데이타 배치 방식에 따른 캐쉬 일관성 유지 기법의 성능 평가

(Performance Evaluation of Cache Coherence Scheme for Data Allocation Methods)

이 동 광 † 권 혁 성 †† 안 병 철 †††

(Dongkwang Lee) (Hyekseong Kweon) (Byoungchul Ahn)

요 약 분산 공유 메모리(Distributed Shared Memory) 시스템에서 데이타 참조의 지역성은 시스템 성능에 중요한 영향을 미친다. 데이타 참조의 지역성을 고려하여 적절하게 데이타를 배치할 경우 전체적인 시스템 성능 향상을 가질 수 있다. 본 논문에서는 데이타 배치 방식을 효과적으로 적용할 수 있는 동적 제한 디렉터리 기법에서 성능을 평가한다. 데이타 배치 방식 정보는 동적 제한 디렉터리 기법에서 존재 비트를 효과적으로 이용할 수 있다. 그리고 적절한 존재 비트의 사용은 메모리 오버헤드를 줄이고 디렉터리 풀을 효율적으로 사용하므로 성능을 향상시킬 수 있다. 성능 평가를 위해 서로 다른 공유 특성을 가진 3개의 응용 프로그램으로 모의 실험하였다. 모의 실험 결과 최적 배치 방식은 3.6 배의 성능을 향상시킬 수 있다.

Abstract The locality of data references at the distributed shared memory systems affects the performance significantly. Data allocation methods by considering the locality of data references can improve the performance of DSM systems. This paper evaluates the performance for the dynamic limited directory scheme which data allocation methods can apply very effectively. The information of the data allocation is used by the dynamic limited directory scheme to set the presence bit effectively. And the proper use of the presence bit improves the performance by reducing memory overhead and using directory pool efficiently. Simulations are conducted using three application programs which have various data sharing. The results show that the optimal data allocation method improves the performance up to 3.6 times in the proposed scheme.

1. 서 론

분산 공유 메모리 시스템은 각각의 처리기가 자신의 캐쉬와 메모리를 가지나 논리적으로 공통의 주소 공간을 지닌다. 각 처리기는 단일한 주소 공간을 가지므로 프로그램의 이식성을 높이고 프로그램의 작성이 쉬운 장점이 있다. 그러나 이 구조는 분산 공유 메모리 접근시 통신을 위한 유휴시간이 증가한다[1,10].

캐쉬의 사용은 유휴시간, 각 처리기간의 통신 병목 현상과 메모리 접근 지연 시간을 줄이므로 시스템의 성능을 향상시키나 캐쉬와 메모리, 캐쉬와 캐쉬간의 일관성(consistency) 문제가 발생한다[1,2]. 캐쉬 일관성 유지를 위해 수 백개 이상의 프로세서를 연결한 대규모 다중 처리기 시스템에서는 디렉터리 방법을 사용한다[2,3]. 존재 비트를 이용하여 디렉터리를 구성하는 방법은 상태 정보가 메모리와 캐쉬에 분산되어 있어 병목 현상을 줄일 수 있다. 그리고 존재 비트를 이용하여 공유 데이타를 사용하는 처리기를 쉽게 관리할 수 있으나 존재 비트 크기가 증가함에 따라 디렉터리의 크기도 증가하므로 시스템의 확장성이 떨어진다[2,3,4,5].

대부분의 분산 공유 메모리 시스템에서는 확장을 위

† 정 회 원 : 경북전문대학 컴퓨터정보과 교수
ldkg@mail.kbc.ac.kr
†† 비 회 원 : LG전자 디스플레이제품연구소 연구원
hskwon@lge.co.kr
††† 종신회원 : 영남대학교 컴퓨터공학과 교수
ahn@yeungnam.ac.kr
논문접수 : 1998년 9월 24일
심사완료 : 2000년 3월 29일

