버스기반의 공유메모리 시스템에서 사용된 비트맵 테이블의 크기 축소와 성능 분석

우 종 정† · 이 가 영††

요 약

버스기반 공유메모리-다중프로세서는 공유버스의 사용으로 인한 병목 현상이 시스템의 성능을 제한하며, 특히 분리형 트랜잭션 환경 하에서 각 프로세서들로부터 생성되는 메모리 접근에 대한 요청의 일부가 불필요하게 메모리 입력 버퍼에 대기함으로써 시스템의 성능을 저하시킨다. 이와 같은 불필요한 메모리 입력버퍼에서의 대기는 각 블록에 대한 상태정보를 이용함으로써 제거될 수 있지만, 메모리의 각 블록에 대하여 상태정보가 완전 사상된 비트맵테이블을 저장하기 위한 SRAM에 대한 부담이 발생되었다. 본 연구에서는 이와 같은 문제점을 해결하기 위하여 비공유부분의 배제와 참조 국부성의 원리를 적용하여 상태정보를 저장하기 위한 SRAM의 용량을 줄이기를 제안한다. 시뮬레이션 결과에 의하면 시스템의 성능에는 거의 영향을 미치지 않으면서 상태정보의 저장 용량을 줄일 수 있어 가격-대-성능의 향상을 도모할 수 있다.

Size Reduction and Performance Analysis of the Bit-map Table Used in the Bus-based Shared Memory System

Jongjung Woo† · Ka Young Lee††

ABSTRACT

The bus contention among bus-based shared-memory multiprocessors limits their performance. In addition, under split bus transaction environment, multiprocessors may make some memory requests unnecessary stand by in the memory access buffer, which makes system performance worse. This unnecessary stand-by can be eliminated by maintaining the bitmap table which contains the status bit for each memory block. However, this mechanism requires a great size of SRAM for the status information, which is fully mapped from the whole memory blocks. To solve this problem, we propose a bitmap cache which exploits partial mapping and locality of references. The simulation results show that the proposed system can greatly reduce the capacity of SRAM for the status information with little deteriorating its performance.

1. 서 론

컴퓨터의 성능 향상에 대한 부단한 노력은 응용분야의 확대와 사용자들의 증가를 초래해왔으며 처리해야 할 작업의 증가로 시스템 규모가 커지고 있다. 기존의 단일 프로세서로는 폭증하는 대규모의 작업을 원하는 시간 내에 효율적으로 처리할 수 없기 때문에 저렴한 고성능 마이크로프로세서로 구성된 다중프로

※이 논문은 1996년도 성신여자대학교 학술연구조성비 지원에 의하여 연구되었음.

† 종신회원:성신여자대학교 전산학과
†† 준 회 원:성신여자대학교 전산학과

논문접수:1997년 8월 25일, 심사완료:1997년 10월 21일

세서 개념이 도입되었다. 특히 버스기반 공유메모리 다중프로세서는 시스템 확장과 구현의 용이성 때문에 널리 사용되고 있지만, 버스와 메모리의 병목현상으로 시스템의 확장과 성능에 걸림돌이 되고 있다[1, 2, 3]. 이를 극복하기 위하여 제안된 로컬캐쉬는 공유데이터에 대한 일관성 유지(cache coherence) 문제가 야기되는데, 이를 위하여 다양한 캐쉬 프로토콜이 연구되어 왔으며 버스기반 다중프로세서 환경에서는 쓰기/무효화 프로토콜이 널리 사용되고 있다[2, 3, 4].

최근 분리형 트랜잭션 버스를 기반으로 공유메모리 다중프로세서의 성능 향상을 위하여 메모리 블록에 대한 상태정보를 고속메모리에 저장함으로써 공유메모리에 대한 불필요한 접근 요구를 미리 제거하여 시스템의 성능 향상을 도모하였다[5]. 그러나 이 방법에서는 메모리의 모든 블록에 대한 상태정보를 완전사상(fully associative mapping)을 하므로 실제로 필요로 하는 공유블록에 대한 상태정보뿐 아니라 불필요한 정보-비공유정보에 대한 상태정보-까지도 저장하는 문제점이 있다. 또한 DRAM의 가격하락과 응용프로그램의 대형화로 인한 메모리 용량의 증가는 이에 대응하는 상태정보의 양을 증가시켜 이를 저장하기 위한 고속메모리에 대한 가격 부담을 가중시키고 있다.

본 연구에서는 상태정보의 저장을 위한 고속메모리의 용량을 줄이기 위하여 불필요한 상태정보의 배제와 상태정보의 국부성을 이용하는 비트맵캐쉬(BC: Bitmap Cache)를 제안하여 시스템의 성능에는 거의 영향을 주지 않으면서 가격-대-성능의 최적화를 도모하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 제시되었던 다중프로세서에 대한 여러 가지 형태와 각각의 문제점을 제시하며, 3장에는 제시된 문제점인 상태정보 저장을 위한 SRAM의 크기를 줄이는 방법을 제안한다. 4장에서는 제안된 방법에 대하여 시뮬레이션을 수행하여 기존 방법들과 성능을 비교 분석한다. 끝으로 5장에서는 결론을 맺는다.

