Journal of Digital Contents Society Vol. 9 No. 1 March. 2008(pp. 77-86)

분산 공유 메모리 시스템에서 거짓 공유를 줄이는 객체-크기 및 호출지-추적 기반 공유 메모리 할당 기법

이종우*, 박영호**, 윤용익***

요 약

거짓 공유는 공유 메모리 다중 처리기 시스템에서 여러 처리기들이 일관성 유지의 단위 메모리 영역을 공유함으로 인해 발생하는 현상으로써, 메모리 일관성 유지의 정확성에는 아무런 도움을 주지 못하면서 그 비용만 증가시키는 주요 요인이다. 특히 메모리 일관성 유지의 단위가 커질수록 그 피해가 더 커진다고 할 수 있다. 페이지-기반 분산 공유 메모리 시스템에서 거짓 공유를 줄이기 위해서는 공유 페이지에 할당되는 객체들의 특성을 미리 예측하여 참조 패턴이 상이한 객체들이 하나의 공유 페이지에 섞이는 것을 방지하는 것이 필수적이다. 본 논문에서는 객체-크기와 호출지-추적에 기반한 거짓 공유 감소 기법인 SCSTallocator: Sized and Call-Site Tracing based allocator)을 제시한다. SCSTallocator는 서로 다른 코드 위치에서 할당 요청된 공유 객체들은 각각 상이한 참조 패턴을 보일 것이라는 가정에 기반함과 동시에 요청된 객체의 크기가 다르면 향후 참조 패턴도 다를 것이라고 가정하고 있다. 본 논문에서는 기존의 두 정책(크기별 할당 정책과 호출지-추적 기반 할당 정책)을 동시에 적용할 경우 거짓 공유를 더 많이 줄일 수 있을 것이라는 예상을 실험을 통해 확인하였다.

Object-Size and Call-Site Tracing based Shared Memory Allocator for False Sharing Reduction in DSM Systems

Jongwoo Lee*, Youngho Park**, Yongik Yoon***

Abstract

False sharing is a result of co-location of unrelated data in the same unit of memory coherency, and is one source of unnecessary overhead being of no help to keep the memory coherency in multiprocessor systems. Moreover, the damage caused by false sharing becomes large in proportion to the granularity of memory coherency. To reduce false sharing in page-based DSM systems, it is necessary to allocate unrelated data objects that have different access patterns into the separate shared pages. In this paper we propose sized and call-site tracing-based shared memory allocator, shortly SCSTallocator. SCSTallocator places each data object requested from the different call-sites into the separate shared pages, and at the same time places each data object that has different size into different shared pages. Consequently data objects that have the different call-site and different object size prohibited from being allocated to the same shared page. Our observations show that our SCSTallocator outperforms the existing dynamic shared memory allocators. By combining the two existing allocation technique, we can reduce a considerable amount of false sharing misses.

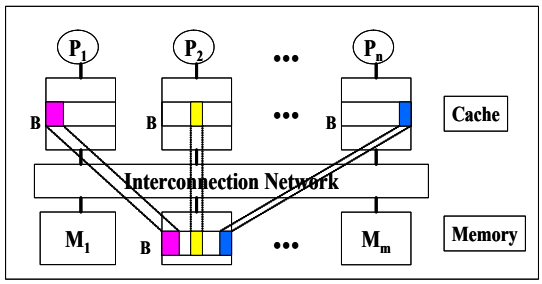
Keywords : False Sharing, Distributed Shared Memory, Dynamic Memory Allocation, Call Site Tracing

1. 서론

분산 공유 메모리(distributed shared memory, 이하 DSM) 시스템에서는 메모리 모듈들이 여러 노드들에 분산되어 존재하기 때문에 데이터의 효과적 캐싱(caching)이 전체 시스템의 성능에 큰 영향을 미친다. 지역(local) 메모리 참조 보다 원격(remote) 메모리 참조에 훨씬 많은 비용이

※ 제일저자(First Author) : 이종우
접수일자:2007년12월18일, 심사완료:2007년12월28일
* 숙명여대 멀티미디어과학전공 조교수
bigrain@sm.ac.kr
** 숙명여대 멀티미디어과학전공 조교수
*** 숙명여대 멀티미디어과학전공 교수
▣ 본 연구는 숙명여자대학교 2007학년도 교내연구비 지원에 의해 수행되었음.

들기 때문에 효과적인 캐싱을 통해 원격 메모리 참조 횟수를 줄일 수 있다면 결과적으로 메모리 참조의 평균 비용을 줄일 수 있게 되므로 시스템 전체의 성능을 향상시킬 수 있다. DSM 시스템에서는 효과적 캐싱을 위해 주로 데이터 복사(replication)나 이주(migration) 기법을 이용하는데, 이는 각 처리기들이 참조하는 데이터 중 자주 참조하는 것들을 자신의 지역 메모리에 위치시켜 메모리 참조 비용을 줄이고자 하는 것이 그 목적이라고 할 수 있다. 그러나 데이터 복사/이주 기법은 메모리 일관성 유지비용을 증가시키는 요인으로 작용하기도 하는데, 특히 데이터 복사 기법의 경우에는 동일 데이터의 여러 복사본들이 각 처리기의 지역 메모리에 상존하게 되므로 일관성 유지비용을 크게 증가시킨다.(그림 1).



