시되어 있듯이 SDE_4는 4 비트 세그먼트 벡터와 2 비트 $(=\log_2(\frac{16}{4}))$ 세그먼트 포인터로 구성되어, 6 비트를 가지고 최대 4 개의 노드를 가리킬 수 있다. 세그먼트 포인터 값이 1 이므로 이는 세그먼트 1 번의 4 개의 노드들, 즉, 노드 4, 5, 6, 7 의 캐쉬 상태를 나타낸다. 세그먼트 벡터는 그 중에서 노드 4, 5, 6 이 메모리 블록의 복사본을 지역 캐쉬에 가지고 있다는 것을 알려준다.

세그먼트 디렉토리의 구성 프레임웍은 기존의 디렉토리 요소인 전체 벡터와 포인터를 포함한다: SDE_N은 전체 벡터이고, SDE_1은 포인터이다. 세그먼트 디렉토리는 기존의 전체 벡터와 포인터를 모두 포함하는 통일된 디렉토리 요소 구성 방법론을 제시하면서 이들 사이에 존재하는 새로운 구성의 효과적인 디렉토리 요소들을 찾아내었다.

SDE_2는 포인터와 같은 양의 메모리를 사용하면서 인접한 두 개의 노드를 동시에 가리킬 수 있으므로 포인터에 비해 두 배의 정적 저장 효율을 갖는다. 2 보다 큰 K를 갖는

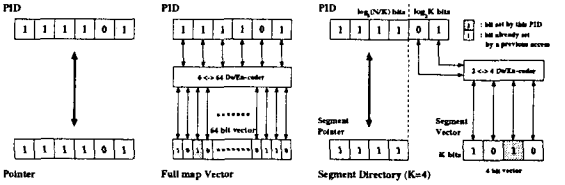


그림 2: 노드 ID와 세그먼트 디렉토리의 변환 ($N=64$).

SDE_K의 저장 효율은 이보다 훨씬 더 크다. SDE_4는 포인터 보다 한 비트를 더 사용하면서 4 개의 노드를 가리킬 수 있고, SDE_8은 $5+\log_2 N$ 비트를 가지고 8 개의 노드를 가리킬 수 있다. K가 증가함에 따라 급속히 정적 저장 효율이 높아져서 전체 벡터의 정적 저장 효율에 빠르게 수렴한다. 결과적으로 작은 크기의 세그먼트 디렉토리를 사용할 경우 포인터처럼 원하는 갯수만큼 사용할 수 있는 동시에 전체 벡터에 근접하는 높은 저장 효율을 갖는다.

2.3 세그먼트 디렉토리의 구현과 응용

세그먼트 디렉토리 요소와 노드 ID 간의 변환은 전체 벡터와 포인터의 변환 로직들을 조합하여 구현된다 (그림 2). 세그먼트 포인터는 NID의 상위 $\log_2(\frac{N}{K})$ 비트로 직접 복사되고, 세그먼트 벡터에서 NID의 하위 $\log_2 K$ 비트로의 변환은 우선 순위 인코더를 통하여, NID 하위 비트에서 세그먼트 벡터로의 변환은 디코더를 통하여 수행된다.

세그먼트 벡터가 전체 벡터보다 훨씬 작은 것이 일반적이므로, 세그먼트 벡터의 변환 로직을 훨씬 작고 빠르게 구현할 수 있다. 게다가, 전체 벡터의 크기는 노드 수에 비례하여 증가하지만, 세그먼트 벡터의 크기는 고정될 수 있다.

세그먼트 디렉토리는 대부분의 포인터 기반 디렉토리 방법에서 포인터 대신 사용된다. 본 논문은 세그먼트 디렉토리를 방송 제한 디렉토리 (LimB) / 거친 벡터 디렉토리 (CV) 와 LimitLESS 디렉토리 (LimLESS) 에 적용하였다.

제한 디렉토리 방법은 디렉토리 넘침 발생 전에는 공유 노드의 ID를 포인터에 저장한다. LimB는 디렉토리 넘침 시 방송 비트를 설정하고, 이후 쓰기 요구가 도착하면 모든 노드들에게 무효화 메시지를 방송한다. CV는 디렉토리 넘침 시 포인터들을 거친 벡터 디렉토리로 재배열한다. 거친 벡터의 각 비트는 복수 개의 노드로 구성된 영역을 가리킨다. 쓰기 요구 도착 시, 1로 설정된 영역들 내의 모든 노드들에게 무효화 메시지를 부분 방송한다. 영역 크기가 N인 CV는 LimB이며 이들은 동일한 디렉토리 넘침 특성을 갖는다. LimLESS에서는 디렉토리 넘침 시 인터럽트를 발생시켜 디렉토리를 노드의 주 메모리로 복사하여 재사용한다.

제한 디렉토리 방법에 세그먼트 디렉토리를 적용할 경우, 1) 쓰기 요구 시 포인터가 오직 하나의 무효화 메시지를 생성시키는 반면, SDE_K는 변환 로직을 통하여 최대 K 개를 생성할 수 있고, 2) 읽기 요구 노드 ID가 기록될 세그먼트 디렉토리가 이미 존재하는지 확인하기 위하여 세그먼트 포인터 필드에 대한 병렬 비교기가 필요하다. 이들은 모두 메모리 접근 지연을 증가시키지 않으면서 구현이 가능하다.

