論文95-32B-5-3

공유메모리 다중처리기에서 효율적인 프로세서 동기화 기법

(An Efficient Processor Synchronization Scheme on Shared Memory Multiprocessor)

尹 碩 漢 *, 元 鐵 虎 *, 金 悳 鎭 **

(Sukhan Yoon, Chulho Won, and Duckjin Kim)

요 약

최근 대규모 다중처리 시스템이나 병렬처리 시스템이 다양하게 개발되고 있다. 이런 시스템에서는 여러 프로세서에 의해 공유데이타 경합이 일어나기 때문에 시스템 성능이 저하될 수 있다. 그래서 프로세서간의 동기화가 중요한 과제중의 하나로 대두되었다. 동기화 문제를 해결하기 위해 스핀록(spin lock)을 기반으로 한 소프트웨어 기법과 하드웨어 기법이 많이 제안되어 왔다. 소프트웨어 기법은 구현하기 쉬운 장점은 있지만, 최적화된 성능을 얻기 위해 많은 시스템에서 하드웨어 기법을 사용하고 있다. 본 연구에서는 효율적인 하드웨어 동기화 기법인 QCX를 제안하였고, 설계 고려사항, 하드웨어, 알고리즘, 프로토콜을 기술하였다. 또한 시뮬레이션을 통해 기존에 제안된 QOLB[5], LBP[7] 기법과 성능을 비교하였다. 시뮬레이션에서는 프로세서 수를 30개 까지 변화시키면서 잠금 변수 경합 정도를 변화시켰을 때 일정한 작업량을 완료하는데 걸리는 프로세서당 평균 수행 시간을 측정하였다. 시뮬레이션 결과 QCX 기법이 실제적으로 가장 우수한 것으로 평가되었다. QCX는 QOLB나 LBP 보다 두가지 측면에서 매우 효율적이다. 첫째, QCX는 하드웨어가 간단하고, 캐쉬 구조의 변경이 필요 없기 때문에 구현 비용이 매우 적다. 둘째, 일반성이 높은 아토믹 명령어를 사용하기 때문에 여러 시스템에 쉽게 적용할 수 있다.

Abstract

Many kinds of large scale multiprocessing and parallel-processing systems have recently been developed. The contention on the shared data caused by multiple processors may degrade system performance. So, processor synchronization has become one of the important issues in these systems. To solve the synchornization issues, a lot of software and hardware schemes based on spin lock have been proposed. Although software schemes are easy to implement, hardware schemes are preferred in many systems to gain optimized performance. This paper proposes an efficient processor synchronization scheme, called QCX,and describes its design considerations, hardware, algorithm, protocol. Also, in this paper, the performance of QCX has been evaluated with QOLB[5] and LBP[7] using a simulation. The simulation, with varying the number of processor and the contention on shared variables, measured the average execution times of a workload. The simulation results show that the performances of QCX is best when practicability is considered. QCX is more efficient than QOLB and LBP in two aspects. First, the hardware of QCX is more simple and cost-effective because the cache structure need not be changed. Secondly, QCX is more general because it uses a generic atomic instruction.

* 正會員, 韓國電子通信研究所 프로세서연구실
(Processor Section, ETRI)
** 正會員, 高麗大學校 電子工學科
(Dept. of Elec. Eng., Korea Univ.)
接受日字:1994年7月6日, 수정완료일:1995年4月27日

I. 서 론

처리해야 할 데이타가 급증하면서 컴퓨터 성능에 대한 요구가 증가하고 있고, 이 요구를 충족시키기 위해서 최근 다중처리 시스템이나 병렬처리 시스템이 많이 개발되고 있다. 이런 시스템들은 대규모 프로세서(processor-CPU)를 상호연결망으로 연결하여 하나의 시스템으로 동작한다. 이렇게 한 시스템내에 프로세서 수가 많아지면 자연히 프로세서간의 데이타 공유와 프로세서간의 동기화를 위한 통신이 증가한다[6].