2. 기존 캐쉬 프로토콜과 상태정보의 응용

다음은 캐쉬 일관성 프로토콜, 버스기반 공유메모리-다중프로세서에서 널리 사용되고 있는 쓰기-무효화 프로토콜중의 하나인 Berkeley 프로토콜, 그리고 Berkeley 프로토콜의 상태정보 적용 및 문제점 등에 대하여 기술한다.

2.1 캐쉬 일관성 프로토콜

버스기반 공유메모리-다중프로세서에 적합한 스누핑 캐쉬 프로토콜은 공유 캐쉬 블록을 갱신할 때 모든 다른 캐쉬에게 방송(broadcasting)하며 각 캐쉬 제어기는 공유버스를 스누핑(snooping)하여 적절한 조치를 수행하는 프로토콜로써, 쓰기-무효화 및 쓰기-갱신 프로토콜로 분류된다[1, 2, 3].

쓰기/무효화 캐쉬 프로토콜은 한번에 다수의 프로세서가 읽을 수가 있지만 단지 하나의 프로세서에게만 블록의 갱신을 허용한다. 초기에는 읽기 목적으로 하나의 블록이 여러 곳의 로컬캐쉬에 의하여 공유된다. 그러나 프로세서 P가 자신의 로컬캐쉬에 있는 공유블록 B를 B'로 갱신하기를 원하면 다른 로컬캐쉬에 위치한 공유블록 B와 상이한 내용을 가지게 된다. 여기서 공유블록 B를 B'로 갱신한 프로세서 P를 소유자 프로세서라고 하며, 프로세서 P가 공유블록 B를 갱신할 때 그 블록을 배타적 상태로 두고 다른 캐쉬에 위치한 공유블록 B에 대하여 무효화하는 메시지를 방송한다. 일단 배타적 상태가 된 공유블록 B에 대하여 다른 프로세서가 요구할 때까지 소유자 프로세서는 그 블록을 자유롭게 갱신할 수 있다. 대표적인 것으로 MESI, Synapse, Berkeley 등의 프로토콜이 있으며[3], 임의의 순간에 공유블록의 내용이 동일하지 않을 수 있지만 속도가 빨라 다중 프로세서 시스템에서 상용으로 널리 사용되고 있다.

쓰기/갱신 프로토콜은 쓰기/무효화 프로토콜과는 달리 다수의 프로세서가 읽을 수 있을 뿐만 아니라 다수의 프로세서로 하여금 쓰기를 허용하는 프로토콜로써, 프로세서 P가 공유블록 B에 쓰기를 원할 때 갱신하고자 하는 내용을 모든 다른 캐쉬에게 알게 하여 공유블록 B의 내용을 일치시킨다. Dragon과 Firefly 프로토콜이 여기에 속하며[3], 프로세서로부터 쓰기 동작이 있을 때마다 메모리와 다른 로컬캐쉬에 대해서도 쓰기 동작이 발생하므로 전체 처리시간이 길어지지만 임의의 순간에 공유블록의 내용이 동일하다.

2.2 Berkeley 프로토콜

Berkeley 소재 California 대학에서 설계된 RISC 다

중프로세서에 구현된 프로토콜로써 네 가지 - Invalid, Valid, Dirty, 그리고 Shared-Dirty - 캐쉬 상태를 가지며[3, 6], 캐쉬의 상태 전이를 위하여 별도의 신호선을 필요로 한다. Invalid 상태는 해당된 블록의 내용이 유효하지 않음을 나타내며, Valid 상태는 해당하는 캐쉬블록과 동일한 블록이 다른 프로세서내의 캐쉬 혹은 메모리에 복사본이 있을 수 있으며 해당되는 모든 블록의 내용이 유효함을 의미한다. Dirty 상태는 캐쉬내의 해당 블록의 자료가 갱신되었으며 다른 캐쉬에서나 메모리에 복사본이 없음을 의미하며, 해당 블록의 자료만 유효하다. Shared-Dirty 상태의 블록은 저장된 자료가 유효하고 그 블록이 갱신되었으며 다른 프로세서의 캐쉬에서도 해당 블록을 Valid 상태로 복사본을 가질 수 있지만, 메모리에 저장된 해당 블록은 유효하지 않음을 의미한다. Shared-Dirty 혹은 Dirty 상태에 있는 블록이 교체대상으로 선택되었을 때 메모리에 쓰여진다.

Berkeley 프로토콜이 블록의 요청에 대하여 상태의 전이를 나타내며 다음과 같이 행동한다[3].

1) 읽기적중 : 해당 블록을 읽고 블록의 상태는 변함이 없다.
2) 읽기실패 : 블록이 Dirty 혹은 Shared-Dirty 상태이면 그 블록을 가진 캐쉬가 블록의 내용을 요청한 캐쉬에게 제공하고 Shard-Dirty로 상태 전이해야하며, 그 이외의 경우 메모리가 자료를 제공해야 한다. 요청한 캐쉬내의 해당 블록은 Valid 상태가 된다.
3) 쓰기적중 : 블록이 Dirty 상태이면 원하는 자료를 해당 블록에 단순히 쓰고, Valid 혹은 Shared-Dirty인 경우에는 쓰기 동작을 수행하기 전에 무효화 신호를 보내어 다른 프로세서에 위치한 해당 블록을 무효화시키고 요청한 캐쉬의 해당 블록은 Dirty로 상태를 전이시킨다.
4) 쓰기실패 : 읽기실패와 같이 그 블록을 가진 캐쉬로부터 자료를 제공받고 원하는 단어를 변경시킨 후 Dirty 상태로 전이한다. 모든 다른 캐쉬는 해당 블록을 Invalid 상태로 전이한다.

