분산공유 메모리 시스템을 위한 동적 제한 디렉터리 기법

이 동 광† · 권 혁 성†† · 최 성 민††† · 안 병 철††††

요 약

분산 공유 메모리(distributed shared memory) 시스템에서 캐쉬는 메모리 접근 지연과 통신 부하 줄임으로 성능을 향상시킬 수 있으나 캐쉬일관성 문제를 해결하여야 한다. 본 논문은 DSM 시스템에서 캐쉬일관성 문제를 해결하고 성능을 향상시킬 수 있는 새 디렉터리 프로토콜을 제안한다. 캐쉬 일관성을 유지하기 일정거리 이내에 있는 처리기는 전체 디렉터리 기법처럼 비트 벡터를 사용하여 통신 오버헤드를 줄일 수 있다. 그리고 일정거리 이상에 있는 처리기는 포인터를 디렉터리 풀에 저장한다. 이 비트 벡터와 디렉터리 풀의 사용은 불필요한 캐쉬 무효화를 방지하므로 시스템의 성능을 향상시킬 수 있다. 제안한 기법은 제한 디렉터리 기법보다 통행량을 66%까지 줄일 수 있으며 동적할당 디렉터리 기법보다 디렉터리 접근 회수도 27%까지 각각 줄일 수 있다.

Dynamic Limited Directory Scheme for Distributed Shared Memory Systems

Dong-Kwang Lee† · Hyek-Seong Kweon†† · Seong-Min Choi††† · Byoung-Chul Ahn††††

ABSTRACT

The caches in distributed shared memory systems enhance the performance by reducing memory access latency and communication overhead, but they must solve the cache coherence problem. This paper proposes a new directory protocol to solve the cache coherence problem and to improve the system performance in distributed shared memory systems. To maintain the cache coherence of shared data, processors within a limited distance reduce the communication overhead by using a bit-vector like the full directory scheme. Processors over a limited distance store pointers in a directory pool. Since the bit-vector and the directory pool remove the unnecessary cache invalidations, the proposed scheme reduces the communication traffic and improves the system performance. The dynamic limited directory scheme reduces the communication traffic up to 66 percents compared with the limited directory scheme and the number of directory access up to 27 percents compared with the dynamic pointer allocation scheme.

1. 서 론

분산 공유 메모리 시스템에서 각각의 처리기는 캐쉬와 메모리를 가지며 메모리는 가상 메모리 시스템을 구성하여 논리적으로 공통의 주소 공간을 지닌다. 각 처리기는 단일한 주소 공간의 일부분을 제공함으로써 프로그램의 이식성을 높일 수 있고 쉽게 프로그램을 작성할 수 있다[1]. 캐쉬는 메모리 접근 지연시간과 통신 병목 현상을 줄여 전반적인 시스템의 성능을 향상

† 종신회원 : 경북전문대학 전산정보처리과 교수
†† 준 회 원 : LG전자연구소 연구원
††† 준 회 원 : 영남대학교 대학원 컴퓨터공학과
†††† 종신회원 : 영남대학교 컴퓨터공학과 교수

논문접수 : 1998년 8월 27일, 심사완료 : 1999년 2월 5일

기인다. 그러나 캐쉬의 메모리 복은 캐쉬와 캐쉬결해 간 일관성(consistency) 문제를 해결하여야 한다[2,3,4]. 이 문제를 해결하고 시스템의 성능을 높이기 위한 다양한 캐쉬 일관성유지 기법이 제안되었다.

수백 개 이상을 연결하는 분산 공유 메모리 시스템에서는 캐쉬 일관성 유지를 위해 일대일 통신에 적합한 디렉터리 방법을 사용한다[3,4]. 지금까지 소개된 디렉터리 방법의 캐쉬일관성 유지 기법은 작은 디렉터리 메모리를 사용할 경우 캐쉬 무효화와 갱신을 자주 하여야 하며 이것은 많은 메시지의 발생과 통신 부하를 증가시킨다. 메시지 발생을 줄이기 위해 디렉터리 메모리를 많이 사용할 경우 프로세서 수 증가에 대한 메모리 공간의 복잡도가 $O(N^2)$ 혹은 $O(N\log N)$이 되어 낭비가 된다. 그러나 인접한 프로세서의 지역성을 고려할 경우 메모리 사용 복잡도를 $O(N)$로 줄여 효과적인 메모리 관리를 할 수 있다. 메시지 발생 수를 줄이는 것은 통신 오버헤드를 줄일 수 있으므로 성능을 향상시킬 수 있다. 본 논문은 인접 프로세서의 지역성을 이용하여 효과적인 메모리의 사용과 통신 오버헤드를 줄일 수 있는 캐쉬 일관성 유지 기법을 제안한다.

2장에서는 캐쉬 일관성 문제 해결을 위한 기존 연구의 장단점을 비교한다. 3장에서는 제안하는 동적제한 디렉터리 방식의 기본 개념과 동작 방법에 대해 설명한다. 4장에서는 모의실험으로 성능을 분석하며 끝으로 5장에서 결론을 맺는다.