(그림 1) DSM 시스템에서 메모리 복사의 예

거짓 공유(false sharing)는 공유 메모리 다중처리기 시스템에서 여러 처리기들이 일관성 유지의 단위 메모리 영역을 공유함으로 인해 발생하는 현상으로써, 메모리 일관성 유지의 정확성에는 아무런 도움을 주지 못하면서 그 비용만 증가시키는 주요 요인이다. 특히 PC-NOW DSM 시스템처럼 메모리 일관성 유지의 단위가 큰(일반적으로, 하나의 가상 페이지) 경우에는 그 피해가 더 커진다고 할 수 있다. [3][4][5][6]에 의하면 페이지 기반 분산 공유 메모리 시스템에서 발생하는 공유 메모리 폴트 중 거짓 공유 폴트가 차지하는 비중이 병렬 응용별로 차이는 있지만 대게 80% 안팎임을 알 수 있는데, 이는 거짓 공유가 분산 공유 메모리 시스템에서 메모리 성능을 떨어뜨리는 주 요인이라는 것을 의미한다. 본 논문에서는 동적 공유 메모리 할당자(dynamic shared memory allocator)를 통해 공유 데이터 영역을 생성하는 병렬 응용들을 대상으로 DSM 시스템에서 거짓 공유 감소에 도움을 주기 위한 동적 공유 메모리 할당 기법을 제시한다.

거짓 공유를 줄이는 동적 공유 메모리 할당을 위해서는 “향후 데이터 객체에 가해질 참조 패턴을 미리 예측”하는 것이 필수적인데, 기존 연구들에서는 이를 위한 다양한 기법들을 제시하였다. 본 연구에서는 이들 기존 연구에서 제시된 기법을 합쳐 동시에 적용할 경우 더 큰 성능 향상을 이룰 수 있을 것이라는 가정 하에, [5][6][8]에서 제시되었던 객체 크기별 할당 기법과 [9]에서 제안한 호출지-추적 기반 할당 기법을 동시에 사용하는 “객체-크기 및 호출지-추적 기반 공유 메모리 할당 기법”을 제시한다.

본 논문의 구성은 다음과 같다. 2절에서는 기존 연구들을 살펴보고, 3절에서는 객체-크기별 할당 기법과 호출지-추적 기반 할당 기법을 합친 SCSTallocator를 설계하고 구현한 내용을 제시한다. 4절에서는 성능 평가 결과를 보이며, 끝으로 5절에서는 결론과 향후 연구 과제를 제시한다.

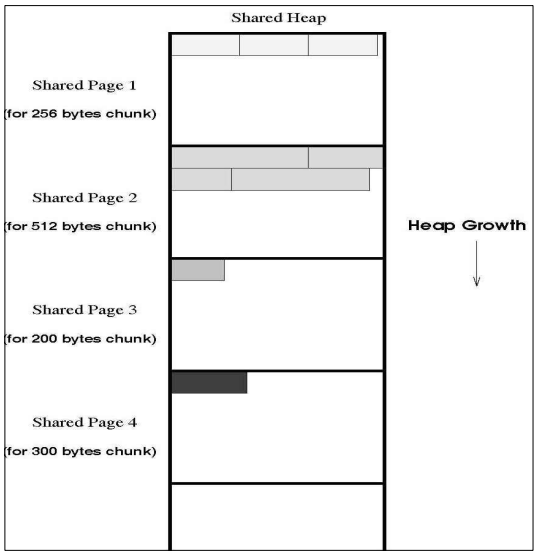
2. 기존 연구

본 논문에서 기본 환경으로 설정하고 있는 DSM 시스템은 시스템 가상 페이지 단위로 메모리 일관성이 유지되는 페이지 기반 DSM 시스템인데, 이러한 시스템을 위한 동적 공유 메모리 할당자가 우선적으로 고려해야 할 점은 요청된 데이터 객체를 어느 공유 페이지에 배치시키느냐 하는 것이다. 만약 요청된 공간에 놓이게 될 데이터 객체의 특성 및 참조 패턴 등을 동적 공유 메모리 할당자가 미리 알 수 있다면 할당자는 앞서 언급한 거짓 공유의 원인을 최대한 없애는 방식으로 데이터 객체들을 공유 페이지에 배치하면 될 것이다. 예를 들어, 참조 패턴이 현저히 다른 데이터 객체들을 같은 공유 페이지에 배치하지 않거나, 또는 서로 관련이 없는 데이터 객체들이 같은 공유 페이지에 섞이지 않도록 하는 등의 기법을 이용하면 거짓 공유의 정도를 상당 부분 완화할 수 있을 것이다. 그러나 동적 공유 메모리 할당자가 요청된 공간에 놓일 데이

터 객체의 특성 및 참조 패턴 등을 미리 알 수는 없으므로 [7]에서는 사용자가 제공한 힌트에 기반한 typed allocation 기법을 제안하기도 하였다. 이 방식에서는 프로그래머가 공유 메모리 요청 시 그 공간의 타입을 지정하도록 하였다. Read-Only 타입과 Write-Mostly 타입, 그리고 Lock 타입 등과 같이 할당받을 공간의 예상되는 참조 패턴을 사용자가 지정하도록 함으로써 서로 다른 타입의 데이터 객체들이 같은 공유 페이지에 섞이지 않도록 하였다. 그러나 이 방법에서는 동적 공유 메모리 할당자의 사용자 인터페이스가 달라지므로 기존 응용들도 소스 코드 수준에서 수정되어야 한다는 추가 비용이 발생한다. 본 연구에서는 동적 공유 메모리 할당자의 인터페이스는 변경시키지 않는다는 것을 가정하고 있기 때문에 이 연구는 본 연구와는 차이가 있다 하겠다. 게다가 아무리 숙달된 프로그래머라 할지라도 프로그램에서 요청하는 공유 공간에 대한 여러 처리기들의 참조 패턴을 미리 알기가 쉽지 않다는 단점도 존재한다.