해 공유 데이타에 접근하는 처리기의 색인(index)을 디렉터리 풀(pool)에 저장하거나 전처리기(preprocessor)와 같은 소프트웨어를 이용하여 처리기의 데이타 접근 순서를 미리 결정하여 디렉터리의 효율성을 높인다. 제한 디렉터리 기법에서 디렉터리의 크기를 초과할 경우 발생하는 시스템의 유휴 시간을 소프트웨어 트랩(trap)과 문맥 전환(context switching)을 이용하여 줄이는 방법도 있다. 디렉터리 풀을 사용하는 동적 포인터 할당 기법은 전체 디렉터리 기법과 같은 성능을 가지며 디렉터리를 효율적으로 사용하는 장점이 있다. 그러나 디렉터리 풀의 접근 시간을 줄이기 위해 SRAM을 사용해야 하고 디렉터리 풀은 공유 메모리 크기의 3배 내지 4배 정도의 크기를 가져야 하는 단점이 있다. 그리고 디렉터리 풀에서 처리기의 색인을 링크 리스트로 연결하므로 처리기 색인을 탐색할 경우 처리기의 유휴 시간이 필요하다[3]. 전처리기를 이용하는 방법은 하드웨어 구조에 독립적이나 실행할 때 발생하는 처리기의 공유 데이타 접근 순서나 접근 종류를 모두 파악할 수 없는 단점이 있다[5]. LimitLESS기법은 소프트웨어 트랩을 이용하여 문맥 전환을 하더라도 같은 캐쉬 라인에 대하여 캐쉬 미스가 발생하면 유휴 시간이 필요하다[4].

2장에서는 용어 정의와 공유 데이타 배치 방법을 설명하고 3장에서는 동적 제한 디렉터리 기법에 대하여 간단히 설명한다. 4장에서는 모의실험을 통하여 데이타 배치에 따른 성능 차이를 평가하며 끝으로 5장에서 결론을 기술한다.

2. 분산 공유 메모리 시스템에서 데이타 배치 방식

2.1 용어 정의

분산 공유 메모리 시스템에서 사용되는 처리기의 집합을 P라 하며 공유 메모리 블럭의 집합을 M이라고 정의하자. $Num(P)$는 처리기의 수를, $Num(M)$은 전체 공유 메모리 블럭의 크기를 각각 의미한다. $Index(P)$와 $Index(M)$은 처리기와 공유 메모리 블럭의 색인을 각각 의미한다. 임의의 처리기 i는 식 1과 같이 표기하며 처리기가 접근하고자 하는 임의의 공유 메모리 S는 식 2와 같이 표기한다. 이때 %는 나머지 연산자이다.

$$P_i = (\lfloor i / \sqrt{Num(P)} \rfloor , i \% \sqrt{Num(P)}), \quad (P_i \in P, 0 \le i \le Num(P)) \quad (1)$$
$$= (i.x, i.y)$$

$$S = \{ S \in M \mid 0 \le S \le Num(M) \} \quad (2)$$

2.2 데이타 배치 방식

다중 처리기 시뮬레이터에서는 일반적으로 공유 데이타를 임의로 배치하거나 블럭 단위로 배치하는 방식을 많이 이용한다. 그러나 이러한 데이타 배치 방식은 네트워크 비용을 고려한 모의실험과 실측에서 상당한 성능 차이를 보인다[8,9]. 성능향상은 다음과 같은 4가지 방법이 있다. 첫째, 데이타의 크기를 분할하여 부하 분산의 효과를 이용할 수 있다. 둘째로, 공유 데이타를 동적으로 배치하여 지역성을 증가시킨다. 셋째, 사용자 쓰레드를 커널 외부에서 구현하여 화일의 생성 및 문맥 전환 등으로 야기되는 과부하를 감소시킨다. 마지막으로 효과적인 스케쥴링 기법을 이용하여 시스템의 성능을 향상시킬 수 있다.[9].

본 논문에서는 데이타의 지역성을 고려하여 배치하고 이 데이타 배치 방법을 효과적으로 적용할 수 있는 동적 제한 디렉터리 기법을 이용하여 성능을 평가한다. 데이타 배치 방법은 아래와 같이 3가지 방식으로 배치한다.