2.3 상태정보를 이용한 Berkeley 프로토콜

쓰기/무효화 캐쉬 프로토콜중의 하나로 2.2절에서 언급한 Berkeley 프로토콜은 비분리형 트랜잭션 환경에서의 동작을 묘사한 것으로 분리형 트랜잭션 환경에서는 상태정보를 이용함으로써 약간의 성능개선을 도모할 수 있다[5, 7, 8]. 분리형 트랜잭션 구조에서는 요청기가 요청을 한 직후 버스 점유권을 포기한 채 응답기로부터 응답을 기다리기 때문에 요청과 응답에 대한 전송을 병렬적으로 수행할 수 있다. 이와 같은 분리형 트랜잭션 환경에 능동적으로 대처하기 위하여 상태정보를 이용할 수 있는데 이는 새로운 캐쉬 프로토콜이 아니라 어떠한 쓰기/무효화 캐쉬 프로토콜로 구현된 다중프로세서 시스템에서도 적용할 수 있다[5]. 상태정보는 각 메모리 블록에 대하여 그 블록이 다른 프로세서내의 캐쉬에 갱신된 상태로 저장되어 있는지 혹은 각 메모리 블록이 유효한 상태로 있는가에 대한 정보를 의미하며 이를 접근속도가 빠른 SRAM에 저장하여 메모리 블록에 대한 접근요청이 있을 때 메모리를 접근하기 전에 SRAM에 저장되어 있는 해당 블록에 대한 상태정보를 참조한다.

분리형 트랜잭션 환경에서는 공유메모리 모듈이 메모리 접근에 대한 요청을 처리하는 중이라도 새로운 요청을 허용하게 되어 대기중인 요청을 보관할 메모리 입력버퍼를 필요로 한다. 여기서 첫 번째 요청 α에 대응하는 블록이 다른 프로세서의 캐쉬에 갱신된 상태로 저장되어 있을 경우 메모리 요청 α는 필요없이 메모리 입력 버퍼에 대기하게 된다. 따라서 블록 요청 α이후의 또다른 요청 β가 발생할 경우 취소될 요청 α로 인하여 버퍼에 대기중인 요청 β가 α를 위한 스누핑 시간동안 대기하게 되어 전반적인 시스템의 성능저하를 초래한다. 만약 α가 메모리 입력버퍼에서 대기할 필요가 없음을 미리 알려진다면 β가 대기시간없이 곧바로 메모리로 접근할 수 있어 성능의 향상을 꾀할 수 있다[5]. 상태정보를 이용한 방법은 공유메모리의 각 블록이 유효한 지 혹은 그 블록이 다른 프로세서의 로컬 캐쉬에 갱신된 상태로 저장되어 있는 지 여부에 관한 정보를 스누핑하기 전에 미리 파악하기 위하여 공유메모리의 각 블록에 대응하는 상태정보를 이용한다. Berkeley 프로토콜에서 상태정보를 이용하는 경우 상태정보에 따라 다음과 같이 동작한다[5].

Dirty 상태 : 해당 블록이 캐쉬 내에 갱신된 상태로 저장되어 있고 공유메모리 내에서는 더 이상 유효하지 않음을 의미한다. 따라서 해당 블록에 대한 메모리 요청이 전송되었을 때 그 요청을 메모

리 입력 버퍼에 저장하지 않음으로써 불필요하게 메모리 입력버퍼에 대기하지 않도록 한다.

Valid 상태 : 공유메모리에 저장된 해당 블록의 유효함을 의미한다. 기존의 방식과 동일하게 메모리에 대한 요청을 메모리 입력버퍼에 저장함과 동시에 스누핑을 시작한다. 스누핑의 결과에 따라 캐쉬 혹은 메모리로부터 해당블록의 내용을 응답 받는다.

이 방법은 상태정보를 이용하여 성능개선을 이루기 위해서 적절한 고속메모리를 선택해야한다. 상태정보를 저장할 수 있는 메모리는 DRAM과 SRAM이 있는데 이들을 살펴보면[9], DRAM의 경우 저속이며 더구나 연간 약 7%의 속도 향상을 보일 정도여서 주로 공유메모리에서 사용되지만, SRAM의 경우는 고속이어서 주로 캐쉬로 사용되며 더구나 연간 약 40% 정도의 속도향상을 보이고 있어 DRAM과의 속도차이가 더욱 커지며 이러한 속도차이는 향후 5년간 지속되리라 예상된다. 따라서 상태정보를 위하여 SRAM을 사용해야 하는데 SRAM은 DRAM에 비하여 약 4배정도의 높은 가격을 보이고 있어 대규모의 상태정보인 경우 경제적인 부담이 주어진다.