2. 캐쉬 일관성 유지 기법과 문제점

대규모 다중 처리기 시스템은 공유 버스 유효 전송량이 제한되므로 데이타의 고유 상태를 메모리에 저장하는 디렉터리 기법을 주로 사용한다. 이 기법은 전체 디렉터리 기법, 제한 디렉터리 기법과 체인 디렉터리 기법 등이 대표적이며 이들 기법의 특징과 장단점은 다음과 같다.

2.1 캐쉬 일관성 유지 기법

전체 디렉터리 기법은 모든 캐쉬 블럭에 대한 정보를 메모리에 저장·관리하는 기법이다. 디렉터리는 각 메모리 블럭을 처리기의 캐쉬 수와 동일한 수의 존재 비트와 블럭의 상태를 나타내는 비트로 구성된다[4]. 이 기법은 상태 정보를 중앙에서 일괄적으로 관리하며 일관성을 유지하기 위한 메시지를 해당 캐쉬에만 전송[illegible] 불필요한 부하를 줄일 수 있는 장점이 있다[2,4]. 그러나 디렉터리를 구성하기 위해 각 메모리 블럭마다 전체 시스템의 캐쉬 수만큼 메모리가 요구되므로 처리기의 증가에 따른 메모리 낭비가 많으며 확장성이 없다.

제한 디렉터리 기법은 단위 시간대에 단지 작은 수의 처리기만이 동일한 블럭을 공유한다는 '데이타 지역성' 개념을 적용하여 공유 메모리 블럭 내의 포인터 수를 줄일 수 있다[2,4,8]. 이 기법은 공유 데이타에 접근하는 처리기의 수가 제한된 포인터 수를 넘지 않으면 전체 디렉터리 기법처럼 사용된다. 그러나 제한된 포인터 수를 넘으면 포인터 하나를 선택하여 무효화한 다음 메모리 접근을 요구한 처리기의 포인터를 저장한다. 이 기법은 포인터 수를 제한함으로서 디렉터리의 사용량을 줄일 수 있으나 제한된 포인터 수 이상의 처리기가 데이타를 공유 하고자 할 때 포인터를 교체하므로 문제점이 있다. 이 기법은 포인트의 교체가 빈번할 경우 시스템의 성능이 저하되는 단점이 있다[2,3].

체인 디렉터리 기법은 공유 데이타 블럭의 수를 제한하지 않아 제한 디렉터리의 단점을 보안한 방식이다. 디렉터리 포인터들이 연결 리스트를 사용하므로 공유블럭을 공유하는 프로세서의 수에 제한을 받지 않는다. 이 기법은 캐쉬 블럭의 교체시 제한 디렉터리 방식보다 복잡한 단점은 있으나 확장성이 좋다[7,8].

Simoni에 의하여 제안된 동적 포인터 할당 기법은 FLASH 시스템에서 사용되었다. 동적 포인터 할당 기법은 공유 데이타에 접근하는 처리기의 포인터를 사용하여 디렉터리 풀에 저장하므로 포인터 교체에 의한 처리기의 유휴 시간을 줄일 수 있다. 포인터의 삽입, 삭제와 무효화는 간단한 함수로 구현이 가능하다[6]. 디렉터리 풀의 크기를 초과하면 삽입된 캐쉬 블럭 중 하나를 무효화 시켜야 하므로 디렉터리 풀의 크기를 캐쉬 블럭 수의 4배나 8배 정도로 한다.

그리고 Chaiken에 의하여 제한된 LimitLESS 프로토콜은 Alewife 시스템에서 사용되었으며 소프트웨어로 구현되었다[7,14]. 공유 데이타에 접근하는 처리기 수가 제한된 포인터의 수를 넘거나 공유 데이타에 대한 캐쉬 미스가 발생하면 소프트웨어 트랩을 발생하고 문맥 전환을 이용하여 다른 작업을 수행한다. 무효화 작업이나 캐쉬 미스에 대한 처리가 종료되면 다시 문맥 전환을 하여 원래의 작업을 수행한다. 그러나 성능

향상을 위해 멀티쓰레드 기법을 사용하므로 빠른 문맥 전환을 위한 하드웨어가 필요하다는 단점이 있다.

2.2 기존 기법의 문제점

디렉터리 기법을 사용해 캐쉬 일관성을 유지할 경우에는 디렉터리를 위한 메모리 사용량과 일관성 유지 메시지에 따른 통신망의 통행량을 고려해야 한다[8]. 전체 디렉터리 기법은 처리기의 수가 증가함에 따라 메모리의 사용량이 $O(N^2)$로 증가하며 디코딩 오버헤드를 증가시킨다. 제한 디렉터리 기법은 메모리의 사용량이 $O(N\log N)$로 증가하여 대규모 시스템으로 확장은 가능하나 포인터의 부족으로 읽기 무효화 메시지를 증가시킨다. 그리고 일대일 통신을 하는 다단계 상호연결 통신망 구조에서는 시스템 전반에 걸친 메시지 전달이 어렵고 무효화가 이루어질 때 전달의 종료를 인식하기 어려운 단점이 있다.