2.2.1 최적 배치 방식

최적 배치는 공유 메모리에 접근하는 처리기의 접근 횟수와 공유 메모리까지의 네트워크 거리를 계산하여 데이타를 배치하는 방식이다. 처리기 P_i가 처리기 P_k에 있는 공유 메모리 S에 접근할 경우 네트워크 거리 $D_{i,k}$는 식 3과 같다.

$$D_{i,k} = D_{k,i} = | k.x - i.x | + | k.y - i.y | \quad (3)$$

네트워크 거리는 처리기가 데이타에 접근하기 위해 지나가는 노드의 수이다. 임의의 P_i가 처리기 P_k에 있는 공유 메모리 S에 대한 접근 횟수를 $C_{i,k}$ 라 정의하면 총 네트워크 비용 Z_i는 식 4와 같다.

$$Z_i = \sum_{k=0}^{P} C_{k,i} \cdot D_{k,i} \quad (4)$$

그러므로 공유 메모리 S에 있는 공유 데이타는 식 5를 이용하여 Z_i값이 최소인 처리기 i에 배치한다.

$$i = MIN(Z_i), \qquad 0 \le i < Num(P) \quad (5)$$

2.2.2 최다 배치 방식

최다 배치는 처리기 P_k에 있는 공유 메모리 S에 접근하는 처리기 중에서 접근 횟수가 가장 많은 처리기에 데이타를 배치하는 방식이다. 따라서 공유 메모리 S에 있는 공유 데이타는 식 6을 이용하여 접근 횟수가 가장 많은 처리기 i에 배치한다.

$$i = MAX(C_{i\cdot k}) \quad (6)$$

2.2.3 연속 배치 방식

연속 배치는 배치 방식이 단순하여 다중 처리기 시뮬레이터에서 많이 사용된다. 이 배치 방식은 전체 공유 메모리 블럭을 처리기의 수로 나누어 각 처리기에 그 연속된 블럭을 하나씩 배치하는 방식이다. 전체 공유 메모리의 크기인 $Num(M)$이 2^m이고, 처리기의 수 $Num(P)$이 2^n 일 경우 이 등식을 만족하는 정수 m과 n이 존재한다고 가정하면 공유 메모리 S에 있는 공유 데이타는 식 7을 이용하여 얻을 수 있다.

$$S = 2^i = 2^{(m-n)}, \quad i = m-n \quad (7)$$

식 7에서 각 프로세서에 할당하는 공유메모리의 크기는 2^i가 된다. 예를 들어 공유메모리의 크기 $Num(M)$이 1024, 프로세서의 수가 $Num(P)$가 64이면, $m - n = 4$가 되어 공유메모리 S의 크기는 $2^i = 2^4 = 16$ 블럭이다. 그리고 프로세서가 P_k의 공유 메모리 S가 162번째 블럭일 경우와 1023 블럭일 경우를 고려해 보자. 첫 번째 경우, 공유 메모리 S는 162/16의 몫이 되어 P_{10}에 배치되고 두 번째 경우, 공유 메모리 S는 1023/16의 몫이 되어 P_{63}에 배치된다. 프로세서의 수 $Num(P)$이 64이고 공유메모리 S에 대하여 프로세서 P_0, P_{10}과 P_{11}만이 접근하며 그 접근회수를 각각 5, 4, 3일 경우 각 프로세서의 통신비용 Z_{10}과 Z_{63}는 다음과 같다.

$$Z_{10} = C_{0\cdot 10}\cdot D_{0\cdot 10} + C_{10\cdot 10}\cdot D_{10\cdot 10} + C_{11\cdot 10}\cdot D_{11\cdot 10}$$
$$= 5\cdot 3 + 4\cdot 0 + 3\cdot 1 = 18$$
$$Z_{63} = C_{0\cdot 63}\cdot D_{0\cdot 63} + C_{10\cdot 63}\cdot D_{10\cdot 63} + C_{11\cdot 63}\cdot D_{11\cdot 63}$$
$$= 5\cdot 14 + 4\cdot 11 + 3\cdot 10 = 144$$

최적배치나 최다배치와 비교할 경우 첫 번째 경우의 통신비용은 크지 않으나 두 번째 경우의 통신비용은 144로서 상당히 큰 것을 알 수 있다.