3. 비트맵 캐쉬

상태정보란 2.3절에서 언급한 바와 같이 메모리 내 데이터 블록의 변경여부를 나타내는 것이며 각 프로세서들은 이 정보로서 자신들이 요청한 공유 데이터 블록의 시스템 내 일관성 유지 여부를 파악할 수 있다. 이러한 상태정보를 공유메모리의 모든 블록에 대하여 2.3절에서 언급한 바와 같이 완전사상 형태로 저장하고 있는 SRAM을 비트맵테이블(BT : Bitmap Table)이라 하며, 각 프로세서는 요청한 블록에 대한 소유자의 존재여부를 파악하기 위해 이를 사용하고 있다. BT를 이용하는 시스템에서는 다음과 같은 문제점이 있다.

- 메모리 블록의 완전사상된 상태정보로 인한 SRAM의 낭비. 즉, 비공유 블록의 상태정보까지 포함하게 됨. 비공유블록의 경우 해당 프로세서의 로컬캐쉬 혹은 공유메모리에 원하는 정보가 있으므로 다른 프로세서의 캐쉬를 스누핑할 필요성이 없음.
- DRAM의 가격 하락과 집적도의 증가, 응용 프로그램의 대형화 등으로 인하여 공유메모리 용량의 증가를 초래하며, 이로 인한 상태정보를 위한 SRAM의 사용이 비례적으로 증가하여 이에 따른 시스템 구성비용의 부담 증가.
- 메모리의 국부성 활용이 없음. 즉, 최근에 참조된 블록 혹은 그 블록의 인근에 위치한 블록에 대한 요청 확률이 높듯이 이에 대응하는 상태정보의 요청 확률도 높아 국부성 원리를 이용할 수 있음.

위와 같은 문제점들에 대한 해결책으로 다음의 두 가지를 제시할 수 있다. 상태정보에 대한 디렉토리를 구성할 때 메모리 블록의 완전사상 대신 비공유블럭에 대한 상태 정보를 배제시키는 것과 상태정보에 대한 국부성 원리를 이용하여 최근에 참조된 블록의 상태정보 혹은 인근한 상태정보만을 저장하는 것이다. 즉, 캐쉬 개념을 도입하여 가까운 장래에 참조될 확률이 높은 정보만을 SRAM에 저장함으로써 전체 시스템 구성에 대한 비용을 절감하고 효과를 극대화할 수 있다.

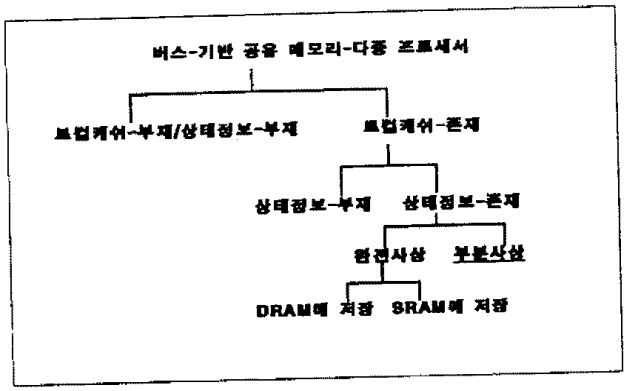
다음은 버스기반 공유메모리-다중프로세서 시스템을 분류하여 제안하고자 하는 다중프로세서 시스템의 위치를 점검하고, 상태정보와 제안하는 시스템의 동작 메커니즘에 대하여 살펴본다.

3.1 버스기반 공유메모리-다중프로세서 시스템의 분류

(그림 3.1)은 버스기반 공유메모리-다중프로세서 시스템을 로컬캐쉬의 존재여부, 메모리 상태정보의 유/무, 그리고 상태정보의 위치 등에 따라 분류한 것이다. 여기서 대부분의 다중프로세서의 경우 로컬캐쉬를 사용하므로 "로컬캐쉬-부재/상태정보-부재"의 경우 의미가 없다. 또한 (그림 3.1)에서 가장 아래쪽에 위치한 "완전사상/DRAM에 저장"의 경우는 로컬캐쉬 및 상태정보가 존재하지만 상태정보를 속도가 느린 DRAM에 저장한 것으로, 비록 Synapse N+1 다중프로세서 시스템에서 유사하게 이용했지만[3, 10], 상태정보를 짧은 시간에 접근해야 유용하므로 분류상의 의미만 있을 뿐이다. 그 이외의 것은 모두 로컬캐쉬가 존재하지만 상태정보의 존재 여부와 메모리 블록과 상태정보의 저장되는 장소와의 사상관계에 따라 로컬캐쉬-존재/상태정보-부재(LC/NS : Local Cache/Non-Status), 로컬캐쉬-존재/상태정보-완전사상(LC/FS : Local Cache/Fully-mapped Status), 및 로컬캐쉬-

존재/상태정보-부분사상(LC/PS: Local Cache/Partially-mapped Status)과 같이 분류된다.

LC/NS 시스템은 Berkeley, 혹은 Illinois 프로토콜 시스템 등 버스기반 공유메모리-다중프로세서 시스템의 대부분이 이 범주에 속하는 시스템이다. LC/FS 시스템의 경우는 2.3절에서 살펴본 시스템이며, 마지막으로 LC/PS 시스템의 경우 본 연구에서 제안하고자 하는 것이다.



