소프트웨어 분산공유메모리 시스템을 위한 적응적 선인출 기법

(An Adaptive Prefetching Technique for Software Distributed Shared Memory Systems)

이 상 권 † 윤 희 철 †† 이 준 원 ††† 맹 승 렬 †††

(Sang-Kwon Lee) (Hee-Chul Yun) (Joonwon Lee) (Seungryoul Maeng)

요 약 공유가상메모리 시스템은 적은 비용으로 고성능 계산 능력을 제공하지만, 메모리 접근 지연시간이 길다는 문제점이 있다. 일반적으로 이 지연시간은 공유 데이타에 대한 반복적인 무효화 작업에 의해 일어난다. 공유 데이타들은 동기화를 통해서 접근되고 쓰레드들은 반복적 패턴에 의해 동기화 되기 때문에, 반복성에 기반한 선인출 기법은 메모리 지연시간을 효과적으로 줄일 수 있다. 본 논문에서는 동기화 변수별로 접근 기록을 분석해서 미래의 메모리 접근을 예측하는 선인출 기법을 제안한다. 제안하는 기법은 8노드 클러스터 상에서 SPLASH-2 응용들을 실행시켜 성능을 측정하였다. 그 결과, 제안하는 기법이 34%~45% 정도의 메모리 접근 지연시간을 감소할 수 있었다.

Abstract Though shared virtual memory (SVM) systems promise low cost solutions for high performance computing, they suffer from long memory latencies. These latencies are usually caused by repetitive invalidations on shared data. Since shared data are accessed through synchronizations and the patterns by which threads synchronizes are repetitive, a prefetching scheme based on such repetitiveness would reduce memory latencies. Based on this observation, we propose a prefetching technique which predicts future access behavior by analyzing access history per synchronization variable. Our technique was evaluated on an 8-node SVM system using the SPLASH-2 benchmark. The results show that our technique could achieve 34% - 45% reduction in memory access latencies.

1. 서 론

최근 고성능 마이크로 프로세서와 고속 네트워크의 등장으로 인해서 NOW(Networks Of Workstations)와 같은 클러스터 시스템을 병렬 처리에 사용하고자 하는 연구들이 활발히 진행중이다. 공유가상메모리(SVM: Shared Virtual Memory) 시스템은 가상메모리 하드웨어를 사용하여 기계들 간의 공유메모리를 제공한다[1, 2,3]. 하지만 이들 시스템은 원거리 데이타 접근시에 접근 지연시간이 길다는 문제점이 있다.

접근 지연시간을 줄이기 위해서 대부분의 공유가상메모리 시스템에서는 다음의 방법들을 사용한다. 첫째, 캐쉬를 사용해서 원거리 접근을 지역 접근으로 변환시킨다. 둘째, LRC(Lazy Release Consistency)[4]와 같은 완화된 메모리 모델을 사용해서 통신을 최대한 미룬다. 셋째, 다중 기록자(multiple writer) 프로토콜을 사용해서 거짓 공유(false sharing)의 효과를 줄인다[3]. 하지만, 이들 기법으로도 해결할 수 없는 접근 지연시간은 여전히 남고, 이것은 응용 프로그램의 성능을 떨어트리는 주원인이 된다.

공유가상메모리 시스템에서 원거리 데이타 접근 지연시간을 줄이는 효과적인 방법으로 선인출(prefetching) 기법이 있다. 선인출 기법은 실제 접근이 발생하기 전에 데이타를 미리 가져와서 원거리 접근 지연시간을 줄인다. 하지만, 선인출 기법을 통해 공유가상메모리의 성능을 향상시키기 위해선 (1) 미래에 일어날 데이타 접근을

† 비 회 원 : 한국과학기술원 전산학과
sklee@camars.kaist.ac.kr
†† 비 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 연구원
hcyun@etri.re.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
joon@camars.kaist.ac.kr
maeng@camars.kaist ac.kr
논문접수 : 2001년 4월 13일
심사완료 : 2001년 7월 26일

정확히 예측할 수 있어야 하고 (2) 선인출로 인한 부담이 적어야 한다.

본 논문에서는 미래의 데이타 접근을 보다 잘 예측할 수 있는 새로운 선인출 기법을 제안한다. 이 방식의 기본 아이디어는 다음과 같다: 배리어를 시작하기 전에 동일한 배리어 변수에 의해 보호되는 이전 두 배리어에서의 접근 기록을 비교해서 유사하다고 판단되면, 이번에도 그 접근 패턴이 반복될 것이라고 예측한다. 유사하지 않다고 판단되면 동일한 배리어 변수에 의해 보호되는 직전 배리어의 접근이 같은 스트라이드를 가지는 접근이었는지 살펴본다. 그렇다면 이번에도 같은 스트라이드를 가진 접근일 것이라고 예측한다. 만약 두 방법 모두 아니라면 선인출을 하지 않는다.