프로세스별 할당 방식은 서로 다른 프로세스에 의해 요청된 데이터 객체들이 같은 캐쉬 라인에 섞이지 않도록 할당하는 방식이다. 이 기법에서는 서로 다른 프로세스가 요청한 데이터 객체들을 별도의 캐쉬 라인에 배치함으로써 관련 없는 데이터 객체나 참조 패턴이 다른 데이터 객체들이 같은 캐쉬 라인에 섞일 가능성을 줄일 수 있다고 가정하고 있으며, 실험을 통해 효과가 있음을 보이고 있다. 이 기법은 공유 메모리 할당을 여러 처리기가 골고루 요청하는 경우에는 효과가 있지만, 공유 메모리 할당을 어떤 한 프로세스가 전담하는 경우에는 당연히 아무런 효과가 없다. 본 연구의 실험에 사용된 병렬 응용 프로그램들에서도 역시 한 프로세스가 공유 메모리 할당을 전담하고 있기 때문에 이 기법 또한 본 연구와의 비교 대상으로는 적절치 않다고 할 수 있다.

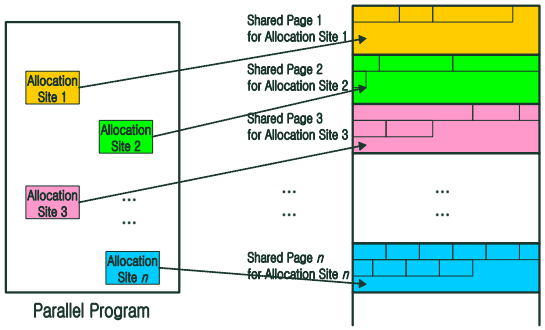
다음은 객체-크기별 할당 방식을 들 수 있다. 객체-크기별 할당 방식이란 하나의 공유 페이지 내에 크기가 다른 데이터 객체가 섞이지 않도록 할당하는 방식을 의미한다. 즉, 한 공유 페이지 내에는 같은 크기의 데이터 객체들만 할당되게 함으로써 이질적인 데이터 객체들이 하나의 공유 페이지에 섞이는 것을 최대한 방지하고자 한 것이다(그림 2).



(그림 2) 객체-크기별 할당 기법에서 객체 할당 예

객체의 특성이나 향후 참조 패턴을 예측하는 근거로 객체-크기 정보를 이용함으로써 할당자 인터페이스의 사용자 투명성(transparency)을 유지함과 동시에 거짓 공유 폴트도 줄일 수 있다는 것이 이 연구들의 결론이라 하겠다. 하지만 객체의 크기가 향후의 참조 패턴을 완전하게 대변하는 것은 아니므로 이를 보완할 수 있는 추가적인 예측 기법이 필요하다 할 것이다.

[9]에서는 CSTallocator라고 하는 호출지-추적 기반 공유 메모리 할당 기법을 제안하였다. 이 기법에서는 공유 객체가 코드 상의 어느 위치에서 할당 요청되었는가를 기준으로 객체의 미래 참조 패턴을 예측한다. 즉, 공유 객체 할당 함수가 호출된 프로그램 상에서의 위치(instruction pointer)를 호출지(Call-Site)로 정의하고, 이 호출지가 각 객체의 향후 참조 패턴을 효과적으로 예측할 수 있다고 제안하고 있다. 호출지가 서로 다른 객체들이 같은 공유 페이지에 섞이지 않도록 하고 있는데(그림 3), 이는 호출지가 다를 경우 향후 참조 패턴도 다를 가능성이 크다는 것을 가정으로 하고 있다. 객체-크기보다 호출지가 각 객체의 특성을 좀 더 의미적으로 표현할 수