동적 포인터 할당 기법은 $M \cdot x + P(\log N+x)$의 메모리 사용 복잡도를 가진다. 여기서 M은 고유메모리 블럭수, x는 처리기의 포인트 수, P는 제한된 포인터를 초과하는 최대포인트 수이며 N은 캐쉬 블럭 수이다. $P(\log N+x)$는 프로그램 실행 시에 디렉터리 풀에서 동적으로 사용되므로 실제 디렉터리의 사용량은 $M \cdot x$이므로 $O(N)$으로 메모리 사용 복잡도를 줄일 수 있다. 그러나 모든 처리기의 포인터를 연결 리스트로 된 디렉터리 풀에 저장하므로 포인터의 제거 혹은 무효화 시에 연결 리스트를 순차적으로 탐색해야 하므로 접근 시간이 길어지는 단점이 있다[7,14].

지금까지 소개된 디렉터리 기법은 메모리 사용의 복잡도가 $O(N^2)$에서 $O(N\log N)$이거나 $O(N)$인 경우는 탐색시간이 긴 단점이 있다. 그러나 인접한 프로세서의 지역성을 고려할 경우 메모리 사용 복잡도를 $O(N)$ 줄일 수 있으며 통신 오버헤드를 줄여 성능을 향상시킬 수 있다.

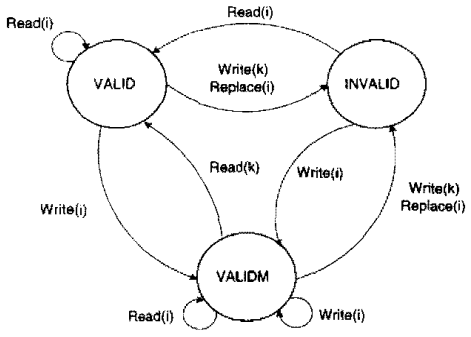
3. 동적 제한 디렉터리 기법

인접한 프로세서의 지역성과 디렉터리 풀의 장점을 이용하여 새 캐쉬 일관성 유지 기법을 제안한다. 이 기법의 프로토콜과 디렉터리 사용의 복잡도를 분석한다.

3.1 프로토콜

동적 제한 디렉터리 기법에서 캐쉬는 상태 비트와 캐쉬 데이타로 구성되어 있다. 캐쉬의 상태는 (그림 1)과 같이 INVALID, VALID와 VALIDM 세 가지가 있다. INVALID는 캐쉬 블럭 내에 원하는 데이타가 존재하지 않는 상태이며, VALID는 캐쉬 블럭 내에 원하는 자료가 존재하고 메모리 블럭의 내용과 같은 상태이다. 그리고 VALIDM은 메모리 블럭이 캐쉬로 복사된 후 그 내용을 갱신하였으며 최근의 갱신 내용이 메모리로 재 복사되지 않은 상태를 각각 의미한다. 이 기법은 블럭이 캐쉬 내에 머물러 있는 동안은 연속해서 갱신되고, 메모리로 대체되어 나가는 경우에는 전체 블럭의 내용이 메모리로 재 복사된다.

(그림 1)에서 Read는 읽기 작업을, Write는 쓰기 작업을, Replace는 캐쉬 블럭의 대체 작업을 나타낸다. i와 k는 각각 다른 처리기를 나타낸다. 예를 들면 Write(k)는 처리기 k의 공유 데이타에 대한 쓰기 작업을 의미한다.



(그림 1) 캐쉬 상태도
(Fig. 1) Cache state diagram

일반적으로 병렬 처리 컴퓨터의 컴파일러는 공유 데이타를 많이 사용하는 처리기에 데이타를 배치시킬 수 있다. 데이타를 요구하는 처리기 간의 통신 거리가 일정 거리 이하인 처리기들은 저장 비트를 이용하고, 그 이외의 처리기는 메모리 포인터를 이용하여 디렉터리 풀에 저장한다. 공유 데이타의 접근 특성상 데이타를 공유하고자 하는 처리기의 대부분은 임의의 거리 이내에 존재할 것이다. 임의의 거리는 시스템 설계자에 의하여 결정되며 이를 제한거리라고 정의한다.

공유데이타의 특성을 고려하면 동적 제한 디렉터리 기법의 디렉터리 구조는 (그림 2)와 같이 상태 비트, 저장 비트와 메모리 포인터로 구성할 수 있다. 저장 비트의 수는 공유 데이타를 가진 처리기로부터 제한거리 이내에 있는 처리기의 수와 같다.