제안된 선인출 기법의 효과를 측정하기 위해 Switched Fast Ethernet으로 연결된 8대의 PC 클러스터에서 네 개의 SPLASH 2 [7] 응용을 실행시켰다. 세 개의 응용에 대해서 페이지 폴트 시간을 34%~45% 정도 줄였고, 8%~16% 정도의 속도 향상을 가져왔다. 나머지 한 응용은 패턴이 불규칙하기 때문에 선인출을 통해 효과를 볼 수 없었다. 기존 선인출 기법들과의 자세한 비교를 통해서 제안된 선인출 기법이 예측을 보다 정확하게 한다는 것을 알 수 있었다.

논문의 구성은 다음과 같다. 2장에서는 공유가상메모리 시스템의 성능 측정을 통해 원거리 데이타 접근 지연시간이 성능에 어떤 영향을 미치는지 살펴본다. 3장에서는 본 연구와 관련된 연구들을 설명한다. 4장에서는 원거리 접근 지연시간을 줄일 수 있는 새로운 선인출 기법을 제안한다. 5장에서는 제안된 선인출 기법의 성능 측정 결과를 설명하고, 6장에서 결론을 맺는다.

2. 실험 환경 및 예비 실험 결과

원거리 메모리 접근 지연시간이 공유가상메모리 시스템의 성능에 어떤 영향을 미치는지 측정하기 위해서, 8대의 PC로 구성된 리눅스 클러스터 상에서 실험을 하였다. 각 PC는 Pentium III 500 MHz CPU와 256 MB 메인 메모리를 가지며, 100 Mbps Switched Fast Ethernet으로 연결되었다.

2.1 KDSM: KAIST Distributed Shared Memory

KDSM(KAIST Distributed Shared Memory) 시스템은 Linux 2.2.13 상에서 실행되는 사용자수준 라이브러리(user-level library)로 구현되었다. 프로세스 간의 통신은 TCP/IP를 통해서 이루어지고, 비동기 메시지의 처리를 위해서 SIGIO 시그널을 가로채는 방식을 사용한다. KDSM은 페이지 기반 무효화 프로토콜(page- based invalidation protocol), 다중읽기 다중쓰기(multiple reader multiple writer) 프로토콜을 바탕으로 HLRC(Home-base Lazy Release Consistency)[8] 프로토콜을 구현한다. 원거리 페이지를 저장하는 캐쉬 페이지들은 다음과 같은 네 가지 상태 중 하나를 가진다: RO(읽기전용 상태), RW(읽기쓰기가 가능한 상태), INV (무효화 상태), 그리고 UNMAP(메모리 맵핑이 안 된 상태). KDSM은 프로그래머를 위해서 다양한 홈 할당 방식을 제공하지만, 모든 실험은 페이지 단위의 라운드 로빈 방식을 사용하였다. KDSM의 주요 연산에 대한 시간은 다음과 같다.

표 1 KDSM의 기본 연산 비용

연산	비용(μs)
4KB 페이지 인출	1047
Lock을 얻음	259
8 프로세서에서 Barrier	1132

2.2 응용

SPLASH-2[7] 벤치마크 중 네 가지 응용을 사용하여 실험을 하였다. BARNES는 4K 개의 바디를 가진 천체의 상호작용을 모의실험하는 BARNES-Hut 방법을 구현하는 응용이다. FFT는 256K 개의 데이타 포인트에 대한 3-D Fast Fourier Transform을 수행한다. OCEAN은 소용돌이와 해류에 기반해서 대규모 해양의 움직임을 모의실험하는 응용으로, 258x258 크기의 해양을 실험한다. RADIX는 2^{20}개의 정수 키를 래딕스 정렬한다. BARNES와 FFT는 CVM[9]에서 가져온 것으로 소프트웨어 분산공유메모리를 위해 재구성된 버전이다. 이 두 응용은 아주 규칙적인 메모리 접근 패턴을 보인다. OCEAN과 RADIX는 SPLASH-2 에서 KDSM으로 직접 포팅해서 재구성하지 않은 버전이다. RADIX는 아주 불규칙적인 패턴을 보이고, OCEAN은 규칙적인 것과 불규칙적인 것의 중간 정도의 패턴을 보인다.

2.3 예비 실험 결과

그림 1은 응용 실행 시간을 100으로 했을 때, 각 부분별로 걸린 시간을 보여준다. 실행 시간은 다음과 같이 세 부분으로 나누어 진다: (1) 응용 프로그램 수행에 걸린 시간 (2) 동기화에 걸린 시간 (락과 배리어) (3) 페이지 폴트 처리에 걸린 시간 (페이지 폴트 처리 시간은 지역 폴트 및 원거리 폴트를 모두 포함하지만 지역 폴트에 걸리는 시간은 거의 무시할 정도로 작다.) KDSM 시스템은 메시지가 도착하면 SIGIO 시그널에 의해 인

터럽트가 걸리는 방식으로 구현되었기 때문에 공유가상 메모리 프로토콜 처리에 걸리는 시간은 따로 분리될 수 없고 세가지 경우 모두에 포함되어 있다. 그림 1에서 대부분의 응용들이 동기화와 원거리 데이타 접근으로 인한 지연으로 많은 시간을 (26%~62%) 보내는 것을 알 수 있다. 선인출 기법은 원거리 데이타 인출에 드는 부담을 완화시키는데 사용될 수 있다.