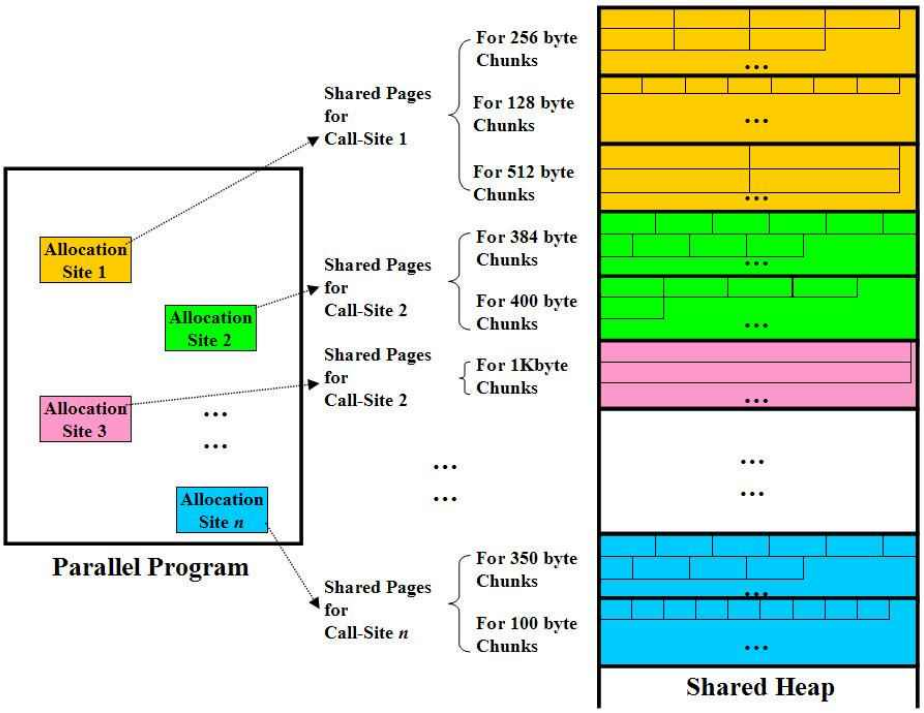


(그림 3) CSTallocator에서 호출지 별 공유 객체 배치 방법

있을 뿐만 아니라 API 투명성도 유지할 수 있고 추가적인 프로그래머 부담도 없다고 밝히고 있다. 병렬 프로그래밍 시 프로그래머들은 대개 각 객체의 용도 별로 서로 다른 위치에서 할당 요청을 하기 때문에 호출지 같은 객체의 의미 정보를 활용하면 객체-크기 같은 단순 정보보다 훨씬 더 거짓 공유를 줄일 수 있다는 것이 [9]의 핵심 아이디어이다.

이와 같은 기존 연구들에서 공통적으로 발견할 수 있는 사항은 공유 객체에 가해질 미래의 참조 패턴을 얼마나 효과적으로 예측할 수 있느냐가 거짓 공유 감소에 큰 영향을 미친다는 점이다. 서로 상이한 참조 패턴을 보일 것으로 예상되는 공유 객체들을 가능한 한 서로 다른 메모리 일관성 유지 단위 블록에 배치해야 하는 것이다. 기존 연구들에서 알 수 있는 또 다른 점은, 객체-크기 같은 단순 정보나 호출지 같은 의미 정보를 동시에 사용하는 할당 기법을 사용한다면 좀 더 많은 거짓 공유를 줄일 수 있을 것으로 기대된다는 점이다. 본 논문에서는 이 같은 점에 착안하여 객체-크기별 할당 기법과 호출지-추적 기반 할당 기법을 동시에 사용하는 SCSTallocator(Sized and Call-Site Tracing based allocator)를 제안하게 된 것이다.

3. 객체-크기와 호출지-추적을 동시에 사용하는 CSTallocator



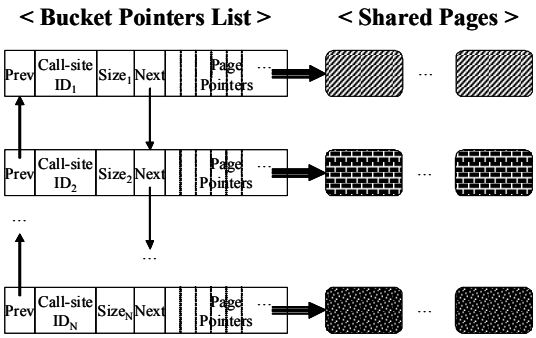
(그림 4) 객체-크기별 할당 방식과 호출지-추적 기반 할당 방식을 모두 사용하는 SCSTallocator에 의한 공유 객체 할당 예

객체-크기와 호출지는 모두 동적 공유 메모리 할당 함수에서 투명하게 얻을 수 있는 정보이지만 그 특성은 서로 다르다. 객체-크기는 할당 함수의 인자를 통해 주어지기 때문에 얻기 쉽다는 장점이 있는 반면 객체의 향후 참조 패턴 예측 정확도는 떨어지는 편이다. 향후 참조 패턴이 판이하게 다른데도 그 객체-크기가 서로 같은 경우가 종종 발생하기 때문이다. 호출지의 경우는 객체-크기에 비해 스택 역추적을 거쳐야 하므로 호출지를 알아내는 과정이 다소 복잡하다는 단점이 있는 반면, 호출지는 객체-크기와는 달리 객체의 사용 의도 정보를 좀 더 많이 내포하고 있기 때문에 향후 참조 패턴 예측 정확도가 높다고 할 수 있다. 물론 호출지만 사용하면 어떤 서로 다른 두 공유 객체의 호출지가 같더라도 그 크기가 다른 경우는 구분하지 못하므로 이는 호출지-추적 기반 할당 방식의 단점이라고 할 수 있다.

본 연구에서는 기존 두 기법의 이 같은 단점을 해소하기 위해 이 두 기법을 모두 사용하였다. (그림 4)는 SCSTallocator에 의해 공유 객체들이 할당된 예를 보이고 있다. (그림 4)에서 볼 수 있듯이 SCSTallocator에서는 요청된 객체의 크기와 요청 함수 호출지를 모두 비교하여 어느 하나라도 다르면 같은 공유 페이지에 섞지 않는다. 결국 한 공유 페이지 내에는 객체-크기와 호출지가 모두 같은 객체들 만이 할당된다. 이렇게 하면 향후에 상이한 메모리 참조 패턴을 보일지모를 객체들이 같은 공유 페이지 상에 섞이게 되는 상황을 최대한 방지할 수 있지 않겠느냐는 것이 SCSTallocator가 기대는 바라고 할 수 있다. 객체-크기별 할당 방식에서는 비록 서로 다른 호출지에서 요청되었다 하더라도 객체-크기만 같으면 같은 공유 페이지에 할당되었었는데 여기서는 이 같은 상황은 발생하지 않는다. 마찬가지로 호출지-추적 기반 할당 방식에서는 비록 객체-크기는 다르더라도 같은 호출지에서 요청되기만 한다면 같은 공유 페이지에 섞였었는데 SCSTallocator에서는 이 같은 상황도 발생하지 않을 것이다.