데이타 공유란 프로세서간의 정보교환을 의미하고, 동기화란 원활한 정보교환을 위하여 시스템내에 존재하는 공유자원을 제어하는 기법이라고 말할 수 있다. 공유자원은 시스템내의 한정된 중요 자원이기 때문에 동기화 기법에 따라 효율적인 이용이 좌우되고, 이로 인해 시스템 성능에 큰 영향을 미친다. 특히 대규모 프로세서를 연결하는 시스템에서 효과적인 동기화 기법이 제공되지 않을 때 통신 자원이나 메모리 자원이 고갈되어 시스템 성능이 급격히 떨어지는 현상이 발생한다. 이처럼 동기화 기법이 시스템 성능에 큰 영향을 주기 때문에 이에 대한 연구는 절실하다.

1980년대 TICOM[8], Multimax[9], Symmetry[10] 등 성능이 우수한 다중처리 시스템이 개발되었다. 하지만 이런 시스템에서도 프로세서 수가 증가함에 따라 병목 현상이 발생할 수 있는데 주요 원인 중의 하나가 프로세서 동기화 때문이다. 그동안 효과적인 프로세서 동기화를 위해 프로세서 자체에 동기화 프리미티브(primitive)를 갖게 되었다. Test&Set, Test&Test&Set[12], Read&Increment[11], Fetch&Operation[13], Full/Empty[14], QOLB[5] 등 많은 프리미티브가 제안되어 여러 시스템에서 사용하고 있으며 최근에는 프로그래머가 쉽게 사용할 수 있게 일반성이 있는 아토믹 명령어[11]로 발전되었다. 아토믹 명령어를 사용한 스핀록(spin lock) 동기화 기법은 그 구조가 매우 간단하여, 다른 동기화 기법에 사용될 수 있고, 임계영역(critical section)이 짧은 응용에는 매우 효과적이기 때문에[11] 많은 컴퓨터에서 사용하고 있다. 하지만 스핀록은 하나의 잠금변수(lock variable)에 대해 여러 프로세서가 경합할 때, 잠금변수를 획득한 프로세서만 정상적인 일을 하고 나머지 프로세서들은 스핀하면서 계속 경합한다. 이 스핀 동작은 i) 통신 대역폭을 많이 사용하고, ii) 잠금변수를 획득한 프로세서의 수행을 지연시키고, iii) 일시적인 과부하 현상인 hot spot을 초래한다. 이 세가지 문제를 해결하기 위하여 스핀 프로세서를 queuing시키고 잠금변수를 caching시키는 효율적인 동기화 기법이 제공되어야 한다.

그동안 효율적인 queuing과 caching을 위해 소프트웨어 기법과 하드웨어 기법에 대한 연구가 수행되었다. 소프트웨어 기법으로는 캐쉬의 장점(caching 기능)을 이용한 Snooping Lock 기법[12], 과부화 현상을 줄이기 위한 Collision Avoidance Lock 기법[11], 경합하는 프로세서 수를 줄이기 위한 Tournament Lock 기법[2], 그리고 스핀하는 프로세서의 경합을 순차화시키는 Queuing Lock 기법[1]들이 연구되었다. 소프트웨어 기법은 임계영역이 긴 응용에서는 효과적이지만, 임계영역이 짧은 응용에서는 소프트웨어 처리시 발생하는 오브헤드가 크다는 단점[3]과 성능 향상에는 한계가 있다. 이를 해결하기 위해 하드웨어 방법으로는 메모리 제어기에 잠금변수를 처리하는 기법[8,9]이 연구되었고, 동기화 프로세서를 사용한 기법[10]이 연구되었다. 최근에는 메모리와 캐쉬에서 스핀 프로세서 큐를 처리하는 QOLB(Queue On Lock Bit) 기법[5]이 제안되었으며, 캐쉬 프로토콜과 동기화 프로토콜을 완전히 결합시킨 LBP(Lock Based Protocol) 기법[7]에 대한 연구가 있었다. 캐쉬와 결합된 하드웨어 기법은 소프트웨어 기법에 비해 성능은 개선되었지만[6] 캐쉬 프로토콜이 복잡해지고, 이로 인해 캐쉬가 매우 복잡해져 캐쉬 효율이 떨어지고 구현 비용이 증가한다. 그리고 캐쉬라인 대체(replacement)시 잠금변수 처리가 매우 어려운 단점[7]이 있다. 더욱 심각한 문제는 캐쉬 구조의 변경이다. 최근에 개발되고 있는 마이크로프로세서들은 성능 향상과 구현 용이성을 위해 마이크로프로세서 내부에 캐쉬를 내장하고 있다. 이 경우 앞에서 기술한 하드웨어 기법을 사용할 수 없다. 이 문제들을 해결하기 위해 본 논문에서는 캐쉬와 완전히 분리된 하드웨어 동기화 기법인 QCX(Queuing and Caching with the eXchange primitive)를 제안하고 그 설계 내용에 대해 기술하였다. 그리고 시뮬레이션을 통하여 QCX의 성능을 QOLB, LBP와 비교 분석하였다.