2.3 데이타 배치와 지역성

식 8은 각 처리기가 공유 데이타에 접근할 때 필요한 평균 네트워크 거리, W_i을 구하는 식이다. 표 1은 식 8을 이용하여 각 배치 방식에 따라 처리기가 공유 데이타 접근시 구해진 평균 네트워크 거리 W_i를 나타낸 것이다. 이 결과는 로체스터 대학에서 개발된 다중 처리기 시뮬레이터인 MINT(Mips INTerpreter)에서 SPLASH의 응용 프로그램인 FFT를 1024의 메모리 블럭에 대하여 모의실험한 것이다[6,7,8].

$$W_i = Z_i \,/\, \sum_{k=0}^{Num(P)} C_{k\cdot i} \quad (8)$$

표 1 데이타 접근시 평균 네트워크 거리

	최적 배치	최대 배치	연속 배치
평균 네트워크 거리	1.107112	1.110039	4.774468

표 1에서 최적 배치나 최다 배치는 네트워크의 평균 거리가 2미만이다. 그 이유는 공유 데이타 블럭이 배치된 임의의 처리기 주위에 공유 데이타가 있기 때문이다. 그러므로 이 정보를 존재 비트를 이용하는 동적 제한 디렉터리 기법에 효과적으로 이용할 수 있다.

3. 동적 제한 디렉터리 기법

제한 디렉터리 기법의 단점은 특정 블럭을 공유하는 처리기의 개수가 디렉터리보다 많을 경우 발생하는 디렉터리 교체에 따른 시스템의 성능 저하이다. 즉, 이미 사용되고 있는 임의의 디렉터리를 선택하여 무효화시키고 새로 요청한 캐쉬를 등록하는 과정에서 시간이 소요된다. 처리기는 무효화 요구 시점부터 작업 종료 확인 신호를 받을 때까지 수행을 중지하여야 하며 해당 블럭에 대한 참조도 금지되므로 시스템의 성능이 저하된다. 동적 포인터 할당 기법은 기존의 디렉터리를 무효화하는 대신 처리기의 색인을 디렉터리 풀에 저장하므로 디렉터리 교체에 따른 오버헤드를 줄였다. 그러나 처리기의 색인을 연결 리스트를 이용하여 저장하므로 포인터 제거·무효화 시에 연결 리스트를 탐색해야 하는 단점이 있다[3].

동적 제한 디렉터리 기법은 특정 네트워크 거리 이하인 처리기는 존재 비트를 이용하여 저장하므로 메모리의 사용량을 줄일 수 있고 네트워크 거리를 초과하는 처리기는 디렉터리 풀에 저장하므로 제한 디렉터리 기법의 단점을 보안한다. 그리고 대부분의 처리기 색인은 디렉터리에 저장되고 소수의 처리기 색인만이 디렉터리 풀에 저장되므로 디렉터리 풀의 사용량이나 접근 횟수가 적어 시스템 확장이 용이하며 탐색 지연에 따른 유휴 시간을 줄일 수 있다.

처리기가 공유 데이타에 접근할 경우 먼저 식 3을 이용하여 네트워크 거리를 계산한다. 그 값이 특정 값 이하이면 처리기의 색인은 식 9를 이용하여 디렉터리의 δ 번째의 존재 비트를 세트한다. 식 9에서 공유 데이타의 위치는 $\left(\sum_{k=1}^{d} 2^k + 1\right)$로 정의한다. 공유 데이타의 위치를 기준으로 하여 $(B' - B)$는 행 좌표의 변화를 의미하며 $S\left|\sum_{k=1}^{A'} 2(d-k+1)\right|$는 열 좌표의 변화를 의미한다. 그러므로

각각 다른 (A', B')에 대하여 $S\left|\sum_{k=1}^{|A'-A|} 2(d-k+1)\right|+(B'-B)$의 값이 다르므로 존재 비트 δ는 유일한 값을 가진다.

$$\delta = S\left|\sum_{k=1}^{|A'-A|} 2(d-k+1)\right| + \left(\sum_{k=1}^{d} 2^k + 1\right) + (B'-B) \qquad (9)$$

여기서 S는 아래와 같다.

$$S = \begin{cases} +1, & \text{if } (A'-A>0) \\ -1, & otherwise \end{cases}$$

그림 2는 d = 2일 경우 디렉터리 풀의 사용 예를 도시한 것이다. d = 3인 E 프로세서는 존재 비트를 사용하는 대신 디렉터리 풀에 저장한다.