S	P_1	...	P_i	X

S : 상태 비트
P : 처리기 포인터
i : 저장 비트의 수 = 제한거리이내의 처리기의 수

$$i = \sum_{k=1}^{d} 2^{k+1} + 1, \quad (1 < i < N,\ d = \text{제한거리})$$

X : 메모리 포인터

(그림 2) 디렉터리 구조
(Fig. 2) Directory structure

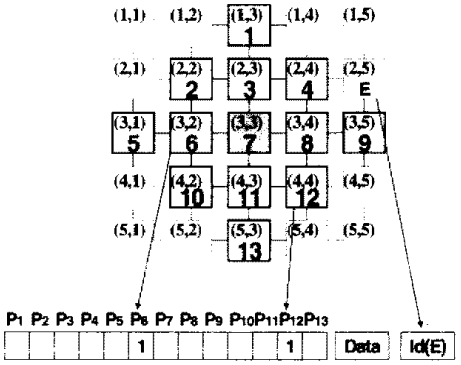
각 처리기의 포인터는 (x, y)와 같은 행렬 표기법을 사용한다. 예를 들어 $n \times n$ 메쉬 구조에서 처리기의 포인터는 (0, 0)에서 $(n-1, n-1)$까지 사용할 수 있다. 여기서 (0, 0)는 행렬의 왼쪽 상단에 $(n-1, n-1)$은 오른쪽 하단에 위치한다. 이 경우 임의의 점 (A', B')에 있는 처리기가 (A, B)에 있는 처리기의 공유 데이타에 접근할 때 처리기간의 통신 거리 d는 식 (1)과 같다.

$$d = |A' - A| + |B' - B| \tag{1}$$

식 (1)의 처리기 간의 통신 거리가 제한 거리 이하인 경우는 저장 비트를 사용한다. 저장 비트에 놓이는 순서는 식 (2)의 δ의 순으로 세트한다. $A' > A$이면 δ는 양수 값이 되며 $A' < A$이면 δ는 음수의 값을 갖는다. 그리고 $A'=A$, $B'=B$ 이면 공유 데이타의 위치가 되며 $\left(\sum_{k=1}^{d} 2^k + 1\right)$로 정의한다.

$$\delta = \pm \left| \sum_{k=1}^{|A'-A|} 2(d-k+1) \right| + \left(\sum_{k=1}^{d} 2^k + 1 \right) + (B' - B) \tag{2}$$

공유 데이타의 위치를 기준으로 하여 $\pm \left| \sum_{k=1}^{|A'-A|} 2(d-k+1) \right|$는 행 좌표의 변화를 의미하며 $(B' - B)$는 열 좌표의 변화를 의미한다. 따라서 각각 다른 (A', B')에 대하여 $\pm \left| \sum_{k=1}^{|A'-A|} 2(d-k+1) \right| + (B' - B)$의 값이 다르므로 저장 비트 δ는 유일한 값을 가진다. 일정한 거리 내에 있는 처리기는 저장 비트 δ가 가리키는 위치에 저장되나 그외의 처리기는 동적 포인터 할당 기법처럼 디렉터리 풀에 포인터를 저장한다. (그림 3)은 메쉬 통신망에서 제한거리가 2이고 포인터 (3, 3)인 처리기의 공유 데이타를 포인터 (3, 2), (4, 4)와 (2, 5)인 처리기들이 접근한 것을 동적 제한 디렉터리 기법에서 처리기를 포인터한 예이다. (그림 3)에서 굵은 글자의 숫자는 식(2)에 의해 계산된 δ값이다. 포인터 (3, 2)과 (4, 4)인 처리기는 식 (1)의 계산 결과에서 처리기간의 통신 거리가 2 이하이므로 식 (2)의 계산 결과로 바트 6과 비트 12에 각각 세트된다. 그러나 포인터 (2, 5)인 처리기는 통신거리가 3이므로 디렉터리 풀에 저장된다.



(그림 3) 동적 제한 디렉터리 기법의 예 (d=2)
(Fig. 3) Example of the dynamic limited directory scheme (d=2)

3.2 프로토콜 분석

동적 제한 디렉터리 기법은 대규모 다중 처리기 시스템에서 사용하는 병렬 처리 프로그램의 메모리 참조 특성을 고려하였다[10]. 이 기법은 요구하는 처리기 간의 통신 거리가 일정거리 이하인 경우 처리기 포인터를 저장 비트에 세트시키고 캐쉬 블럭에 쓰기 작업을 수행한다. 일정거리 이상인 처리기는 동적 포인터 할당 기법처럼 디렉터리 풀에 처리기 포인터를 저장한다. 제한된 영역에 대하여 비트 단위로 처리기를 포인터 하므로 디렉터리의 효율성을 높일 수 있다. 제한 영역밖의 소수의 처리기만이 디렉터리 풀에 저장되므로 연결 리스트에 의한 디렉터리 풀의 접근 시간과 사용량을 줄일 수 있다.