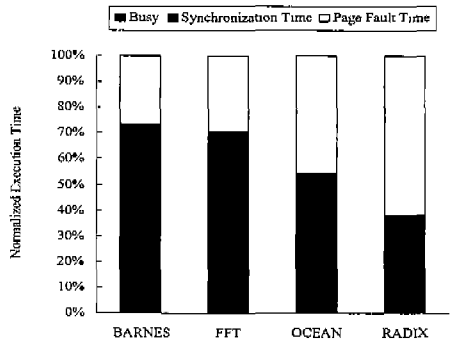

그림 1 8프로세서 클러스터 상에서 KDSM 실행 결과

3. 관련 연구

B+ 기법은 동기화 지점에서 무효화된 페이지들을 선인출한다[5]. 무효화가 접근 패턴을 잘 예측하는 응용에 대해서는 성능이 좋지만, 그렇지 않은 응용에 대해서는 불필요한 선인출로 인해 성능이 나쁘다. 즉, B+ 기법은 응용의 패턴에 따라 적응적으로 반응할 수 없기 때문에 성능이 응용의 패턴에 좌우된다는 문제점을 가진다.

Amza 등은 dynamic aggregation이라는 기법을 제안하였다[10]. 이 기법은 데이타 접근 패턴에 따라 페이지 인출(fetch)의 단위를 동적으로 변화시켜 한번에 여러 페이지를 인출함으로써 선인출과 비슷한 효과를 가진다.

Karlsson 등은 데이타 접근 패턴(생산자-소비자 패턴)을 활용하는 역사 선인출 기법을 제안하였다[11]. 만약 패턴이 검출되지 않으면, 페이지 폴트가 났을 때 연속된 페이지들을 함께 인출하는 순차적 선인출을 사용한다. 이 기법은 생산자-소비자 패턴만을 검출할 수 있기 때문에 약간 제한적이다. 또한 반드시 역사 선인출이나 순차적 선인출 둘 중 한 방식으로 선인출을 하기 때문에 불규칙적인 응용에 대해서 불필요한 선인출을 발생시킬 가능성이 크다.

Adaptive++ 기법은 repeated-phase 모드와 repeated-stride 모드를 적응적으로 선택해서 선인출한다[6]. Adaptive++는 기본적으로 Karlsson이 제안한 방법과 유사하지만, 생산자-소비자 패턴에 국한되지 않았다는 점에서 좀 더 일반적이고, 패턴이 검출되지 않을 때는 선인출을 하지 않는다는 점에서 부담이 적다는 장점을 가진다. 하지만, 배리어 변수를 고려하지 않은 채 이전 배리어(들)의 접근 기록에 기반해서 선인출을 판단하기 때문에 데이타 접근을 정확히 예측하지 못한다.

Mowry 등은 컴파일러 및 프로그래머가 소스 코드에 직접 선인출을 삽입했을 때 선인출의 효과가 어느 정도인지에 대한 연구를 하였다[12]. 이 연구의 결과는 선인출을 사용해서 얻을 수 있는 성능 향상의 한계값을 제시한다고 볼 수 있다.

4. 적응적 선인출 기법

4.1 선인출 아이디어

본 논문에서 제안하는 선인출 기법은 동기화 변수별로 접근 패턴을 분석해서 역사 모드 선인출과 스트라이드 모드 선인출을 적응적으로 선택한다. 패턴이 발견되지 않을 때에는 선인출을 하지 않는다. 이 기법은 배리어 및 락 변수 모두에 적용이 가능하지만, 본 논문에서는 배리어 변수만을 다룬다. 구체적인 설명에 앞서, 간단한 예제를 통해 동기화 변수를 고려해야 할 필요성에 대해 설명한다.

4.1.1 간단한 병렬 프로그램 예

그림 2는 반복적 알고리즘을 수행하는 간단한 병렬 프로그램의 예를 보여준다. 이 프로그램은 세 개의 배리어 변수로 보호되는 세 개의 프로그램 영역으로 구성된다. compute_a(), compute_b(), compute_c()는 각각 배리어 변수 1, 2, 3에 의해 보호된다. 그림 3은 이 프로그램이 실행되는 과정을 보여준다. compute_a()는 원거리 페이지 1, 2, 5, 7을, compute_b()는 원거리 페이지 8, 9, 11, 14를, compute_c()는 원거리 페이지 15, 16, 18, 21을 각각 접근한다. 루프가 반복됨에 따라서 각 영역에서 동일한 페이지 접근 패턴이 반복된다. 따라서 그림 4처럼 동일한 배리어 변수에 의해 보호되는 영역별로 실행 기록을 살펴보면 동일한 접근 패턴이 반복됨을 알 수 있다.

```
int i;
for (i = 0; i < iterations; i++)
{
  barrier(1);
  compute_a();
  barrier(2);
  compute_b();
  barrier(3);
  compute_c();
}
```