(그림 3.1) 버스기반 공유메모리-다중프로세서 시스템의 분류
(Fig 3.1) Classification of bus-based shared memory multiprocessors

3.2 상태정보와 동작 메커니즘

상태정보는 공유메모리의 두 가지 상태-VALID과 MODIFIED-중의 하나를 나타내는 것으로 공유메모리 영역에 대하여 로컬캐쉬의 한 블록에 해당하는 크기마다 하나의 비트씩 유지된다. 따라서 상태정보캐쉬의 크기는 공유메모리의 크기, 블록의 크기, 그리고 공유메모리에서의 프로세서간의 공유블록 비율에 달려 있다. 예를 들어 64 Mbytes의 메모리 영역에 대해 64 Bytes의 블록 크기를 갖는 캐쉬를 사용한다고 가정하자. LC/FS 시스템의 경우 메모리의 모든 블록에 대하여 1 비트씩 필요하므로 128 Kbytes(=1 Mbits)의 SRAM이 상태정보 데이터부분으로 소요되며 태그(tag)부분은 필요로 하지 않는다.

그러나 메모리 내의 블록에 대한 상태정보를 필요하지 않는 부분이 있다. 예를 들어 프로세서에서 필요한 메모리 영역 중에서 코드(code)부분은 비록 공유할지라도 변경되지 않는 부분이므로 상태정보를 유지하면 항상 Valid 상태로 있을 뿐이다. 따라서 이와 같은 코드부분이나 혹은 비공유부분 등은 상태정보와 관련이 없다. 또한, 스택이나 동적 메모리(heap) 등의 경우 프로세서간 공유 가능한 블록이 있지만 컴파일 과정 등의 도움으로 메모리내의 공유블록이 많은 연속성을 가질 수 있다.

공유블록(여기서부터는 코드부분과 비공유블록을 제외한 나머지 부분을 언급함) 비율이 25%이며 공유블록이 연속적일 경우 32 Kbytes(=1/4M bits)의 상태정보가 존재하므로 32 Kbytes의 SRAM이면 완전사상인 LC/FS 시스템의 경우와 동일한 효과를 가진다. 그러나 부분사상을 하는 경우 상태정보를 위한 데이터 부분만 필요한 것이 아니라 상태정보에 대응하는 메모리의 블록주소를 명시하는 태그(tag)부분이 필요하다. 예를 들어, 공유블록의 비율이 25%이며 2:1의 부분사상을 하는 경우 BC의 블록 크기를 메모리 블록의 크기와 동일한 64 Bytes 라고 가정하면 16 Kbytes의 BC는 256개의 블록으로 구성되므로 하나의 블록에 대하여 블록 주소를 위한 20 비트가 필요하므로 640 Bytes(=256 ×20 bits×1 bytes/8 bits) 정도의 태그부분이 필요하다. 비록 640 Bytes 정도의 태그부분이 추가로 필요하지만 완전사상의 경우 128 Kbytes의 SRAM과 비교하면 상당한 비용 절감의 효과를 얻을 수 있다.

상태정보의 국부성 원리를 이용한 LC/PS 시스템의 경우 최근에 사용된 공유블록에 대한 상태정보만을 저장하게 되어 요청된 공유블록의 상태정보가 BC내에 존재하지 않을 경우가 발생한다. 이와 같은 경우 해당 상태정보블록을 메모리로부터 가져와야 하므로 BC내에 있는 하나의 블록을 축출해야한다. 이와 같은 스와핑(swapping) 현상은 메모리내의 공유블록의 연속성 여부에 따라 크게 달라진다. 즉, 공유블록들이 컴파일러 등의 도움으로 강한 연속성을 유지하면 BC가 BT와 유사한 성능을 얻을 수 있다.

BC를 이용할 때 프로세서의 캐쉬의 읽기/쓰기, 적중/실패 등에 따른 기본적인 동작을 살펴보면 다음과 같다.

1) 읽기적중: 상태정보와 관련 없이 지역적으로 읽기 동작을 진행한다.
2) 읽기실패: 읽기 실패가 발생한 블록에 대한 메모리에서의 상태정보를 알아보기 위해 BC를 조사한다. 해당 블록의 상태정보가 Modified 상태이면 블록의 소유자를 찾아 캐쉬-대-캐쉬 데이터 전송을 수행하며, Valid 상태이면 메모리 입력버

퍼에 해당 요청을 대기시키고 LC/NS 시스템의 과정과 동일하게 스누핑 결과에 따라 프로세서 캐쉬 혹은 메모리로부터 원하는 블록의 내용을 읽는다. 만약 BC내에 해당 블록에 대한 상태정보비트가 존재하지 않을 경우 스와핑 동작으로 해당 상태정보를 포함하는 상태정보 블록을 메모리로부터 BC로 가져온다. 이와 같은 스와핑 과정중에 스누핑 동작을 수행하여 원하는 블록이 다른 프로세서의 로컬캐쉬에 존재하면 캐쉬-대-캐쉬 전송을 통하여, 메모리에 존재하면 메모리로부터 요청한 블록을 구한다.

3) 쓰기적중 : 요청 프로세서의 로컬캐쉬에서의 동작 과정은 사용하는 캐쉬 프로토콜에 따라 수행하며 블록의 갱신에 의하여 파생되는 무효화 혹은 갱신 신호등을 통하여 해당 상태정보비트를 Modified 상태로 전이시킨다. 만약 해당 상태정보가 BC내에 존재하지 않을 경우 스와핑 과정이 추가된다.

4) 쓰기실패 : 쓰기적중에서 발생한 현상과 동일하나 무효화 동작만 발생한다.