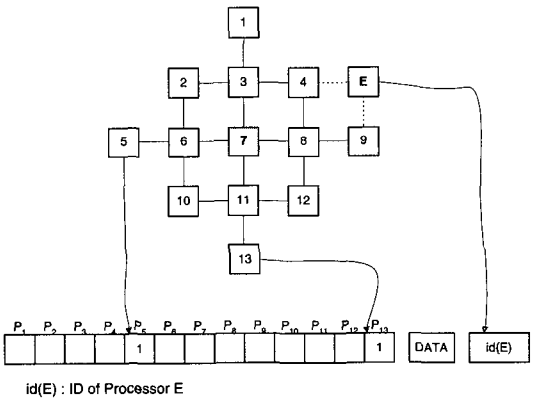


그림 1 동적 제한 디렉터리 기법의 적용 예(d = 2일 경우)

4. 모의 실험

4.1 모의 실험 환경

데이타 배치 방식에 따른 성능을 평가하기 위해 MINT를 사용하였다[6]. 공유 데이타 배치를 위하여 추적 구동(trace-driven) 기법을 사용하였으며 그림 2에 도식된 방식으로 메모리 배치 정보 화일을 생성한다. 이 화일을 이용하여 2.2절에서 기술한 최적, 최다, 연속 알고리즘을 이용하여 데이타 배치 정보 화일을 각각 생성한다. 그림 2는 입력된 병렬 평가 프로그램의 실행 코드를 해석하여 각 처리기들의 실행에서 발생하는 메모리의 참조를 공유 데이타 배치 방식에 따라 모의 실험 하는 과정이다. 표 2는 본 논문에서 사용된 모의 실험의 시스템 파라미터들이다[3,4,5,10].

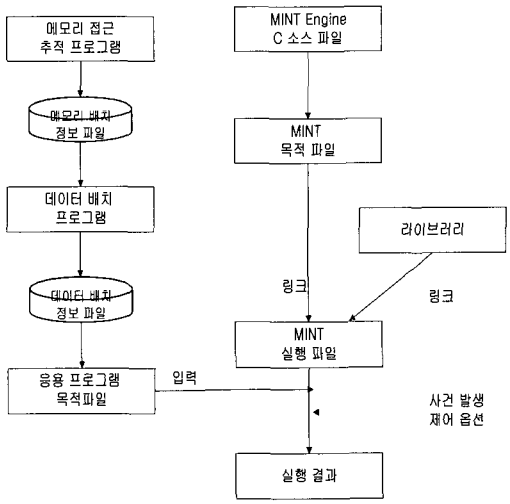


그림 2 모의 실험 방법

표 2 모의 실험 파라미터

하드웨어 비용 관련 인자	비용
메모리 참조	20 사이클
한 노드당 평균 통신망 지연	20 사이클
통신망의 통행량	32 바이트
연속 무효화 메시지 지연	1 사이클
캐쉬 연관도	1
캐쉬 블럭 크기	16/32/64 바이트
캐쉬 크기	128k 바이트

모의 실험에서 8×8의 메쉬 구조를 가진 분산 공유 메모리 시스템을 사용한다. 이 시스템은 처리기 상호간의 통신을 제어하는 네트워크 제어부, 분산 공유 메모리, 일관성 유지를 위한 디렉터리와 디렉터리 풀, 캐쉬 메모리와 처리기 등으로 구성된다.

4.2 평가 프로그램

모의 평가를 위해 사용한 평가 프로그램은 공유 특성을 기준으로 스탠포드 대학의 SPLASH와 로체스터 대학에서 얻은 3개의 병렬 프로그램이다. 데이타 배치 방식은 공유 특성에 많은 영향을 받으므로 공유 특성이 다른 3개의 평가 프로그램을 이용하였다[6,7]. 대부분의 연구 결과에서 모의 실험의 정확성을 부여하기 위해 많은 공유 데이타 블럭을 사용하는 프로그램으로 모의 실험을 실행하였다. FFT 프로그램은 65536 복소수 원소를 1차원 FFT 알고리즘을 수행한다. Sor 프로그램은

640×640 그리드에서 레드-블랙 연속 overrelaxation 알고리즘을 이용하여 금속의 표면온도를 계산한다. Water 프로그램은 N-바디 분자역학을 모의 실험하는 프로그램으로 액체 상태의 물 분자계에서 위치 에너지와 힘을 계산한다. 표 3은 모의 실험에 사용되는 응용 프로그램의 총 메모리 참조 횟수, 공유 데이타의 크기와 메모리의 참조 특성을 나타낸 것이다.