그림 2 반복적 알고리즘을 수행하는 간단한 병렬 프로그램의 예

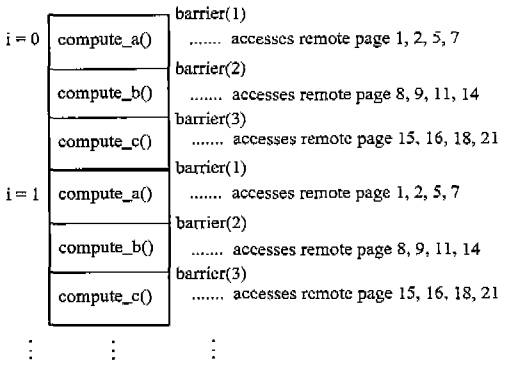

그림 3 반복적 알고리즘의 실행 패턴

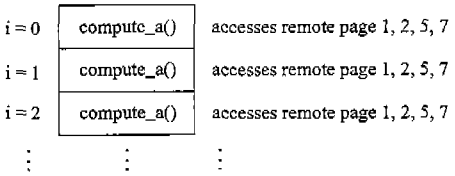

그림 4 배리어 변수 1에 의해 보호되는 실행 영역의 페이지 접근 패턴

4.1.2 역사 모드 선인출

역사 모드 선인출은 다음과 같이 동작한다: 동일한 배리어 변수에 의해 보호되는 마지막 두 단계의 접근 패턴을 비교했을 때 서로 유사하다고 판단되면, 이 모드는 미래의 접근도 이 패턴을 따를 것이라고 예측한다 (phase: 배리어와 배리어 사이의 실행 구간을 phase, 즉 단계라고 한다).

배리어 변수별로 접근 역사를 기록하기 위해서 last, before_last, expected라 불리는 세 개의 리스트를 유지한다. last는 직전 단계에서 폴트가 난 페이지들의 번호를 가지고, before_last는 두 단계 앞에서 폴트가 난 페이지들의 번호를 가진다. 각 리스트는 폴트가 난 순서대로 페이지 번호를 저장한다 (단, 락에 의해 보호되는 임계영역 내에서 폴트가 난 페이지는 포함하지 않는다). last와 before_last의 유사성(similarity)은 다음 수식과 같이 계산된다:

$$similarity = \frac{|last \cap before\_last|}{|last| + |before\_last| - |last \cap before\_last|}$$

만약 유사성이 50% 이상이면 두 접근 패턴이 유사하다고 판단하고, 이번 단계에서도 유사한 접근 패턴이 반복되리라고 예측한다. 두 리스트에 공통으로 포함된 페이지 리스트 ($last \cap before\_last$)를 expected 리스트에 저장하고 이것을 선인출의 대상으로 한다.

4.1.3 스트라이드 모드 선인출

스트라이드 모드 선인출은 역사 모드 선인출과 마찬가지로 배리어 변수별로 접근 기록을 분석한다. 배리어를 시작할 때 last의 페이지들을 정렬하여 temp 리스트에 저장한다. temp의 페이지 중에서 가장 자주 나오는 스트라이드 값이 전체 스트라이드 중 어느 정도 나오는지 빈도(frequency)를 계산한다. 스트라이드의 빈도가 50% 이상이면 다음 단계에서도 이 스트라이드 값만큼 분리되어 페이지들이 접근될 것이라고 예측한다.

4.1.4 선인출 모드의 선택

각 배리어 시작시 어떤 선인출 모드를 선택할지 결정하는 것은 다음과 같다. 앞서 설명한 방법대로 역사 모드의 유사성과 스트라이드 모드의 빈도를 각각 계산한다. 계산된 유사성과 빈도를 비교해서 큰 쪽의 모드 선택하는데, 값이 같을 때는 역사 모드를 선택한다. 그리고 선택된 값이 50% 이상일 때만 해당 선인출 모드로 동작하고, 50% 이하이면 선인출을 하지 않는다. 즉, 특정한 접근 패턴을 보이지 않는다고 판단되면 선인출을 하지 않음으로써 불필요한 선인출을 막는다.

4.2 구현

4.2.1 배리어 변수별로 페이지 접근 기록을 유지하기

배리어 변수별로 페이지 접근 기록을 유지하기 위해서 앞서 설명한 last, before_last, expected 리스트 외에 current 리스트를 유지한다. current는 현재 배리어 (예를 들어, 배리어 변수 bar1) 내에서 폴트가 난 페이지들의 리스트를 가지며 폴트가 난 순서대로 기록된다. 응용 프로그램 실행 중 페이지 폴트가 발생하는 경우는 다음과 같다:

- RO 상태인 지역 페이지에 대한 쓰기
- INV 상태인 원거리 페이지에 대한 읽기/쓰기
- RO 상태인 원거리 페이지에 대한 쓰기

이 중 첫번째와 두번째 경우에만 페이지 번호를 current에 추가시킨다. 첫번째 경우를 포함시키는 이유는 접근 기록에서 스트라이드를 계산할 때 원거리 페이지뿐 아니라 지역 페이지의 접근도 고려하기 위해서이다. 응용 프로그램의 페이지 접근 패턴은 홈의 할당과 관계없는 성질이기 때문에 지역 페이지에 대한 접근도 고려되어야 한다. bar1의 현재 단계가 끝날 때, 관련된 모든 리스트들을 적절하게 재구성한다.