4. 성능 평가

본 연구에서는 분리형 트랜잭션 환경 하에서 Berkeley 프로토콜을 사용하는 기존의 두 가지 방법-LC/NS 및 LC/FS-으로 구현된 다중프로세서 시스템과 상태정보의 국부성을 이용한 LC/PS 시스템의 여러 비율의 부분사상에 대하여 시스템의 성능변화를 살펴보고자 한다. 즉, 상태정보의 부분적 사상이 상태정보를 유지하지 않는 시스템과 완전사상한 시스템에 비하여 시스템의 성능에 어느 정도의 영향을 미치는 지 살펴본다.

다중프로세서 시스템의 성능을 평가하기 위하여 합성 모델을 근거한 시뮬레이션을 수행하였는데, 이는 트레이스 구동 시뮬레이션은 다중프로세서의 구조적 특성에 많은 영향을 받기 때문이다. 공유블록의 참조를 위해서 국부성을 잘 반영할 수 있는 LRU (Least Recently Used) 스택 모델[3, 11]을 사용하였으며, 비공유 블록에 대해서는 고정된 확률 분포를 사용한 독립 참조 모델(Independent Reference Model)[3, 9]을 사용하였으며, 시뮬레이션 언어로는 동시에 여러 프로세서들을 생성시켜 이들간의 동기화 및 메시지 전송 가능하게 하여 다중프로세서 시스템을 모델링하기 쉬운 시뮬레이션 언어인 CSIM [12]을 사용하였다. 실험에 필요한 환경변수들은 기존의 연구에서 사용된 것으로[3, 5], 〈표 4.1〉에 나타내었다. 또한 시뮬레이션에서 사용한 하나의 메모리 모듈은 32 Mbytes로 하였으며 메모리 입력버퍼의 크기는 시스템의 프로세서 개수와 동일하게 주어 모든 프로세서가 하나의 메모리 접근에 대한 요청이 있을 경우 버퍼 내 입력 요청간의 충돌은 발생하지 않게 하였다.

성능 비교를 위하여 기존의 여러 연구에서 사용된 성능 척도인 System Power를 사용하였으며[3, 5], 이는 시스템 내에서 사용된 프로세서의 활용도(Processor Utilization)에 대한 합에 100을 곱한 수치로써 다음과 같이 산출된다.

$$\text{SystemPower} = \sum_{i=0}^{N_{cpu}} (Ep_i / (Ep_i + Ip_i)) \times 100$$

여기서 N_{cpu}는 시스템 내에 존재하는 CPU의 개수, Ep_i는 프로세서 P_i의 수행시간, 그리고 Ip_i는 프로세서 P_i의 유휴시간(idle time)을 의미한다.

〈표 4.1〉 실험에 사용된 환경변수
〈Table 4.1〉 Environment variables used in the simulation

환경 변수	값	환경 변수	값
공유블록의 참조 확률	15-50%	CPU의 수	1-32 개
읽기 요청 확률	70%	메모리 모듈 개수	1-16 개
캐쉬 적중 확률	95%	메모리 모듈당 입력 버퍼 개수	1 개
SC의 블록교체 발생 확률	35%	블록의 크기	64 Bytes
변경블록에서 쓰기적중확률	95%	캐쉬의 크기	256 Kbytes
공유블록의 개수	256-32768 개	CPU 클럭	66 Mhz
버스 클럭	16.5 Mhz	메모리 클럭	33 Mhz

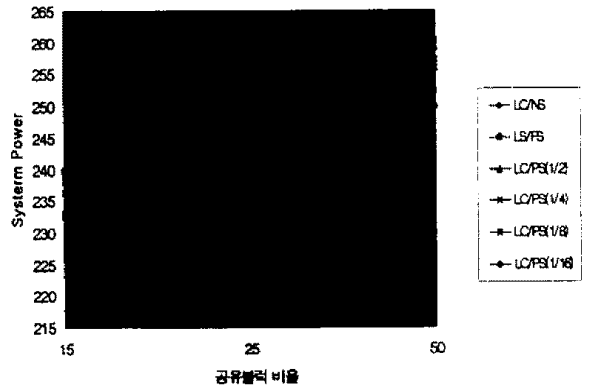
4.1 공유블록에 대한 접근 확률에 따른 변화

공유블록에 대한 접근확률(SR : Shared block access Ratio)이 높다는 것은 각 프로세서들이 공유블록을 사용할 확률이 높다는 것을 의미하며, 따라서 대응하

는 상태정보의 참조율도 높음을 의미하므로 상태정보의 활용에 따른 시스템의 성능 향상을 기대할 수 있다. 본 절에서는 공유블록에 대한 접근율의 변화에 따른 기존의 두 가지 환경-LC/NS 및 LC/FS-과 본 연구에서 제안하는 LC/PS의 네 가지(1/2, 1/4, 1/8, 그리고 1/16) 부분사상에 대한 다중프로세서 시스템의 성능 변화를 살펴보고자 한다. 공유블록에 대한 접근 확률은 세 가지를 적용했으며 각각 15%, 25%, 50%이다.