SCSTallocator도 물론 다음과 같은 상황이 닥치면 그 효용성이 떨어질 수는 있다. 즉, 서로 다른 여러 객체에 대해 객체-크기도 모두 같고 그 호출지도 모두 같아서 같은 공유 페이지에 할당했는데 향후 메모리 참조 패턴을 보니 예상과는 전혀 다르게 참조되는 상황이 발생했을 경우이다. 하지만 병렬 프로그래밍에 조금이라도 경험이 있는 프로그래머라면 이 같은 상황은 거의 발생하지 않는다는 것을 쉽게 알 수 있다. 왜냐하면 병렬 프로그램에 필요한 어떤 특정 공유 객체에 대해 객체-크기를 달리하여 요청하는 경우도 그리 많지는 않은데 호출지 마저 달리하여 할당 요청을 하는 경우는 거의 없기 때문이다. 극단적인 예로, A 구조체를 담기 위해 할당했던 영역을 모두 사용한 후에 공유 메모리를 절약할 목적으로 B 구조체와 C 구조체를 섞어 담아 놓고 재사용하는 경우를 들 수 있다. 하지만 정상적인 프로그래머라면 A 구조체 share_malloc() 후 share_free()로 반환한 후 B 구조체와 C 구조체를 위해 다시 share_malloc()을 호출하기 때문에 이 같은 극단적인 상황은 논외로 해도 무방할 것으로 판단된다.

(그림 5)는 SCSTallocator 구현에서 공유 페이지 구분 관리를 위해 사용한 버켓 양방향 연결 리스트의 구현 모습을 보이고 있다.



(그림 5) SCSTallocator에서 공유 페이지 구분을 위한 버켓 양방향 연결 리스트 구현 모습

4. 성능 평가

이번 절에서는 실험 환경에 대해 설명하고, 3절에서 제시한 SCSTallocator의 성능이 객체-크기별 할당 기법만 사용했을 때와 호출지-추적 기반 할당 기법만 사용했을 때에 비해 얼마나 좋아지는가를 보인다.

<표 1> SCSTallocator, 객체-크기별 할당 방식, 그리고 CSTallocator의 성능 비교 결과 (페이지크기는 4KB)

(a) Cholesky

측정 내용 / 할당 방식	버켓 개수	거짓 공유 폴트 발생 횟수	감소율(%)
객체-크기별	10	44,717	
CSTallocator	17	36,599	18.2
SCSTallocator	21	34,687	22.4

(b) Mp3d

측정 내용 / 할당 방식	버켓 개수	거짓 공유 폴트 발생 횟수	감소율(%)
객체-크기별	8	6,147,589	
CSTallocator	5	5,754,143	6.4
SCSTallocator	11	5,312,768	13.6

(c) Barnes

측정 내용 / 할당 방식	버켓 개수	거짓 공유 폴트 발생 횟수	감소율(%)
객체-크기별	27	5,805,705	
CSTallocator	7	5,104,413	12.1
SCSTallocator	29	4,814,970	17.1

(d) Volrend

측정 내용 / 할당 방식	버켓 개수	거짓 공유 폴트 발생 횟수	감소율(%)
객체-크기별	11	953	
CSTallocator	12	883	7.3
SCSTallocator	17	798	16.3

4.1 실험 환경

본 논문에서는 16개의 노드로 구성된 DSM 시스템을 시뮬레이션하기 위해 실행-구동형(execution-driven) 시뮬레이션 기법을 이용하였다. 사용된 시뮬레이터는 크게 전반부(front-end) 시뮬레이터와 후반부(back-end) 시뮬레이터로 구성되어 있는데, 전반부 시뮬레이터는 주어진 병렬 응용 프로그램의 실행 코드를 해석하여 각 처리기들의 실행을 시뮬레이션 하는 기능을 한다. 우리는 전반부 시뮬레이터로 MINT(Mips INTerpreter)를 이용하였다. 후반부 시뮬레이터는 MINT의 출력을 바탕으로 메모리 관리 시스템의 여러 정책들을 시뮬레이션 한다. MINT는 주어진 실행 코드를 해석하여 데이터 참조를 위한 모든 메모리 참조 시마다 후반부 시뮬레이터에서 제공한 일정 함수들을 호출하기 때문에 시뮬레이션 하고자 하는 메모리 관리 정책이나 메모리 일관성 프로토콜은 후반부 시뮬레이터에 의해 구현된다.

본 실험에서 사용된 병렬 응용 프로그램들은 Cholesky와 Mp3d, Barnes, 그리고 Volrend 인데, 이들은 스탠포드 대학의 SPLASH 사이트와 SPLASH II 사이트에서 얻을 수 있는 병렬 벤치마크 프로그램들에서 임의적으로 선택한 것이다. 실험은 기존의 객체-크기별 할당 방식을 사용했을 때와 CSTallocator를 사용했을 때 발생하는 거짓 공유 폴트 횟수, 그리고 본 논문에서 제시한 SCSTallocator를 사용했을 때 발생하는 거짓 공유 폴트의 횟수를 비교하는 방식으로 실시하였다.