4.2.2 선인출을 하는 시기와 방법

역사 모드 선인출이 선택되었을 때, 선인출은 두 단계에 걸쳐 일어난다.

- 배리어를 시작할 때, expected 리스트의 처음 25 개의 페이지에 대해서 선인출 요청 메시지인 PREFETCH

를 보낸 후 expected 리스트에서 제거한다. 일반적으로 다른 기법들이 선인출 메시지를 보낸 후 계산을 계속하는 것과는 달리, PREFETCH 메시지를 보낸 후 페이지들이 도착하기를 기다린다. 그 이유는 마지막 페이지에 대한 PREFETCH를 보낸 후 쯤에는 그 응답 메시지인 PREFETCHGRANT가 연속적으로 도착하기 때문에 실제로 계산과 통신이 중첩되지 않기 때문이다.

- 원거리 페이지 폴트가 발생할 때마다 (단, 락에 의해 보호되는 임계영역 내에서 폴트가 났을 때는 제외), 해당 페이지에 대한 페이지 요청 메시지인 GETP를 보낸 후, expected 리스트에서 처음 5 개의 페이지에 대해서 PREFETCH를 보내고 리스트에서 제거한다. 배리어의 경우와 같은 이유로 PREFETCHGRANT가 도착하기를 기다린다.

스트라이드 모드 선인출이 선택되었을 때는 다음과 같다. 원거리 페이지 폴트가 발생할 때마다 (단, 락에 의해 보호되는 임계영역 내에서 폴트가 났을 때는 제외), 그 페이지에서 정해진 스트라이드 만큼 떨어진 연속한 5 개의 페이지를 선인출 대상으로 한다. 예를 들어, 스트라이드가 2이고 페이지 100에서 폴트가 났으면 102, 104, 106, 108, 110 페이지를 대상으로 한다. 실제 선인출 메시지는 유효한 공유메모리 주소를 가지면서 INV 상태인 원거리 페이지에 대해서만 전송한다.

선인출을 통해 가져온 페이지들은 RO 상태가 아닌 INV 상태로 설정하고 (이렇게 하는 이유는 선인출된 페이지들이 실제 접근되었는지 검사하기 위해서이다), prefetched란 플래그 값을 사용된 선인출 모드로 설정해서 해당 페이지가 선인출된 페이지임을 표시한다. 이후에 그 페이지에 폴트가 발생하면, INV 상태를 폴트 종류에 따라 RO/RW 상태로 바꾸고 계산을 진행시킨다.

4.3 Adaptive++ 기법과의 비교

Adaptive++[6] 기법은 여러 가지 면에서 본 논문에서 제안하는 선인출 기법과 유사하지만, 다음과 같은 두 가지 문제점을 가진다.

1. Adaptive++ 기법은 대부분의 응용들이 연속적인 (consecutive) 패턴을 보이던지 두 패턴이 교대로 발생하는 (alternating) 형태라고 가정하고, 접근 기록을 분석할 때 배리어 변수에 대해 어떤 고려도 하지 않는다. 따라서, Adaptive++ 기법은 루프 내에 세 개 이상의 배리어 변수를 가지는 응용들에 대해서 미래의 접근 패턴을 제대로 예측할 수 없다. 그림 2의 응용을 예로 설명하면 다음과 같다(그림 5). Adaptive++ 기법은 barrier(3)이 실행되기 직전에 compute_a()와 compute_b()의 접근 패턴을 비교한다. 이 둘이 비슷하면 compute_a()에서 폴트가 난 페이지들을 expected 리스트로 설정하고, 이 둘이 비슷하지 않으면 compute_b()에서 폴트가 난 페이지들을 expected 리스트로 설정한다. 따라서 barrier(3)에서의 접근 패턴을 제대로 예측하지 못한다.

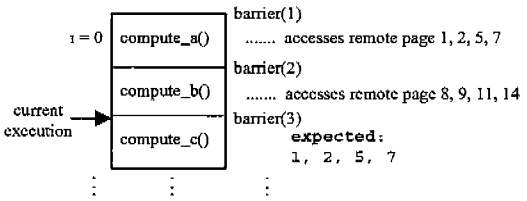

그림 5 Adaptive++ 기법이 미래의 접근 패턴을 잘 못 예측하는 경우 1

2. Adaptive++ 기법은 그림 6과 같이 두 패턴이 교대로 발생하는 형태를 보이는 응용에 대해서도 예측을 잘 못한다 (실제 이 패턴은 BARNES의 실행 패턴이다). 숫자는 페이지 번호이고, 괄호 안의 문자는 지역 폴트(H)인지 원거리 폴트(R)인지를 표시한다. Adaptive++ 기법은 barrier(13)이 시작할 때 repated-phase 모드가 아닌 repeated-stride 모드를 선택하기 때문에 barrier(13)의 패턴을 제대로 예측하지 못한다. 이것은 바로 앞 단계에서 repeated-phase 모드의 유용성(usefulness)이 0인데 반해 (모든 폴트가 지역 페이지에 대한 폴트이기 때문에 유용한 선인출이 없다), 스트라이드 8의 빈도가 50% 이상이기 때문이다.