사용한 응용 프로그램의 공유 특성을 보면 Sor와 FFT 프로그램은 임의의 공유 데이타 블럭에 대하여 접근하는 처리기의 수가 대부분 5개나 10개 이하를 각각 차지하여 공유도가 낮은 프로그램으로 분류할 수 있다. Water 프로그램은 임의의 공유 데이타 블럭에 대하여 접근하는 처리기의 수가 대부분 20개 정도이므로 공유도가 높은 프로그램으로 분류할 수 있다[6,7,10].

표 3 평가 프로그램 특성

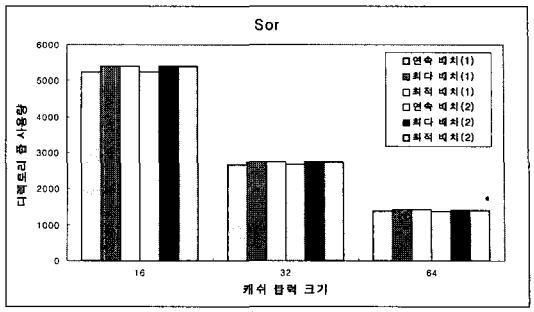
프로그램 이름	Sor	FFT	Water
총 메모리 참조 회수 (단위 : 106)	17.72	208.02	237.98
공유 데이타 크기 (단위 : MB)	2.614	2.003	0.183
Private Read (단위:%)	30.4	55.1	59.9
Private Write (단위:%)	7.1	38.2	22.9
Shared Read (단위:%)	48.6	4.2	15.6
Shared Write (단위:%)	13.9	2.5	1.6
SR : SW	1:0.28	1:0.54	1:0.09

4.3 모의실험 결과

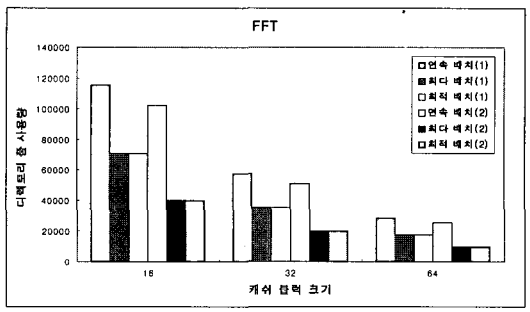
그림 3, 그림 4와 그림 5는 동적 제한 디렉터리 기법에서 최적, 최다, 연속 배치 방식을 이용하여 네트워크 거리를 1과 2로 하여 디렉터리 풀의 최대 사용량, 네트워크 통행량과 디렉터리 풀의 접근 횟수를 각각 측정한 것이다. 캐쉬 블럭 크기에 대한 네트워크 통행량과 디렉터리 풀의 관계 변화를 16바이트, 32바이트와 64바이트에 대하여 각각 모의 실험하였다.

4.3.1 디렉터리 풀의 사용량 평가

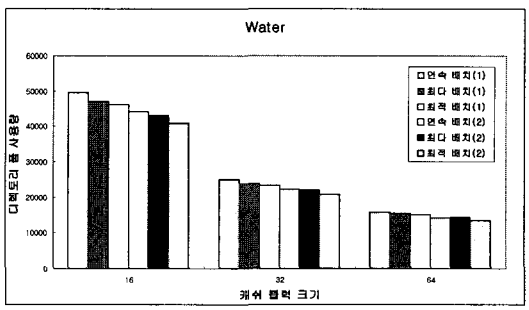
그림 3은 동적 제한 디렉터리 방식에서 각 배치 방법에 따른 디렉터리 풀의 사용량을 측정한 것이다. Sor 프로그램에서는 낮은 공유 특성과 읽기 접근이 쓰기 접근에 비해 월등히 많아서 각 배치 방식에 따른 성능 차