3. 성능 분석

세그먼트 디렉토리의 효과를 분석하기 위하여 Stanford SPLASH-2 벤치마크를 정확한 결과를 신속하게 얻을 수 있는 Tango Lite 실행 구동 시뮬레이션 환경에서 수행하여 결과를 얻었다. 메쉬 웜홀 라우팅 네트워크로 연결된 64 노드 CC-NUMA 시스템을 대상으로 하였다. MSI 무효화 프로토콜을 사용하였으며 일관성 관련 메모리 참조에 집중하기 위하여 무한 크기 캐쉬를 가정했다. 단일 프로세서 노드

표 1: 디렉토리 넘침의 제거 (%는 $(4,1(,r))$ 방법에 정규화).

디렉토리 방법	Radix	Barnes
$LimB_{4,1} \cdot CV_{4,1,r}$	15,650 (100%)	33,178 (100%)
$LimB_{4,2} \cdot CV_{4,2,r}$	14,880 (95%)	27,768 (84%)
$LimB_{4,4} \cdot CV_{4,4,r}$	13,595 (87%)	20,583 (62%)
$LimB_{5,1} \cdot CV_{5,1,r}$	15,050 (96%)	15,393 (98%)
$LimLESS_{4,1}$	127,031 (100%)	79,716 (100%)
$LimLESS_{4,2}$	77,534 (61%)	69,931 (88%)
$LimLESS_{4,4}$	40,355 (32%)	53,709 (67%)
$LimLESS_{5,1}$	100,082 (79%)	57,361 (72%)

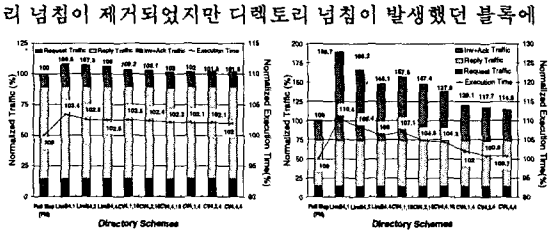
이고, 캐쉬와 메모리 블록 크기는 32 바이트이다. 캐쉬, 버스, 디렉토리, 네트워크 인터페이스 등을 정확하게 모델링하였으며, 네트워크는 Agarwal의 모델을 사용하였다.

제한 디렉토리 방법들을 각각 $LimB_{i,k}$, $LimLESS_{i,k}$, 그리고 $CV_{i,k,r}$ 로 표기한다. i는 메모리 블록 당 디렉토리 요소의 갯수, k는 세그먼트 벡터의 크기, 그리고 r은 거친 벡터 디렉토리 방법에서 영역의 크기이다. $(4,1(,r))$, $(4,2(,r))$, $(4,4(,r))$, $(5,1(,r))$ 인 구성을 비교하였다. $(4,1(,r))$은 네 개의 포인터를 사용한다. 성능 향상을 위해 $(4,2(,r))$ 구성은 네개의 SDE_2를 사용하고 $(4,4(,r))$는 네개의 SDE_4를 사용한다. 기존의 접근 방식에서는 하나의 포인터를 추가해서 $(5,1(,r))$로 구성하여 성능 향상을 꾀했을 것이다.

표 1은 디렉토리 넘침의 감소를 잘 보여 준다. $(4,2(,r))$ 구성은 $(4,1(,r))$의 디렉토리 넘침의 39%까지를 제거하였다. 두 구성 모두 28 비트의 디렉토리 블록을 사용하고, 모든 경우에 디렉토리 넘침이 감소하였으므로, 디렉토리의 실제 동적 저장 효율이 향상되었다고 볼 수 있다. $(4,4(,r))$ 구성은 메모리 블록 당 4 비트를 추가로 사용하면서 68%까지의 디렉토리 넘침을 제거하였다. 반면에, $(5,1(,r))$은 7 비트를 추가로 사용하면서 최대 28%의 디렉토리 넘침을 제거했을 뿐이다. 모든 경우에서 $(4,4(,r))$은 $(5,1(,r))$보다 더 많은 디렉토리 넘침을 제거하였다. 따라서, 세그먼트 디렉토리를 사용하는 것이 단순히 포인터를 추가하는 것 보다 디렉토리 넘침을 제거하는 데 훨씬 효과적이다.

비방송 제한 디렉토리 (LimNB) 의 디렉토리 넘침이 가장 빈번하다. 디렉토리 넘침 시 캐쉬 블록을 무효화 시킴으로써 높아진 캐쉬의 미스율이 다시 디렉토리 넘침을 증가시키는 정궤환이 존재하기 때문이다. 표 1로부터 LimLESS의 디렉토리 넘침이 LimB/CV에서보다 빈번하다는 것을 알 수 있다. LimB/CV에서는 일단 디렉토리 넘침이 발생하면 무효화 시점까지 더 이상 디렉토리 넘침이 발생하지 않는 반면, LimLESS에서는 각 넘침 후 연속된 i번 이상의 읽기 요구가 도착하면 디렉토리 넘침이 다시 발생한다.