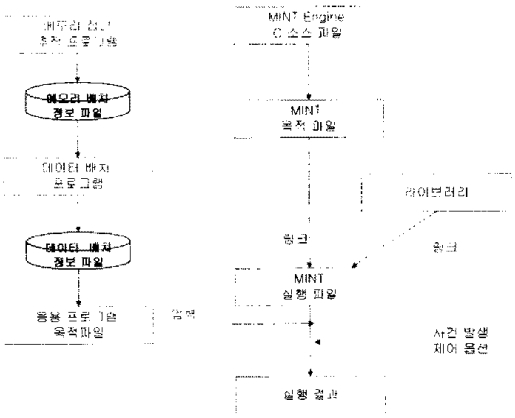
동적 제한 디렉터리 기법은 $M(i+x) + P(\log N+x)$의 메모리 사용 복잡도를 가진다. 여기서 M은 고유메모리 블럭수, i는 저장비트 수, x는 처리기의 포인트 수, P는 제한된 포인터를 초과하는 최대포인트 수이며 N은 캐쉬 블럭 수이다. $P(\log N+x)$는 프로그램 실행 시에 디렉터리 풀에서 동적으로 사용되므로 실제 디렉터리의 사용량은 $M(i+x)$이므로 $O(N)$으로 메모리 사용 복잡도를 가진다. 동적 제한 디렉터리 기법은 동적 포인터 할당 기법보다 디렉터리가 $M \cdot i$만큼 크지만 디렉터리의 저장 비트에서 처리기를 i개 포인터할 수 있다. 그러므로 디렉터리 풀의 사용량인 $P(\log N+x)$가 동

적 오인나 캐시 미스보다 작다. 실제적으로 디렉터리 풀의 사용량의 감소는 프로세서의 캐시 무효화 시에 연결 리스트를 순차적으로 탐색하는 횟수를 줄일 수 있으므로 평균적인 디렉터리 풀의 접근 시간을 줄일 수 있다.

4. 모의실험

4.1 모의실험 환경

본 논문에서는 동적 제한 디렉터리 기법의 효율성을 분석하기 위해 Rochester 대학에서 개발된 다중 처리기 시뮬레이터인 MINT(Mips INTerpreter)를 사용하였다[9]. (그림 4)는 입력된 병렬 응용프로그램의 실행 코드를 해석하여 각 처리기들의 실행을 모의실험한다. 여기서 발생하는 메모리 참조의 횟수를 공유 데이타 배치 정책에 따라 예상되는 공유 데이타의 참조 수를 구할 수 있다. 모의실험에 사용한 인수는 <표 1>과 같다[8,9].



(그림 4) 모의실험 방법
(Fig. 4) Simulation method

〈표 1〉 모의실험 인자
〈Table 1〉 Simulation parameters

하드웨어 비용 관련 인자	비 용
메모리 참조	20 cycles
한 노드당 평균 통신망 지연	20 cycles
통신망의 통행량	32 Bytes
연속 무효화 메시지 지연	1 cycle
캐쉬 사상 방법	Direct Mapped
캐쉬 블럭 크기	32 Bytes
캐쉬 크기	128 KB

4.2 평가 프로그램

모의실험에 사용된 응용 프로그램은 여러 연구에서 많이 사용된 프로그램인 Gauss, Water, FFT이며 공유도가 각각 상, 중, 하이다. Gauss는 448×448행렬에 대해 피벗팅(pivoting)없이 가우스소거법을 실행하는 프로그램이다. Water 프로그램은 N-바디 분자역학을 모의실험하는 프로그램으로 액체 상태에서 물분자계에서 위치에너지와 힘을 계산하는 프로그램이다. 그리고 FFT는 65536개의 복소수 원소를 1차원 FFT 알고리즘에 적용한 프로그램이다. 8×8 메쉬 구조를 가진 64개의 처리기에서 동적 제한 디렉터리 기법과 제한 디렉터리 기법의 성능을 각각 측정하였다[9,10].

〈표 2〉 응용 프로그램의 특성
〈Table 2〉 Characteristics of application programs

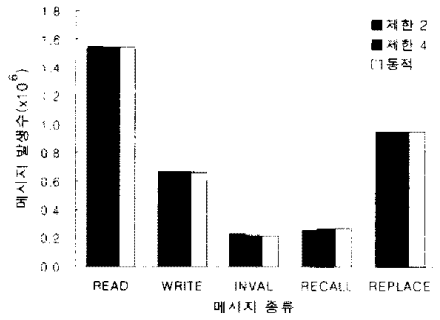
프로그램 이름	총메모리 참조 회수 (단위:10^6)	공유데이타 크기 (단위:MB)	메모리 참조율(%)			
			Private Read	Private Write	Shared Read	Shared Write
FFT	208.03	2.003	55.1	38.2	4.2	2.5
Water	237.98	0.183	59.9	22.9	15.6	1.6
Gauss	265.92	2.536	0.01	0.01	66.88	33.1