4.2 실험 결과

실험에 사용된 4 개의 병렬 응용프로그램 각각에 대해 본 논문에서 제시한 SCSTallocator를 사용했을 때와 기존 객체-크기별 할당 방식과 CSTallocator를 사용했을 때 발생하는 거짓 공유 폴트 횟수 측정 결과는 <표 1>과 같다. 이 표의 두 번째 컬럼에 있는 “버켓 개수”는 공유 메모리 할당 함수가 반복적으로 호출되는 과정

에서 발견한 고유한 할당 슬롯의 개수를 의미하는데, 객체-크기별 할당 방식에서는 프로그램에서 요청한 객체-크기가 몇 가지 종류인지를 나타내고, CSTallocator에서는 할당자가 추적해낸 호출지 ID의 개수를 의미한다. SCSTallocator에서의 버켓 개수는 (호출지 ID, 객체-크기) 쌍의 개수를 의미한다. 객체-크기별 할당 기법에서의 버켓 개수와 CSTallocator에서의 버켓 개수를 합한 것이 SCSTallocator의 버켓 개수와 일치하지 않는 것은 객체-크기는 달라도 같은 호출지에서 요청된 객체나 같은 호출지에서 요청되었더라도 그 크기가 다른 객체들이 존재한다는 의미이다. 구현된 공유 메모리 할당자에서는 그림 5에서 보았듯이 버켓 구분자 별로 별도의 포인터를 사용해 공유 페이지를 연결 리스트로 관리하게 된다. 즉, 같은 버켓 포인터에 연결된 공유 페이지에는 호출지 ID가 같거나(CSTallocator) 요청 크기가 같거나(객체-크기별 할당 기법), 또는 (호출지 ID, 객체-크기) 쌍이 같은 공유 객체들만이 할당된다. 따라서 버켓의 개수가 많다는 것은 그 만큼 세밀하게 객체의 특성을 구분했다는 의미가 된다. 대개의 경우 구분 기준이 합리적이기만 하다면 버켓의 개수가 많을수록 거짓 공유 감소 가능성은 커질 것이다. 하지만 응용에 따라서는 버켓의 개수가 다른 방식보다 많다 할지라도 거짓 공유 폴트의 횟수가 줄지 않을 수도 있는데, 이는 할당자가 사용한 버켓 구분 알고리즘이 그 응용에 대해서는 별다른 효과를 보지 못했음을 의미한다.

<표 1>에 나타난 실험 결과는 실험에 사용된 4개의 병렬 응용프로그램 모두에 대해 기존 객체-크기별 할당 방식이나 CSTallocator에 비해 본 논문에서 제시한 SCSTallocator가 거짓 공유 감소에 훨씬 더 효과적이라는 것을 보여주고 있다. 이는 객체의 미래 메모리 참조 패턴 예측에 있어서 (객체-크기, 호출지) 쌍이 객체-크기나 호출지가 단독으로 사용된 경우보다 좀 더 정확했다는 것을 의미한다. 또한, 예상대로 버켓의 개수가 많을수록 좀 더 정확한 예측을 할 수 있다는 사실을 보여주고 있다.

버켓의 개수가 많아지면 단점이 발생하는데 바로 버켓의 개수가 많아지는 만큼 공유 페이지를 더 사용한다는 점이다. SCSTallocator의 버켓 개수가 기존 두 기법보다 많기 때문에 공간 효율성 측면에서 보면 SCSTallocator는 불리하다고 할 수 있다. 다음 절에서는 SCSTallocator의 공간 효율성 오버헤드를 분석한다.

4.3 공간 효율성 분석

지금까지의 실험 결과는 SCSTallocator가 거짓 공유 폴트를 줄이는데 효과적이라는 것을 보이고 있다. 그러나 SCSTallocator가 유발하는 공간 오버헤드가 어느 정도인지를 살펴보는 것도 필수적일 것이므로 본 절에서는 SCSTallocator와 기존 객체-크기별 할당 기법, 그리고 CSTallocator가 보이는 공간 오버헤드를 분석한다. 여기서 공간 오버헤드란 제시된 기법이 기존 기법에 비해 추가적으로 사용하는 공유 메모리의 양을 의미한다.

먼저 객체-크기나 호출지 ID 같은 버켓을 사용하지 않는 일반 공유 메모리 할당자, 즉 할당 요청 순서대로 공유 페이지에 섞이도록 할당하는 경우의 공간 효율성을 분석하면 다음과 같다. 일반 할당 방식에서 i 번째 할당 요청의 크기를 s_i라고 할 때 할당 요청 스트림 S는 다음과 같이 표현될 수 있다.

$$\begin{aligned} S &= \{ s_1, s_2, \cdots, s_n \} \\ s_i &= \text{requested size of } i-th \text{ allocation} (1 \le i \le n). \\ n &= \text{total \# of requests}. \end{aligned} \tag{1}$$

이와 같은 할당 요청 스트림을 기존 할당 방식으로 처리하기 위해 필요한 페이지 수는

$$\text{\# of pages required} = \left\lceil \frac{\sum_{i=1}^{n} s_i}{\text{pagesize}} \right\rceil \tag{2}$$

이 된다. 반면 SCSTallocator나 CSTallocator, 객체-크기별 할당 방식에서 각 할당 요청 스트림은 요청 순서를 무시할 경우 다음과 같이 표현될 수 있다.