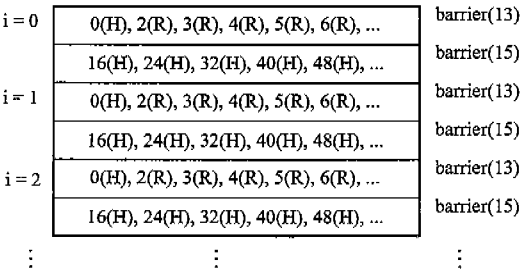

그림 6 Adaptive++ 기법이 미래의 접근 패턴을 잘 못 예측하는 경우 2

5. 실험 결과

동일한 플랫폼 상에서 선인출 기법들의 성능을 비교하기 위해서, B+, Adaptive++, 본 논문에서 제안하는 기법을 KDSM 상에 구현하였다. 구현상의 주목할 점들은 다음과 같다: 첫째, B+, Adaptive++는 LRC 하에서

diff를 선인출하지만, 제안하는 기법은 HLRC 하에서 전체 페이지를 선인출한다. 같은 조건하의 실험을 위해서 세 기법 모두 페이지를 선인출하게 구현하였다. 둔째, B+ 및 Adaptive++ 방식은 선인출 메시지를 보낸 후 메시지가 도착할 때까지 대기하지 않고 계산을 계속 진행한다. 하지만 앞서 설명한 것처럼, 실제 통신과 계산이 거의 중첩되지 않기 때문에 모든 기법들을 선인출 메시지에 대한 응답이 올 때까지 대기하도록 구현하였다.

5.1 선인출의 효과

선인출 기법들의 효과를 측정하기 위해 선인출 coverage 및 hit ratio를 살펴본다. *coverage*란 전체 원거리 페이지 폴트 중 선인출을 통해서 제거된 페이지 폴트의 비율을 뜻한다. *hit ratio*란 전체 선인출 메시지 중 유효한 선인출의 비율을 뜻한다. 유효한 선인출이란 페이지 폴트가 일어나기 전에 성공적으로 선인출이 되어 페이지 폴트를 줄인 것을 말한다.

표 2는 사용된 각 선인출 기법들의 coverage 및 hit ratio를 보여준다 (B+는 B+ 기법을, A++는 Adaptive++ 기법을, New는 본 논문에서 제안하는 선인출 기법을 표시한다). Coverage 및 hit ratio는 다음과 같이 계산된다: Coverage=유용한 선인출/전체 페이지 폴트, Hit Ratio =유용한 선인출/전체 선인출.

표 2 선인출 기법들의 선인출 coverage 및 hit ratio

응용	선인출 기법	전체 원거리 페이지 폴트	전체 선인출	유용한 선인출	coverage	hit ratio
BARNES	B+	20084	19497	18111	90.18	92.89
	A++	20084	1414	1390	6.92	98.30
	New	20084	14558	14539	72.39	99.87
FFT	B+	19433	15960	15163	78.03	95.01
	A++	19433	10335	10164	52.30	98.35
	New	19433	12266	12260	63.09	99.95
OCEAN	B+	29501	23827	11728	39.75	49.22
	A++	29501	8806	8216	27.85	93.30
	New	29501	21881	19510	66.13	89.16
RADIX	B+	20156	18094	1317	6.53	7.28
	A++	20156	21	20	0.10	95.24
	New	20156	24	23	0.11	95.83

B+ 기법은 BARNES, FFT, RADIX의 경우에 coverage가 세 기법 중 가장 좋은데, 특히 BARNES와 FFT의 경우 coverage가 각각 90%, 78%로 아주 높다. 이것은 BARNES와 FFT의 경우에 무효화 메시지가 미래의 접근 패턴을 잘 예측할 수 있다는 것을 의미한다. BARNES와 FFT의 hit ratio는 각각 90% 이상으로 매우 높고, RADIX는 7%로 아주 낮으며, OCEAN은 49%이다. RADIX의 coverage가 낮은 이유는 페이지 접근 패턴이 매우 불규칙적이기 때문이다.

A++ 기법은 모든 응용에 대해서 coverage가 세 기법 중 가장 낮다. BARNES가 패턴이 일정한 응용임에도 불구하고 coverage가 낮은 것은 4.3절에서 설명한 것처럼 배리어 변수를 고려하지 않기 때문이다. RADIX의 coverage가 낮은 것은 RADIX가 불규칙적인 응용이기 때문으로 BARNES의 경우와는 틀리다. RADIX와 같이 불규칙적인 응용은 선인출을 하지 않는 것이 오히려 도움이 된다. 하지만, hit ratio가 모든 응용에 대해서 90% 이상으로 일단 선인출을 하면 대부분 성공한다는 의미가 된다.