본 논문의 구성은 다음과 같다. 제2장에서는 스핀록의 알고리즘과 동작에 대해 설명하고 문제점을 분석하였다. 그리고 QOLB와 LBP 기법에 대해 간단히 기술하였다. 제3장에서는 QCX의 설계 고려사항, 하드웨어, 알고리즘, 프로토콜에 대해서 기술하였다. 제4장에서는 시뮬레이션 모델과 시뮬레이션시 사용한 작업부하 모델에 대해서 기술하였고, 시뮬레이션 결과를 분석하였다. 마지막으로 제5장에서는 결론과 앞으로의 연구 방향에 대해서 언급하였다.

Ⅱ. 하드웨어 동기화 기법

1. 스핀록 기법

그림 1은 exchange/release 프리미티브를 사용한 임계영역에서 스핀록 사용 예를 보여 주고 있다. 그림 1에서 보는 것처럼 스핀록은 구조가 간단하고 임계영역을 쉽게 구현할 수 있기 때문에 많은 시스템에서 사용하고 있다.

```
while(EXCHANGE('locked',lock_variable)='
      locked') do nothing ;
        .
        .
   (Critical Section)
        .
        .
RELEASE(lock_variable);
```

그림 1. 임계영역에서 사용된 스핀록
Fig. 1. Spin Lock Used on Critical Section.

스핀록에서는 잠금변수를 '잠금 (locked)' 상태로 만든 프로세서만 임계영역으로 들어가고 나머지 프로세서들은 스핀하면서 계속 잠금변수를 접근하여 시스템 성능에 큰 영향을 미친다. 첫째, 잠금변수가 공유메모리에 존재하기 때문에 스핀 프로세서들은 잠금변수 경합을 위해 공유메모리를 접근한다. 이로 인해 상호연결망의 대역폭을 많이 소모하고 시스템 성능을 저하시킨다. 특히 공유버스 시스템인 경우 버스를 통해 공유메모리를 접근하기 때문에 버스 포화 상태를 유발시킨다. 둘째, 공유메모리를 계속 접근하기 때문에 잠금 변수를 획득한 프로세서의 공유메모리 접근을 방해하여 일을 더욱 지연시킨다. 셋째, 캐쉬를 사용하는 시스템의 경우 잠금변수가 해제될(release) 때 다른 프로세서의 캐쉬가 모두 무효화되어(invalidated) 캐쉬 읽기 미쓰가 일어난다. 이때 모든 스핀 프로세서가 일시에 공유메모리를 접근하기 때문에 hot spot이 발생된다. 이 문제들을 해결하기 위해 잠금변수를 caching하고 스핀 프로세서를 queuing하는 효과적인 동기화 기법이 제공되어야 한다. 잠금변수 caching은 잠금변수를 캐쉬에 읽어와 캐쉬에서 스핀하게 만들어 공유메모리 접근을 줄일 수 있고, 스핀 프로세서 queuing은 스핀 프로세서를 큐에 대기하게 만들어 잠금변수 경합을 줄일 수 있다.