그림 3은 LimB/CV 상에서 수행된 두 워크로드의 통신량과 수행시간을 보여준다. 요구와 응답 통신량이 항상 동일함을 알 수 있다. LimB/CV가 미스율을 높이지 않기 때문에 무효화 통신만이 증가한다. Radix의 경우 대부분의 쓰기가 하나 이하의 무효화 메시지를 유발한다. 많은 디렉토리 넘침이 제거되었지만 디렉토리 넘침이 발생했던 블록에



(a) Radix (b) Barnes

그림 3: LimB와 CV의 통신량과 수행시간.

표 2: LimLESS의 수행시간 (%는 전체 벡터 방법에 정규화).

디렉토리 방법	Radix	Barnes
$LimLESS_{4,1}$	140.1	103.4
$LimLESS_{4,2}$	129.8	103.3
$LimLESS_{4,4}$	124.5	102.1
$LimLESS_{5,1}$	133.0	102.2

대한 쓰기의 발생 빈도가 적기 때문에 성능 개선이 그다지 크지 않다. Barnes에서는 많은 무효화 메시지를 발생시키는 쓰기가 빈번히 발생한다. 이러한 워크로드의 경우, 제한 디렉토리 방법의 성능 저하가 크다.

FM에 비해 $LimB_{4,1}$의 통신량은 89% 증가하였고 프로그램은 10% 정도 느리게 수행되었다. 세그먼트 디렉토리는 이러한 통신량을 40% 이상 감소시켰고, 프로그램 수행을 5% 정도 가속시켰다. 64 노드 시스템에서 CV의 영역 크기(r)를 4로 하였다. $CV_{4,1,4}$의 성능이 이미 매우 좋기 때문에 세그먼트 디렉토리를 사용한 성능 향상이 크지는 않다. 통신량은 6% 정도 감소하고 수행시간은 2% 정도 줄어든다. 그런데, 512 노드 시스템에서 거친 벡터의 크기를 줄이려면 $r = 16$으로 하여 거친 벡터가 32 비트를 갖게 한다. 64 노드 시스템에서의 $CV_{4,k,16}$의 성능을 그림 3에 표시하였다. 23%의 통신량이 감소되었고 수행 시간은 3%까지 감소하였다.

LimLESS의 통신량은 FM과 동일하기 때문에 수행시간만을 표 2에 나타내었다. LimLESS에서 수행시간은 디렉토리 넘침 시 프로세서 인터럽트와 인터럽트 처리 지연에 직접 연결되므로, LimLESS의 성능은 무효화 쓰기보다는 디렉토리 넘침에 더 관련이 있다. Radix의 수행 시간 증가는 LimB/CV의 경우에서 처럼 작지 않다. 세그먼트 디렉토리는 Radix의 경우 수행 시간을 10% 이상 줄일 수 있었다. $LimLESS_{4,4}$가 $LimLESS_{5,1}$보다 항상 좋은 성능을 가진다.

디렉토리 넘침은 디렉토리 제어기 점유도를 떨어뜨린다. 디렉토리 넘침을 처리하는 데 필요한 디렉토리 제어기의 동작을 줄이기 때문이다. 특히 LimLESS 같은 프로토콜과 SMP 노드 기반의 CC-NUMA 성능을 크게 향상시킨다.

4. 결 론

본 논문에서 제안된 세그먼트 디렉토리는 전체 벡터와 포인터를 포함하는 통합 디렉토리로서, 포인터처럼 작은 단위로 사용되면서 전체 벡터처럼 높은 정적 저장 효율을 갖는다. CC-NUMA 성능 결정 경로에 놓여있는 메모리 접근 지연을 전혀 증가시키지 않고, 고속 구현이 가능하다.

세그먼트 디렉토리를 제한 디렉토리 방법에 적용하여 많은 디렉토리 넘침을 제거하였고, 이에 따라 CC-NUMA의 통신 요구량과 프로그램 수행시간이 감소되는 것을 확인하였다. 또한, 디렉토리 제어기의 점유도 또한 감소된다.

세그먼트 디렉토리의 효과가 매우 큰 것은 프로그램 내에 존재하는 특정 참조 지역성의 덕택이다. 컴파일러가 이러한 참조 지역성을 증가시키기 위한 최적화를 수행한다면 세그먼트 디렉토리의 성능은 더욱 향상될 수 있을 것이다.

참고 문헌

[1] L. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. *IEEE Trans. Comput.*, C-27(12):1112–1118, Dec. 1978.

[2] J. H. Choi and K. H. Park. Segment directory enhancing the limited directory cache coherence schemes. *Proc. 13th Int'l Parallel Processing Symp.*, pp. 258–267, Apr. 1999.

[3] M. Tomašević and V. Milutinović. Hardware approaches to cache coherence in shared-memory multiprocessors: Part 1/2. *IEEE Micro*, 14(5/6):52–59/61–66, Oct./Dec. 1994.