본 논문에서 제안하는 기법은 OCEAN에 대해서 coverage가 66%로 세 기법 중 가장 좋고, BARNES와 FFT에 대해서 72%, 63%로 두번째로 좋다. RADIX는 불규칙한 응용이기 때문에 선인출을 거의 하지 않는다. BARNES, FFT에 대해서 전체 선인출의 90% 이상이 역사 모드를 동작하고 hit ratio가 거의 100%이다. OCEAN에 대해서 60% 이상이 역사 모드로 동작하고 hit ratio는 거의 90%이다. 배리어 변수를 고려해서 접근 패턴을 분석했을 때, 패턴이 일정한 응용이라면 매 단계마다 동일한 패턴을 보이고, 따라서 역사 모드로 동작할 확률이 커진다. 이것이 New 기법이 스트라이드 모드보다 역사 모드 선인출을 더 많이 사용하고, 역사 모드의 hit ratio가 높은 이유이다.

5.2 네트워크 트래픽

표 3은 선인출을 사용하지 않았을 때(Base)와 각 기법들을 사용했을 때 네트워크에 전송된 모든 메시지의 수와 양을 보여준다. 전체적으로 선인출을 사용했을 때가 사용하지 않을 때보다 메시지 수와 양이 늘어난다. B+ 기법이 OCEAN과 RADIX에 대해서 특히 많이 늘어나는데, 이것은 불필요한 선인출을 많이 하기 때문으로 두 경우 모두 hit ratio가 낮다는 것에서 확인할 수 있다.

표 3 네트워크 트래픽

응용	메시지 수				메시지 크기(KBytes)			
	Base	B+	A++	New	Base	B+	A++	New
BARNES	43005	45777	43053	43043	92647	98332	92745	92725
FFT	78290	79884	78632	78302	236299	239568	237001	236324
OCEAN	88488	110136	89146	93116	243096	287566	244436	252625
RADIX	46702	80261	46705	46699	112257	181073	112257	112241

5.3 전체 성능

그림 9는 각 응용에 대해서 선인출을 하지 않았을 때를 기준으로 정규화시킨 실행시간을 보여준다. 그림의 각 항목의 의미는 다음과 같다: Page Fault는 페이지 폴트 처리 시간, Prefetch Overhead는 선인출을 하기 위해 추가된 시간, Synchronization는 락과 배리어 시간, Busy는 그 외 시간을 의미한다. 전체적으로 선인출

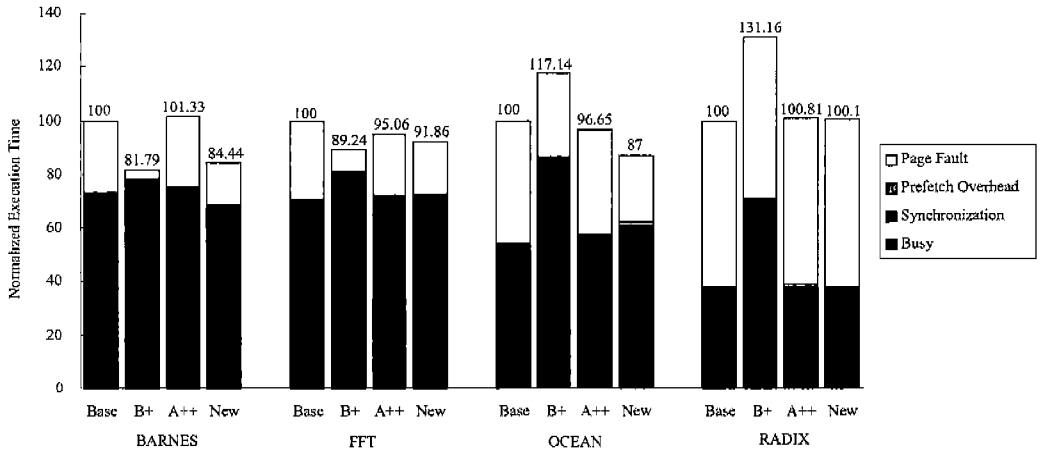

그림 9 8 프로세서 상에서 실행시켰을 때 정규화된 실행 시간

기법들을 사용했을 때 페이지 폴트 처리 시간이 줄어드는 것을 볼 수 있다. 단, RADIX의 경우 선인출을 통해 얻을 수 있는 효과가 적기 때문에 페이지 처리 시간이 거의 줄지 않는다. 선인출로 인한 부담은 모든 기법에서 그리 크지 않다. B+는 모든 응용에 대해서 부담이 0이고, A++는 0.3%~0.8%, 본 논문에서 제안하는 기법은 0.2%~0.8% 정도의 부담을 가진다.

B+ 기법은 BARNES와 FFT에 대해서 성능 향상이 각각 18%, 11%로 세 기법 중 가장 좋은 성능을 보이는 반면, OCEAN과 RADIX에 대해서 -17%, -31%로 가장 나쁜 성능을 보인다. BARNES와 FFT 처럼 무효화가 미래의 접근 패턴을 잘 나타내는 응용들은 페이지를 선인출 함으로써 성능이 향상을 기대할 수 있지만, 그렇지 않은 응용들은 불필요한 선인출로 인해 상당한 성능 저하가 일어난다. 한가지 주목할 점은 모든 응용에 대해서 동기화에 걸린 시간이 증가하는데, 이것은 베리어 시작 시점에 무효화된 페이지 전체를 선인출하기 때문이다.