<표 2>에서 FFT와 Gauss 프로그램은 공유 데이타의 크기가 크나 Water는 공유 데이타의 크기가 작다. 총메모리 참조회수는 3개의 응용 프로그램이 비슷하다. 제한 디렉터리 기법과 동적 제한 디렉터리 기법의 메시지 수와 통신 부하량을 서로 비교하여 성능을 평가하였다. 제한 디렉터리 기법에서는 포인터의 크기가 2와 4인 경우를 사용하였으며 동적 제한 디렉터리 기법에는 제한거리를 2로 하였다. 여기서 얻은 결과를 가지고 디렉터리 사용량과 접근 회수를 동적 포인터 할당 기법과 비교하였다. 동적 포인터 할당 기법은 동적 제한 디렉터리와 같이 전체 디렉터리의 효과를 가지므로 캐쉬 일관성 유지를 위한 메시지의 수와 통신망의 통행량은 같다고 할 수 있다. 그러므로 메시지의 수와 통신 부하량 비교는 실험에서 제외시켰다. 제한 디렉터리 기법은 디렉터리 풀을 사용하지 않으므로 디렉터리 풀의 참조 회수 및 사용량 비교는 실험에서 제외시켰다.

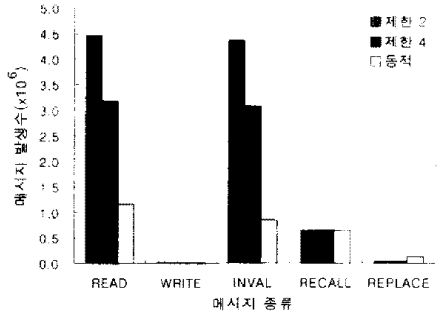
4.3 메시지의 수와 통신 부하량 비교

캐쉬 일관성 유지 기법에서 응용 프로그램의 공유

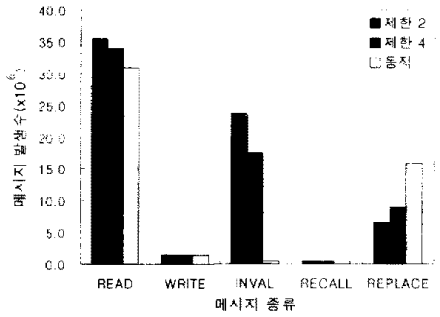
또는 포인터의 수와 밀접한 관계가 있다. FFT와 같이 공유도가 낮은 프로그램은 공유 데이타 블럭에 접근하는 처리기의 수가 적으면 포인터 부족으로 인한 포인터 교체가 적다. 그러므로 포인터의 수를 증가시켜도 (그림 5.a)처럼 메시지의 수는 감소하지 않는다. 그러나 Water나 Gauss 프로그램과 같이 공유도가 높은 프로그램은 공유 데이타 블럭에 접근하는 처리기의 수가 많으므로 포인터를 많이 교체해야 한다. 포인터의 부족은 많은 읽기 무효화 메시지를 발생시켜서 통신망의 통행량을 증가시킨다. 모의실험에서 포인터의 수를 증가시키면 (그림 5.b)와 (그림 5.c)처럼 메시지의 수가 급격하 감소한다. (그림 5.c)에서 동적 제한 디렉터리



a. FFT



b. Water



c. Gauss

(그림 5) 평가 프로그램의 메시지 수 비교
(Fig. 5) Comparison of message numbers of application programs

가 제한 디렉터리 보다 무효화 메시지는 적으나 대체 메시지가 많은 것을 관찰할 수 있다. 그것은 동적 제한 디렉터리 기법에서 캐쉬의 내용을 무효화시키지 아니하고 교체하기 때문에 일어나는 메시지이다.

<표 3>은 각 응용 프로그램을 실행시켜 메쉬 통신망의 통행량을 비교한 것이다. 제한 디렉터리 기법에서 응용 프로그램의 공유도가 낮을 경우 포인터의 수를 증가해도 통신망의 부하는 줄어들지 않으나 공유도가 높은 프로그램에서는 통신망 부하가 급속히 줄어든다. Water 프로그램에서 통신망 부하가 Gauss 프로그램 보다 적은 이유는 쓰기 메모리 접근 회수가 적고 접근하는 공유 데이타 수도 작기 때문이다. 응용 프로그램에서 쓰기 메모리 접근은 통신망의 통행량에 상당한 영향을 미친다. 즉, 2개 이상의 처리기들이 하나의 공유 변수에 대해 연속적으로 읽기와 쓰기 메모리 접근을 수행함으로써 다른 처리기의 캐쉬 내용을 무효화시킬 수 있다. 그 결과 처리기간의 빈번한 캐쉬블럭 전송은 통신망의 통행량을 급격히 증가시킨다.