(a) Sor 프로그램



(b) FFT 프로그램



(c) Water 프로그램

그림 3 디렉터리 풀 최대 사용양

이가 거의 없다. 또한 캐쉬 블럭의 크기를 증가시킬 경우 그림 4와 같이 네트워크의 통행량도 감소함을 알 수 있다. 그러나 FFT 프로그램은 공유 특성이 다소 높으므로 최적 배치와 최다 배치에서 비슷한 결과를 가지나 공유 데이타의 사이즈가 크므로 연속 배치와는 상당한 차이가 있다. 쓰기 접근의 비중이 높아서 캐쉬 무효화의 영향으로 캐쉬 블럭의 크기를 증가시킬 경우 통행량이 급격히 증가함을 그림 4에서 관찰할 수 있다. Water 프

로그램은 비교적 높은 공유 특성을 가지므로 배치 방식에 따라 성능 차이가 있으나 공유 데이타의 사이즈가 작아 큰 성능 차이는 없다. 그리고 네트워크의 통행량에 있어서도 공유 데이타의 사이즈가 작으므로 쓰기 접근의 비중이 낮으며 캐쉬 블럭의 크기가 증가하여도 큰 변화가 없다.

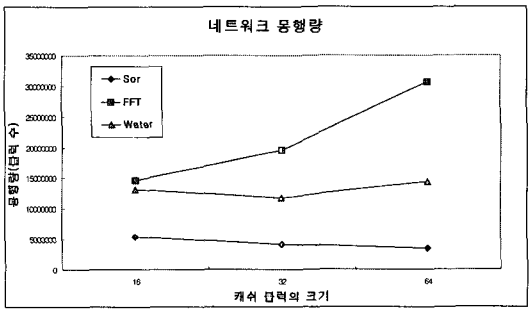
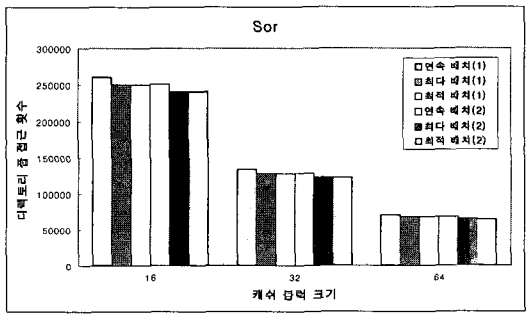
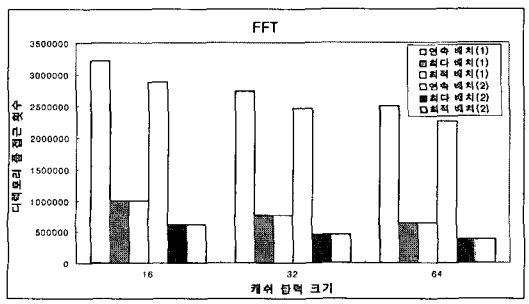


그림 4 네트워크 통행양

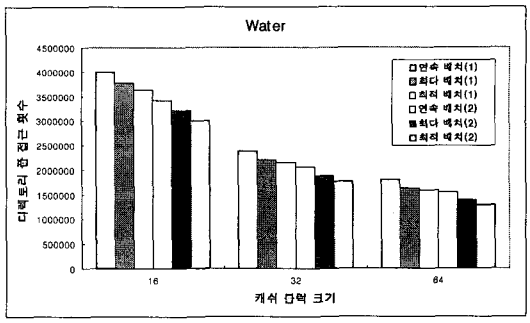
평균적으로 최적 배치, 최다 배치 그리고 연속 배치 순으로 디렉터리 풀의 사용량이 낮아짐을 알 수 있다. 디렉터리 풀의 사용량과 네트워크 통행량을 고려하면 캐쉬 블럭의 크기는 32바이트가 적당함을 알 수 있다.



(a) Sor 프로그램



(b) FFT 프로그램



(c) Water 프로그램

그림 5 디렉터리 풀 접근 횟수

4.3.2 디렉터리 풀 접근 횟수 평가

그림 5는 동적 제한 디렉터리 방식에서 각 배치 방법에 따른 디렉터리 풀의 접근 횟수를 조사한 것이다. Sor 프로그램은 낮은 공유 특성으로 인하여 각 배치 방식에 따른 성능 차이가 거의 없다. FFT 프로그램은 공유 특성이 다소 높고 공유 데이타의 사이즈가 크므로 최적 배치와 최다 배치에서 비슷한 접근 횟수를 가지나 연속 배치에 비교해 상당히 적다. Water 프로그램은 공유 특성은 높지만 공유 데이타의 사이즈가 작으므로 각 배치 방식에 따른 성능 차이가 작다.