2. QOLB 기법

QOLB 기법[5]에서는 Test&Set 프리미티브를 사용하면서 새로운 두가지를 제안하였다. 첫째, 일종의 잠금변수인 잠금비트(lock bit)를 메모리의 각 라인(캐쉬라인과 일치시킴)에 할당하고 이 잠금비트와 연결되는 FIFO 큐를 만들었다. 둘째, non-blocking, pre-queuing 프리미티브인 QOLB를 제안하였다. QOLB 프리미티브가 수행되면 프로세서는 스핀하지 않고 잠금비트에 연결된 큐에 자신을 pre-queuing 시킨다. 그리고 본래의 일을 수행하다가 잠금비트 획득이 필요한 시점에 도달하면 그때 큐에 들어가고, 그후 자신의 차례가 오면 Test&Set 프리미티브를 사용하여 잠금비트를 획득한다. 이 기법은 잠금비트와 공유데이타를 하나의 캐쉬라인으로 만들어 caching 효과를 얻을 수 있고, 또한 프로세서를 pre-queuing시켜 잠금비트 대기 시간을 줄이고 프로세서간의 경합을 없애주는 queuing 효과를 얻을 수 있다. 하지만 메모리 각 라인마다 잠금비트를 두어야하고, 두개 이상의 메모리 라인을 차지하는 공유데이타인 경우 잠금비트가 두개 이상 생기기 때문에 잠금처리가 쉽지 않다. 그리고 캐쉬 내에서 잠금비트와 큐를 처리하기 때문에 캐쉬가 복잡해지고 구현 비용이 높아지는 단점이 있다.

3. LBP 기법

LBP 기법[7]에서는 캐쉬와 동기화 프로토콜을 완전히 결합한 새로운 프로토콜을 제안하였다. 캐쉬에 동기화 프로토콜을 추가하기 위해 캐쉬 읽기/쓰기 모드에 4가지 잠금변수 읽기/쓰기 모드(read-lock, write-lock, read-unlock, write-unlock)를 추가하였으며, 이를 제어하기 위해 13개의 캐쉬 상태를 추가하였다. 그리고 캐쉬내에 잠금변수와 연결된 FIFO 큐를 만들었고, 이 큐를 이용하여 스핀 프로세서를 큐에 대기시킨다. 동기화 프로토콜이 캐쉬와 결합되어 caching과 queuing을 처리하기 때문에 캐쉬 상태가 매우 복잡하여 캐쉬 효율이 떨어지고 캐쉬를 구현하는 비용이 높아진다. 그리고 캐쉬라인 대체가 일어날 때 잠금변수 대체가 동시에 일어나기 때문에 이에 대한 효과적인 해결책이 마련되어야 한다.

Ⅲ. QCX 동기화 기법

1. 설계시 고려 사항

본 논문에서 제안하는 QCX(Queuing and Caching with eXchange primitive) 기법은 exchange/release 프리미티브를 사용하여 스핀 프로세서를 queuing하고 잠금변수를 caching하는 하드웨어 동기화 기법이다. 이 기법은 동기화 수행시 i) 잠금변수의 아토믹 접근을 보장하고, ii) 잠금변수를 caching하고, iii) 스핀 프로세서를 queuing하고, iv) 공유 버스에서 일어나는 동작을 스누핑하는 기능을 갖고 있다. 그리고 QCX가 구현될 시스템은 다음과 같은

기능을 제공할 수 있는 것으로 가정하였다. i) 스핀록을 위하여 non-cacheable 잠금변수를 사용하고, ii) exchange/release 프리미티브를 구현하기 위해 아토믹 명령어가 제공되어야 하고, iii) 버스 스누핑이 제공되어야한다. II.1 절에서 기술한 스핀록의 문제점을 효과적으로 해결하고 위의 가정을 수용하기 위해서 QCX를 설계할 때 다음 사항을 고려하였다.