〈표 3〉 통신망 통행량 비교
〈Table 3〉 Comparison of communication traffic

기법 \ 프로그램	FFT	Water	Gauss
제한 디렉터리 기법 (포인터수=2)	19.5MB	35.0MB	275.9MB
제한 디렉터리 기법 (포인터수=4)	19.5MB	26.1MB	262.2MB
동적 제한 디렉터리 기법	19.5MB	11.8MB	234.5MB

제한거리 이상에 있는 처리기들의 성능은 제안한 기법이 제한 디렉터리 기법 보다 낮을 경우도 있다. 그러나 캐쉬 블럭을 공유하는 처리기가 많다면 제한 디렉터리 기법에서는 포인터 부족으로 인한 무효화 메시지가 발생하므로 보다 제안한 기법보다 성능이 더 못할 수 있다. 모의실험 결과와 같이 동적 제한 디렉터리 기법은 처리기를 모두 포인터할 수 있어 전체 디렉터리 기법의 효과를 나타내므로 기존의 제한 디렉터리 기법보다 통신망의 통행량을 줄일 수 있다.

4.4 디렉터리 풀의 참조 회수의 비교

<표 4>는 각 응용 프로그램 실행시 필요한 디렉터리 풀의 수와 접근 회수를 나타내었다. 각 결과를 동적 포인터 할당 기법의 디렉터리 풀의 사용량과 디렉

더리 풀의 접근 회수를 기준으로 도표화한 것이다. 동적 포인터 할당 기법을 보는 처리기의 포인터를 디렉터리 풀에 저장하므로 인접 리스트가 길어진다. 제거·무효화시 많은 리스트를 탐색해야 하므로 디렉터리 풀의 접근 시간이 길어진다. 동적 제한 디렉터리 기법은 제한 디렉터리 이하의 처리기는 저장 비트에 저장하므로 적은 수의 처리기의 포인터만을 디렉터리 풀에 저장한다. 따라서 동적 포인터 할당 기법보다 상대적으로 디렉터리 풀의 접근 시간이 줄어든다. Water 프로그램에서 동적 제한 디렉터리 기법이 동적 포인터 할당 기법보다 디렉터리 풀의 사용량과 접근 회수에서 각각 23%와 27%까지 감소하였음을 관찰할 수 있다.

〈표 4〉 디렉터리 풀 사용량과 접근 회수의 비교
〈Table 4〉 Comparison of directory pool usage and access number

프로그램 / 기법	FFT		Water		Gauss	
	사용량 (포인터)	접근 회수	사용량 (포인터)	접근 회수	사용량 (포인터)	접근 회수
동적 포인터 할당 기법	39.9×10^{3}	2.9×10^{6}	28.6×10^{3}	2.8×10^{6}	171.0×10^{3}	163×10^{6}
동적 제한 디렉터리 기법	51.2×10^{3}	2.5×10^{6}	22.2×10^{3}	2.1×10^{6}	151.2×10^{3}	127×10^{6}

5. 결 론

분산 공유 메모리에서 캐쉬일관성 유지를 위해 여러 가지 기법들이 제시되었으나 처리기의 수가 증가함에 따라 메모리의 낭비와 통행량의 증가를 가져왔다. 특히 2차원 메쉬의 분산 공유 메모리 구조에서 통행량의 증가는 시스템 성능을 저하시키는 직접적인 요인이 된다. 본 논문에서 하드웨어와 소프트웨어를 혼용하여 성능을 향상시키면서 캐쉬 일관성 유지를 할 수 있는 동적 제한 디렉터리 기법을 제안하였다. 일정거리 이내에 있는 처리기는 전체 디렉터리 기법처럼 비트 단위로 포인터하여 메모리의 효율성을 높이고 일정거리 이상에 있는 포인터는 동적 포인터 할당 기법처럼 연결 리스트를 사용하여 디렉터리 풀의 접근시간을 줄일 수 있다. 디렉터리 풀의 사용은 포인터 교체시 야기되는 처리기의 유휴 시간을 감소시켜 전반적인 성능을 향상시킬 수 있다.

동적 제한 디렉터리 기법은 포인터를 디렉터리 풀에 저장하므로 포인터의 부족으로 생기는 읽기 무효화 메시지를 줄일 수 있음을 모의실험을 통해 확인하였다. 이 메시지의 감소는 통신망의 통행량을 줄일 수 있으므로 성능 향상을 할 수 있다. 공유도가 낮은 응용 프로그램에서는 다른 기법과 차이가 없으나 공유도가 높을 경우 메시지 수가 현저히 감소함을 관찰하였다. 그 이유는 비트 단위로 처리기를 포인터하므로 적은 양의 메모리로 많은 수의 처리기를 포인터 할 수 있으며 디렉터리 풀의 사용량과 접근 시간을 줄일 수 있기 때문이다. 동적 제한 디렉터리 기법은 제한 디렉터리 기법보다 통신망의 통행량을 최대 66%까지 줄일 수 있었다. 그리고 동적 제한 디렉터리 기법이 동적 포인터 할당 기법보다 디렉터리 풀의 사용량에서 23% 감소와 접근 회수에서 27% 감소를 관찰할 수 있다.