$$\begin{aligned} &S = \{ S_{bucket_1}, S_{bucket_2}, \cdots, S_{bucket_k} \} \\ &S_{bucket_k} = \text{set of allocations with bucket ID } bucket_k. \\ &S_{bucket_1} \cap S_{bucket_2} \cap \cdots \cap S_{bucket_k} = \emptyset \\ &BS = \{ bucket_1, bucket_2, \cdots, bucket_k \} : \text{set of unique bucket IDs} \end{aligned} \tag{3}$$

이와 같은 할당 요청 스트림을 버켓 별로 할당하기 위해 필요한 페이지 수는

$$\#\ of\ pages\ required = \sum_{bucket_k \in BS} \left\lceil \frac{|S_{bucket_k}| \times AvgSize_{bucket_k}}{\text{pagesize}} \right\rceil \quad (4)$$

$$Here,\ AvgSize_{bucket_k} = Average\ size\ of\ each\ allocation\ request\ heading\ \text{for}\ bucket_k$$

이 된다.

식 (2)와 (4)를 비교해 보면 그 차이가 단지 올림 함수(ceiling)의 횟수에 의존한다는 것을 알 수 있다. 즉, 식 (2)에서는 올림 함수가 한번만 적용되지만 식 (4)에서는 집합 BS의 크기($|BS|$)에 해당하는 횟수만큼 적용된다는 것을 알 수 있다. 이는 일반 할당 방식에 비해 추가로 요구되는 페이지의 수가 객체-크기별 할당 방식에서는 객체-크기 종류의 수 이하이고, CSTallocator에서는 호출지 ID 개수 이하, SCSTallocator에서는 (객체-크기, 호출지) 쌍의 개수 이하라는 것을 의미한다. 따라서 다음 식이 성립한다.

$$Space\ Overhead = \left(\sum_{bucket_k \in BS} \left\lceil \frac{|S_{bucket_k}| \times AvgSize_{bucket_k}}{\text{pagesize}} \right\rceil \right) - \left\lceil \frac{\sum_{i=1}^{n} s_i}{\text{pagesize}} \right\rceil \le |BS| \quad (5)$$

식 (5)에서 알 수 있는 중요한 사실은 사용한 기법이 무엇이건 간에 최대로 구분 기준에 의한 버켓 개수만큼의 공유 페이지가 공간 오버헤드로 작용하게 된다는 사실이다. 식 (5)를 근거로 SCSTallocator와 CSTallocator, 객체-크기별 할당 방식의 공간 효율성을 비교해보면 <표 2>와 같다. <표 2>에서 볼 수 있듯이 CSTallocator의 공간 효율성은 Mp3d와 Barnes에서는 객체-크기별 할당 방식보다 더 좋았지만 Cholesky와 Volrend에서는 다소 떨어지는 등 객체-크기별 할당 방식과 큰 차이를 보이고 있지 않음을 알 수 있었다. 또한 예상대로 SCSTallocator의 공간 효율성이 다른 두 기법보다 떨어지긴 하지만 그리 큰 오버헤드라고 할 수는 없다는 것도 알 수 있다. 이는 현재 보통의 컴퓨터 시스템들에서 지원하고 있는 메모리 사양으로 볼 때 그리 큰 편은 아니라고 할 것이다.

<표 2> SCSTallocator, 객체-크기별 할당 방식, CSTallocator의 공간 효율성 비교 결과 (페이지 크기 : 4KB)

병렬 응용프로그램 이름 (괄호 안은 일반 할당 방식에서의 총 페이지 수)	추가로 소요되는 페이지 수 (괄호 안은 공간 오버헤드(%))		
	객체-크기별 할당 방식	CSTallocator	**SCSTallocator**
Cholesky (738)	10 (1.36)	17 (2.30)	21 (2.85)
Mp3d (553)	8 (1.45)	5 (0.90)	11 (1.99)
Barnes (308)	27(8.77)	7 (2.27)	29 (9.42)
Volrend (441)	11 (2.49)	12 (2.72)	17 (3.85)

5. 결론

우리는 본 논문에서 DSM 시스템에서 동적으로 할당된 공유 메모리를 통해 통신하는 병렬 응용들을 대상으로 거짓 공유를 줄이는 효과적인 동적 공유 메모리 할당 기법을 제시하였다. 공유 메모리 할당자의 사용자 인터페이스를 수정하지 않으면서도 기존 객체-크기별 할당 방식이나 호출지-추적 기반 할당 방식이 단독으로 사용되었을 때보다 효과적으로 거짓 공유를 줄이는 SCSTallocator가 그것이다. SCSTallocator는 각 객체의 향후 메모리 참조 패턴을 예측하기 위해 객체-크기와 객체 할당이 요청된 프로그램 상의 위치(호출지)를 동시에 사용한다. 따라서 객체의 크기와 그 호출지가 모두 같은 객체들만이 같은 공유 페이지 상에 할당된다. 우리는 (객체-크기, 호출지) 쌍이 객체-크기나 호출지가 단독으로 사용되었을 때보다 프로그래머의 객체 사용 의도를 좀 더 정확히 예측할 수 있다고 가정하였고, 실행-기반 시뮬레이션을 통해 SCSTallocator가 다른 두 기존 기법들에 비해 거짓 공유 감소 성능이 뛰어남을 확인할 수 있

었다. 아울러 제시된 기법으로 인해 추가적으로 소요되는 메모리 양을 분석한 결과 (객체-크기, 호출지 ID) 쌍의 개수만큼의 페이지 밖에는 더 소요되지 않음을 증명하였다. 결국 기존 객체-크기별 할당 방식이나 CSTallocator에 비해 공간 오버헤드는 그리 크지 않으면서도 거짓 공유는 더 많이 줄일 수 있었다. 본 논문에서 제시한 기법을 통해 거짓공유가 감소하면 DSM 시스템에서 불필요한 메모리 일관성 유지비용이 줄어들 것으로 예상된다.