(그림 4.1)은 메모리 개수를 1개, 프로세서를 8개로 하여 실험한 결과로써, SR에 따른 시스템의 성능을 나타낸 것으로, 괄호 안의 수치는 부분사상의 비율을 의미한다. LC/PS의 여러 결과를 살펴보면 대체적으로 LC/NS의 경우보다 좋은 성능을 보이는 것을 알 수 있다. 특히 SR이 증가할수록 LC/FS가 LC/NS에 비하여 성능향상이 커지며, 아울러 LC/PS도 비례적으로 성능 향상을 보여준다. 그러나 LC/PS(1/16)의 경우에는 LC/NS의 경우에 비하여 System Power 수치가 저조하거나 비슷한데, 이는 필요한 상태정보를 BC에 저장하기에는 BC가 너무 작아 국부성의 원리를 제대로 이용하지 못하여 스와핑현상이 빈번히 발생하기 때문이다. LC/PS(1/2) 혹은 LC/PS(1/4)의 경우에는 거의 LC/FS 시스템의 경우에 비하여 과도한 성능차이가 없다.

따라서 LC/PS의 경우 부분사상 비율을 1:16 보다 높을 때 LC/FS보다 스와핑으로 인한 약간의 성능감소는 있지만 상태정보를 위하여 절반이하에 해당하는 저장공간으로써 LC/NC보다 좋은 성능을 구할 수 있다. 특히, 컴파일러 등의 도움으로 연속적인 공유블록을 유지할 수 있을 경우 LC/FS에서 상태정보 공간의 5%이하의 상태정보캐쉬의 용량으로 우수하게 성능 향상을 기대할 수 있다.



(그림 4.1) 공유블록에 대한 접근 확률에 따른 시스템의 성능 변화

(Fig 4.1) System performance according to access probability of the shared memory

4.2 프로세서 개수에 따른 변화

본 절에서는 프로세서 개수에 따른 시스템의 성능 변화를 알아보고자 한다. 〈표 4.2〉에서는 메모리 개수가 1개, 공유블록에 접근할 확률은 15%, 프로세서의 개수가 4, 8, 16, 32개, 그리고 메모리 입력버퍼의 크기는 사용되는 프로세서의 개수와 동일하게 주었다. 공유블록의 접근확률에 따른 시스템 성능을 위한 실험에서와 마찬가지로 상태정보를 사용한 시스템의 성능이 더욱 좋아지는 것으로 나타났으며 LC/NS에 대해 LC/FS와 LC/PS(1/2)를 비교해 보면 각각 최고 8.3%와 8.2%의 성능향상을 보이고 있다. 여기서 LC/FS 시스템보다 LC/PS(1/2)는 거의 성능의 감소 없이 LC/FS의 상태정보를 위한 BT에 비하여 50%-92.5%(=1−0.5×0.15)의 용량을 줄인 BC로써 유사한 성능을 보인다.

프로세서의 개수가 증가할수록 LC/NS에 비하여 LC/FS 혹은 LC/PS의 System Power가 크게 향상되는데 이는 프로세서의 개수가 증가함에 따라 하나의 프로세서가 갱신한 블록을 다른 프로세서가 참조하기를 원할 확률이 빈번하기 때문이다. BC의 경우에도 임의의 상태정보 블록이 한번 BC에 적재되면 여러 프로세서로부터 빈번하게 사용된 후에 축출되기 때문에 소규모 용량의 BC인 경우에도 LC/NS보다 향상된 성능을 보여준다. 그러나 프로세서의 개수가 작

〈표 4.2〉 프로세서 개수에 따른 시스템의 성능 변화

〈Table 4.2〉 System performance according to the number of processors

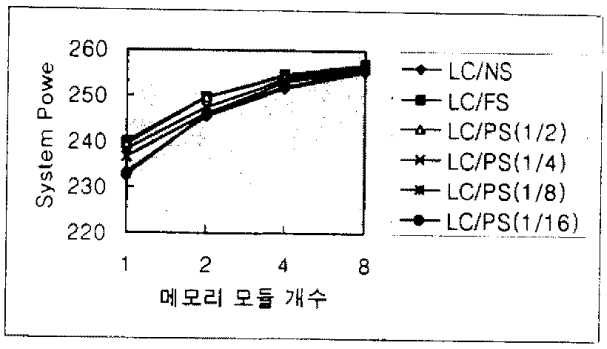
# of CPU / type	4	8	16	32
LC/NS	129.67	233.23	412.00	651.21
LC/FS	132.12	239.70	427.60	705.12
LC/PS(1/2)	132.00	239.20	426.78	704.87
LC/PS(1/4)	130.20	238.10	424.90	703.32
LC/PS(1/8)	129.80	236.40	424.20	701.45
LC/PS(1/16)	128.1	232.46	422.75	698.98

은 경우에는 BC가 작은 용량일 때 LC/NS에 대하여 스와핑 현상으로 인하여 거의 성능향상이 없음을 나타낸다.

4.3 메모리 개수에 따른 변화

(그림 4.3)은 메모리 모듈의 개수가 증가함에 따른 시스템 성능의 변화를 보기 위해 메모리 수를 1, 2, 4, 8개와 같이 변화시켜 가면서 실험한 결과이다. 앞의 실험들과 마찬가지로 프로세서의 개수는 8개이며 메모리 입력 버퍼의 크기도 시스템 내의 프로세서 개수와 동일하게 주었고 공유블록에 대한 접근확률은 15%로 하였다.