A++ 기법은 OCEAN과 FFT에 대해서 각각 3%, 5% 정도 성능이 향상되지만, BARNES와 RADIX에 대해서는 -1% 정도 성능이 떨어졌다. BARNES와 RADIX는 앞서 보았듯이 coverage가 낮기 때문에 선인출을 통해 성능 향상을 기대하기 힘들다.

본 논문에서 제안하는 기법은 BARNES, OCEAN, FFT에 대해서 각각 16%, 13%, 8% 정도 성능이 향상되고, RADIX에 대해서는 성능 향상이 없다. 앞서 보았듯이 BARNES, OCEAN, FFT의 coverage 및 hit ratio가 높기 때문에 이런 성능 향상은 미리 예측할 수 있었다.

6. 결 론

본 논문에서는 미래의 데이타 접근을 보다 잘 예측할 수 있는 새로운 선인출 기법을 제안하였다. 제안된 기법은 동기화 변수별로 페이지 접근 패턴을 분석함으로써 반복적인 작업을 하는 응용에 대해서 데이타 접근을 정확히 예측할 수 있다. 8노드 클러스터 상에서 4개의 SPLASH 2 응용을 실행시킨 결과, 세 응용에 대해서 페이지 폴트 시간을 34%~45% 정도 줄였으며, 8%~16% 정도의 속도 향상을 가져왔다.

참 고 문 헌

[1] K. Li and P. Hudak. Memory Coherence in Shared Memory Systems. ACM Transaction on Computer Systems, 7(4), November 1989.

[2] J .B. Carter, J. K. Bennette, and W. Zwaenepoel. Implementation and Performance of Munin. In Proceedings of the 13th SOSP, February 1991.

[3] C. Amza, S. Dwarkadas, P. Keleher, A. L. Cox, and Z. Zwaenepoel. Treadmarks: Shared Memory Computing on Networks of Workstations. IEEE Computer, 29(2), February 1996.

[4] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In Proceedings of the 19th ISCA, May 1992.

[5] R. Bianchini, L. I. Kontohanassis, R. Pinto, M. Maria, M. Abud, and C. L. Amorim. Hiding Communication Latency and Coherence Overhead in Software DSMs. In Proceedings of the 7th ASPLOS, October 1996.

[6] R. Bianchini, R. Pinto, and C. L. Amorim. Data Prefetching for Software DSMs: In Proceedings of the International Conference on Supercomputing, July 1998.
[7] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd ISCA, May 1995.
[8] Y. Zhou, L. Iftode, and K. Li. Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems. In Proceedings of USENIX OSDI, October 1996.
[9] P. Keleher. The Relative Importance of Concurrent Writers and Weak Consistency Models. In Proceedings of the 16th ICDCS, May 1996.
[10] C. Amza, A. L. Cox, K. Rajamani, and W. Zwaenepoel. Tradeoffs Between False Sharing and Aggregation in Software Distributed Shared Memory. In Proceedings of the 6th PPoPP, June 1997.
[11] M. Karlsson and P. Stenstrom. Effectiveness of Dynamic Prefetching in Multiple-Writer Distributed Virtual Shared Memory Systems. Journal of Parallel and Distributed Computing, 43(7), July 1997.
[12] T. Mowry, C. Q. C Chan, and A. K. W. Lo. Comparative Evaluation of Latency Tolerance Techniques for Software Distributed Shared Memory. In Proceedings of the 4th HPCA, February 1998.

이 상 권
1996년 부산대학교 전자계산학과 학사. 1998년 한국과학기술원 전산학과 석사. 1998년 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 Software Distributed Shared Memory, Parallel Processing, Computer Architecture, Operating System

윤 희 철
1999년 한국과학기술원 전자전산학과 전산학전공 학사. 2001년 한국과학기술원 전자전산학과 전산학전공 석사. 2001년 ~ 현재 한국전자통신연구원 컴퓨터 소프트웨어 기술연구소 연구원. 관심분야는 Software Distributed Shared Memory, Parallel Processing, Computer Architecture, Operating System

이 준 원
1983년 서울대학교 계산통계학과 졸업 (학사). 1990년 Georgia Tech. 전산학과 (석사). 1991년 Georgia Tech. 전산학과 (박사). 1983년 ~ 1986년 (주)유공 근무. 1991년 ~ 1992년 IBM 근무. 1992년 ~ 현재 한국과학기술원 전산학과 부교수. 관심분야는 운영체제, 병렬처리 등임.

맹 승 렬
1977년 서울대학교 공과대학 전자공학과 졸업. 1979년 한국과학기술원 전산학과 석사학위 취득. 1984년 한국과학기술원 전산학과 박사학위 취득. 1984년 ~ 현재 한국과학기술원 전산학과 정교수. 1988년 ~ 1989년 펜실바니아대학 교환교수. 1994년 ~ 1995년 텍사스대학 교환교수. 관심분야는 *parallel computer architecture, dataflow machines,* vision architecture, multimedia임