향후에는 실험 환경을 시뮬레이션이 아닌 실제 DSM 시스템으로 개선함으로써 거짓 공유 폴트의 감소가 병렬 응용들의 응답 시간 감소에 어느 정도 기여하는지를 측정할 예정이다.

참 고 문 헌

[1] Mark Weiser was best-known for his advocacy of "ubiquitous computing" a concept he first proposed in 1988, http://www-sul.stanford.edu/weiser/Ubiq.html

[1] Andrew S. Tanenbaum. Distributed Operating Systems, chapter 6, pages 333-345. PRENTICE HALL, 1995.

[2] 이 종우, 조 유근. NUMA 다중 처리기에서 조정 가능한 지연 카운터를 이용한 페이지 복사 기법. 전자공학회논문지, 33(6):23-33, June 1996.

[3] Josep Torrellas, Monica S. Lam, and John L. Hennessy. Shared Data Placement Optimizations to Reduce Multiprocessor Cache Miss Rates. In Proceedings of the 1990 International Conference on Parallel Processing, volume II(Software), pages 266-270, August 1990.

[4] Susan J. Eggers and Tor E. Jeremiassen. Eliminating False Sharing. In Proceedings of the 1991 International Conference on Parallel Processing, volume I(Architecture), pages 377-381, August 1991.

[5] 이 종우, 조 유근. NUMA 다중 처리기에서 거짓 공유를 줄이는 공유 메모리 할당 기법. 정보과학회논문지, 23(5):487-497, May 1996.

[6] JongWoo Lee and Yookun Cho. An Effective Shared Memory Allocator for Reducing False Sharing in NUMA Multiprocessors. In Proceedings of 1996 IEEE 2nd International Conference on Algorithms & Architectures for Parallel Processing(ICA3PP '96), pages 373-382, June 1996.

[7] Roger L. Adema and Carla Schlatter Ellis. Memory Allocation Constructs to Complement NUMA Memory Management. In Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, December 1991.

[8] 이종우, 김문희, 한장희, 지대구, 윤종완, 김장선. 분산 공유 메모리 시스템에서 동적 공유 메모리 할당 기법이 거짓 공유에 미치는 영향. 정보과학회논문지, 24(12):1257-1269, December 1997.

[9] Jongwoo Lee, Sung-Dong Kim, Jae Won Lee, and Jangmin O. CSTallocator: Call-Site Tracing based Shared Memory Allocator for False Sharing Reduction in Page-based DSM Systems, In Proceedings of the 2nd International Conference on High Performance Computing and Communications 2006 (LNCS 4208), pages 148-159, September 2006.

[10] J. E. Veenstra. MINT Tutorial and User Manual. Technical Report TR452, Computer Science Department, University of Rochester, July 1993.

[11] J. E. Veenstra and R. J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In Proceedings of the Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS '94), pages 201-207, January-February 1994.

[12] J. P. Singh, W. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. ACM SIGARCH Computer Architecture News, 20(1):5-44, March 1992.

[13] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 24-36, June 1995.

이 종 우

1990년 : 서울대학교 컴퓨터공학과 (학사)
1992년 : 서울대학교 컴퓨터공학과 대학원(석사)
1996년 : 서울대학교 컴퓨터공학과 대학원(박사)
1996년～1998년 : 현대전자㈜ 정보시스템사업본부 과장
1998년～1999년 : 현대정보기술㈜ 책임연구원
1999년～2002년 : 한림대학교 정보통신공학부 조교수
2002년～2003년 : 광운대학교 컴퓨터공학부 조교수
2003년～2004년 : 아이닉스소프트㈜ 개발이사
2004년～현 재 : 숙명여자대학교 정보과학부 멀티미디어과학전공 조교수
관심분야 : Mobile System Software, Storage Systems, Computational Finance, Cluster Computing, Parallel and Distributed Operating Systems, Embedded System Software

박 영 호

1986년～1992년 : 동국대학교공과대학 컴퓨터공학과(학사, 석사)
1999년～2005년 : 한국과학기술원 전산학과(공학박사)
1993년～1999년 : 한국전자통신연구원(ETRI) 교환전송연구단 선임연구원
2005년～2006년 : 한국과학기술원 첨단정보기술연구센터 연구원
2006년～현 재 : 숙명여자대학교 이과대학 멀티미디어과학과 조교수
관심분야 : 데이터베이스관리시스템, 정보검색, XML, Telecommunication System

윤 용 익

1985년 : 한국과학기술원 전산학과 석사(전산학)
1994년 : 한국과학기술원 전산학과 박사(전산학)
1985년～1997년 : 한국전자통신연구원 책임연구원
1997년～현 재 : 숙명여자대학교 정보과학부 교수
2004년～2005년 : 미국 University of Colorado Visiting Professor
관심분야 : 미들웨어, 멀티미디어 시스템, 모바일 시스템, 콘텐츠 전달 시스템, 임베디드 시스템, 실시간 시스템