(그림 4.3)은 메모리 개수가 많아지더라도 BT 혹은 BC를 통하여 LC/NS에 비해 LC/FS와 LC/PS의 성능이 증가됨을 보여주고 있다. 그러나 성능 향상의 폭이 메모리 모듈의 개수가 증가할수록 줄어드는데 이는 메모리 인터리빙의 효과로 인해 메모리 블록에 대한 상태정보를 이용하는 효과가 감소하기 때문이다. LC/PS(1/16)인 경우에는 작은 용량의 BC로 인하여 빈번한 스와핑이 발생하여 대부분의 경우 LC/NS의 System Power 수치와 비슷하거나 저조함을 나타낸다.



(그림 4.2) 메모리 모듈 개수에 따른 시스템의 성능 변화
(Fig 4.2) System performance according to the number of memory modules

5. 결 론

버스기반 공유메모리-다중프로세서는 시스템의 확장성과 구현의 용이성으로 널리 사용되고 있지만 공유메모리와 공유버스의 접근에 대한 병목현상으로 성능향상에 한계가 있어 이를 극복하기 위한 다양한 연구가 시도되었다. 특히 분리형 트랜잭션 환경에서 병목 현상의 원인을 제거하기 위하여 상태캐쉬를 이용하여 공유메모리가 능동적으로 성능향상에 기여하는 방법이 제안되었지만[5], 이는 상태정보를 공유메모리 전체블록에 대한 완전사상을 이용함으로써 상태정보를 저장하기 위한 고속메모리인 SRAM에 대한 경제적 부담을 나타내었다. 최근 메모리로 이용되는 DRAM의 가격 폭락과 고밀도성, 그리고 응용 프로그램의 대형화 등으로 이에 대한 부담은 더욱 커지고 있다. 따라서 본 연구에서는 상태정보의 저장 장소의 용량을 줄이기 위하여 비공유 블록의 배제와 상태정보의 국부 참조성을 이용한 상태정보캐쉬를 도입하여 소규모 용량의 SRAM이라도 부분사상 비율이 1:16 보다 높다면 기존 방법에 비하여 거의 성능 저하 없이 공유메모리 다중프로세서의 구현이 가능함을 보였다. 특히 상태정보가 연속성을 가질 경우에는 거의 경제적인 부담 없이 완전사상된 상태정보를 이용한 시스템과 유사한 성능을 보였다. 그러나 메모리 내에 분산되어 있는 공유메모리 블록에 대하여 연속성을 유지할 수 있도록 컴파일러 등의 분야에 지속적인 연구가 필요하다.

참 고 문 헌

[1] K. Hwang, *Advaned Computer Architecture*, McGraw-Hill Book Co. 1996.

[2] D. A Patterson and J. L. Hennessy, "Computer Architecture A Quantitative Approach, 2nd. ed.", Morgan Kaufmann Publishers, Inc., 1996.

[3] J. Archibald and J.-L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", ACM Transactions on Computer System, Vol. 4, No. 4, pp. 273-298, 1986.

[4] Shveekant, M. Dubois, A. T. Laundrie and G. S. Sohi., "Scalable Shared_Memory Multiprocessor Architectures", IEEE, pp. 71-74, June, 1990.

[5] 이상은, "캐쉬 상태 정보를 이용한 공유 메모리 모듈의 성능 향상에 관한 연구", 서울대학교 컴퓨터 공학과 공학석사학위 논문, 1995.

[6] R. Katz, S, Eggers, D. A. Wood, C. Perkins, and R. G. Sheldon, "Implementing a Cache Consistency Protocol," 12th ISCA, 1985, pp. 276-283.

[7] 한국전자통신연구소 주전산기 개발 사업단, "고속 중형 컴퓨터(주전산기 III) 공동 연구 개발 사업, 제 3, 4회 연구 개발 워크ˆ발표자료집", 1993.

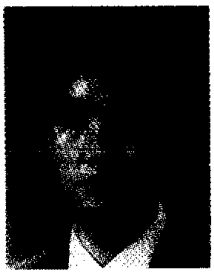
[8] D. J. Shanin, "The Design and Development of a Very High Speed System Bus-The Encore Multimax Nanobus," Proc. of COMPON. pp. 28-36, 1986.

[9] http://www.cs.wits.ac.za/~philip/memories.

[10] S. J. Frank, "Tightly coupled multiprocessor system speeds memory-access times", *Electronics*, pp. 164-169, Jan 12, 1984.

[11] Dubois M., Briggs F., "Effects of Cache Coherency in Multiprocessors," IEEE Transactions on Computers, Vol. C-31, No. 11, pp. 1083-1099, Nov. 1982.

[12] Herb Schwetman, "CSIM User's Guide Rev. 2", Microelectronics and Computer Technology Corporation, July, 1992.



우 종 정

1982년 경북대학교 전자공학과 학사.

1982년~1988년 산업연구원 책임연구원.

1988년~1993년 Univ. of Texas at Austin 전기 및 컴퓨터 공학과 석사 및 박사.

1993년~현재 성신여자대학교 조교수.

관심분야: 병렬처리, 컴퓨터구조, 멀티미디어시스템

이 가 영

1994년 성신여자대학교 전산학과 학사.

1997년 동대학원 이학석사.

관심분야: 병렬처리, 컴퓨터구조