동적 제한 디렉터리 기법은 인접한 프로세서의 지역성을 이용하였으므로 데이타 배치 방법에 따라 성능이 달라질 수 있는 단점이 있다. 그리고 이 단점을 극복하기 위해 컴파일러 개발이 필요하다. 향후 공유도와 데이타의 배치 방법에 따른 통행량을 분석하고자 한다.

참 고 문 헌

[1] Jelica Protic, Milo Tomasevic, adn Veljko Milutinovic, "Distributed Shared Memory : Concepts and Systems," *IEEE Parallel & Distributed Technology*, pp.63-79, summer, 1996.

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

[3] James Archibald and Jean-Loup Baer, "Cache Coherence Protocols : Evaluation Using A Multiprocessor Simulation model," *ACM Trans, on Computer Systems*, pp.273-298, Nov, 1986.

[4] Anant Agarwal, Richard Simon, John Hennessy, and Mark Horowitz, "An Evaluation of Directory Schemes for Cache Coherence, *"Proc., 15th Int'l Symp. Computer Architecture*, pp.280-289, 1988.

[5] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache System," *IEEE Trans. on Computer*, Vol.C-27, No.12, pp.1112-1118, Dec. 1978.

[6] Richard Simoni, "Cache Coherence Directories

for Scalable Multiprocessors," *PhD. Thesis*, Stanford Univ., 1992.

[7] David L. Chaiken, "Mechanisms and Interfaces for Software-Extended Coherent Shared Memory," *PhD. Thesis*, MIT, 1994.

[8] Ross Evan Johnson, "Extending The Scalable Coherent Interface for Large Scale Shared Memory Multiprocessors," *PhD. Thesis*, Wisconsin-Madison Univ. 1993.

[9] J. E. Veenstra and R. J. Fowler, "MINT : A Front End for Efficient Simulation of Shared Memory Multiprocessors," *Proc., 2nd Int'l Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 201-207, Jan. 1994.

[10] J. P. Singh, W. Weber and A. Gupta, "SPLASH : Stanford Parallel Applications for Shared Memory," *ACM SIGARCH Computer Architecture News, 20(1)*, pp.5-14, March 1992.

[11] S. Eggers and R. Katz, "The Effect of Sharing on the Cache and Bus Performance of Parallel Programs," *Proc. Third ASPLOS*, pp.257-270, Apr. 1989.

[12] 박규호, 정봉순, 최종혁, 홍춘성, "병렬처리 컴퓨터의 구조 및 응용", 정보과학회지, Vol.14, No.6, 1978년~1984년 국1996.

[13] Timothy B. Brecht, "Multiprogrammed Parallel Application Scheduling in NUMA Multiprocessors," *PhD. Dissertation*-CSRI Technical Report CSRI-303.

[14] David L. Chaiken, "LimitLESS Directories : A Scalable Cache Coherence Scheme," *ASPLOS IV proceedings*, pp.224-234, Apr. 1991.

이 동 광

e-mail : ldkg@cc.kp-c.ac.kr

1986년 영남대학교 전자공학과 졸업(학사)

1989년 영남대학교 대학원 전자공학과(공학석사)

1999년 영남대학교 컴퓨터공학과(공학박사)

1992년~현재 경북전문대학 전산정보처리과 조교수

1996년~현재 경북 전문 대학 전자계산소장

관심분야 : 멀티미디어 시스템, 병렬 처리, 멀티쓰레드 시스템.

권 혁 성

e-mail : hskwon@mail2.lge.co.kr

1995년 영남대학교 전산공학과 졸업(학사)

1997년 영남대학교 대학원 전산공학과(공학석사)

1997년~현세 LG 전자 디스플레이 제품 연구소 연구원

관심분야 : 컴퓨터 구조, 병렬 처리, 실시간 운영체제

최 성 민

e-mail : smchoi@cse.yeungnam.ac.kr

1995년 영남대학교 전산공학과 졸업(학사)

1995년~현재 영남대학교 대학원 컴퓨터공학과

관심분야 : 컴퓨터 구조, 멀티미디어, 네트워크, 컴퓨터 게임

안 병 철

e-mail : ahn@cse.yeungnam.ac.kr

1976년 영남대학교 전자공학과 졸업(학사)

1986년 오레곤 주립대 전기 및 컴퓨터공학과 졸업(석사)

1989년 오레곤 주립대 전기 및 컴퓨터공학과 졸업(박사)

1978년~1984년 국방과학연구소 연구원

1989년~1992년 삼성전자 컴퓨터부문 수석연구원

1992년~현재 영남대학교 공과대학 컴퓨터공학과 부교수

관심분야 : 컴퓨터 구조, 그래픽스, 멀티미디어 및 실시간 운영체제