각 데이타 배치 기법에 따른 성능 평가에서 공유 특성이 낮은 프로그램에서는 10%이내로 큰차이가 없으나 공유 특성이 높은 프로그램에서는 70% 이상 차이가 있다. 그러므로 지역성을 이용한 데이타 배치는 공유 데이타 접근을 위해 네트워크 비용을 줄이고 디렉터리 풀의 최대 사용량이 적으므로 시스템 확장이 용이하다.

5. 결 론

동적 제한 디렉터리 기법을 이용하여 세가지 데이타 배치 방식에 따른 성능 비교를 하였다. 성능비교는 공유 특성이 다른 3개의 평가 프로그램을 사용하여 디렉터리 풀 사용양, 네트워크의 통행양과 디렉터리 풀의 접근회수를 비교한 결과에서 데이타의 지역성을 높일 수 있는 최적 배치 방식이 기존의 연속 배치 방식보다 3.6배의 성능을 향상시킬 수 있다. 그러므로 기존의 디렉터리 풀을 사용하는 기법보다 데이타의 지역성을 고려하여 데이타를 배치하면 디렉터리 풀의 크기 증가에 따른 오버헤드를 줄일 수 있다. 그리고 데이타의 지역성을 이용하여 존재 비트를 사용하므로 시스템 확장에 유리하고 풀의 탐색 시간이 적어 시스템의 유휴 시간을 줄일 수 있는 장점이 있다.

참 고 문 헌

[1] Jelica Protic, Milo Tomasevic, and 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] Richard Simoni, "Cache Coherence Directories for Scalable Multiprocessors," *Ph.D. Thesis*, Stanford Univ., 1992.

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

[5] Ross Evan Johnson, "Exending The Scalable Coherent Interface for Large Scale Shared Memory Multiprocessors," *Ph.D. Thesis*, Wisconsin-Madison Univ. 1993.

[6] 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.

[7] 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.

[8] 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.

[9] Timothy B. Brecht, "Multiprogrammed Parallel Application Scheduling in NUMA Multiprocessors," Ph.D. Dissertation - CSRI Technical Report CSRI-303, pp.79-110, Jun. 1994.

[10] 이동광, 권혁성, 안병철, "메쉬 구조를 가진 분산 공유 메모리의 효과적인 캐쉬 일관성 유지기법", 한국정보과학회 논문지, Vol. 24, No. 9, 1997.

이 동 광

1985년 영남대학교 공과대학 전자공학과 졸업. 1989년 영남대학교 공과대학원 전자공학과(컴퓨터전공)졸업. 1999년 2월 영남대학교 컴퓨터공학과(컴퓨터시스템전공) 대학원 졸업, 공학박사. 1992년 ~ 현재 경북전문대학 컴퓨터정보과 조교수, 멀티미디어기술사, (주)시텍 대표이사. 관심분야는 멀티미디어 시스템, 멀티스래드 시스템

권 혁 성

1995년 영남대학교 컴퓨터 공학과 졸업. 1997년 영남대 대학원 컴퓨터 공학과 졸업. 1997년 ~ 현재 LG전자 디스플레이 제품 연구소 주임 연구원. 관심 분야는 병렬 컴퓨터 시스템, 이미지 처리, 실시간 멀티미디어 시스템

안 병 철

1976년 영남대학교 전자공학과 졸업. 1986년 오레건 주립대 전기 및 컴퓨터 공학과 석사학위취득. 1989년 오레건 주립대 전기 및 컴퓨터 공학과 박사학위 취득. 1978년 ~ 1984년 국방과학연구소 연구원. 1989년 ~ 1992년 삼성전자 컴퓨터 부문 수석 연구원. 1992년 ~ 현재 영남대학교 공과대학 컴퓨터공학과 부교수. 관심분야는 컴퓨터구조, 그래픽스, 멀티 미디어 및 실시간 운영체제