- 기존의 캐쉬와 분리된 동기화 프로토콜을 설계한다.
  기존의 캐쉬 구조를 변경하지 않아야 하는 이유는 다음과 같다. 첫째, 최근에 개발된 마이크로프로세서는 성능을 향상시키고 구현 용이성을 사용자에게 제공하기 위해 마이크로프로세서 내부에 캐쉬를 내장하고 있다. 이런 마이크로프로세서를 사용하여 시스템을 개발하는 경우 캐쉬 구조를 변경할 수 없다. 둘째, 동기화를 캐쉬에서 처리하면 캐쉬 효율이 떨어지고, 캐쉬가 복잡해져 캐쉬 구현 비용이 증가한다. 셋째, 잠금변수는 일반 데이타와 달리 동시에 여러 프로세서들에 의해 읽기와 쓰기가 집중되는 특성을 갖고 있기 때문에 이를 효과적으로 처리할 수 있는 전용 하드웨어가 필요하다.
- 잠금변수를 caching 할 수 있어야 한다.
  각 프로세서에 데이타 캐쉬와 별도로 잠금변수 캐쉬(1라인)를 두어 caching 문제를 해결하였다. 즉, 잠금변수는 여러 프로세서들에 의하여 동시에 읽기/쓰기 시도가 발생하지만 한순간에 하나의 프로세서만 읽고/쓰기를 허용하므로써 아토믹 접근을 보장하였다. 그리고 잠금변수를 잠금 캐쉬에 저장하여 caching 기능을 갖도록 하였으며, 다른 프로세서에서 일어나는 동작과 상태를 관찰하기 위해 버스를 통한 스누핑 기능을 두었다.
- 스핀 프로세서를 queuing 할 수 있어야 한다.
  각 프로세서에 분산 FIFO 큐를 만들어 queuing 문제를 해결하였다. 잠금변수를 경합하는 프로세서들을 큐를 통하여 순차화시켜, 스핀 프로세서를 없애고 잠금변수 해제시 발생하는 hot spot을 제거하였다.
- 일반성이 높아야 한다.
  동기화를 캐쉬와 분리시켜 캐쉬 구조와 프로토콜에 영향을 주지 않고, 마이크로프로세서에서 제공하는 아토믹 명령어을 사용하여 여러 시스템에서 쉽게 적용할 수 있도록 하였다.
- 하드웨어 구현 비용이 적어야 한다.
  앞에서 언급한 4가지 고려사항을 충분히 반영하여 하드웨어 구현 비용을 최소화 시킬 수 있는 경제적이고 실제적인 동기화 기법을 설계하였다.

2. QCX 하드웨어

그림 2는 QCX의 하드웨어 블록 다이아그램이다. 하드웨어는 잠금캐쉬(Lock Cache, LC), 잠금번지(Lock Address, LA), 큐(Queue, Q), Tail Flag(TF), 잠금데이타 비교기 (Data Comparator, DC), 잠금번지 비교기 (Address Comparator, AC), 프로세서 ID 비교기 (ID Comparator, IC), 스누핑 로직(Snooping Logic), 그리고 이들을 제어하는 스테이트 머신(State Machine, SM)으로 구성된다. LC는 잠금변수를 저장하는 1라인 캐쉬이고, LA는 잠금번지를 저장하는 버퍼이고, Q는 linked list 형태의 큐를 형성하기 위한 ID 버퍼이고, TF는 tail을 나타내는 플래그이다(이 플래그는 스테이트 머신 안에 존재하기 때문에 점선으로 표시하였다.). 그리고 AC는 잠금번지를 비교하여 같은 번지의 잠금변수인지 판단하고, DC는 잠금변수의 값을 비교하여 잠금 상태인지 판단하고, IC는 선행 프로세서의 ID와 잠금변수를 해제한 프로세서 ID를 비교하여 큐의 head가 될 것인지 판단한다. 그리고 SM은 하드웨어 전체를 제어한다.

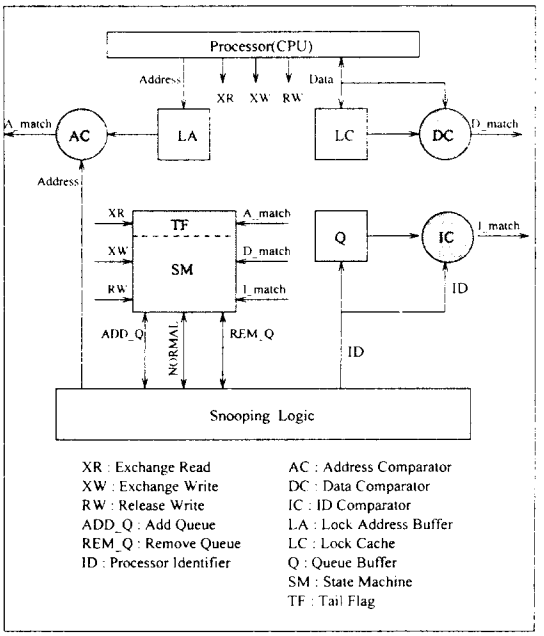

그림 2. QCX 하드웨어 블록 다이아그램
Fig. 2. Hardware Block Diagram of QCX.

SM의 상태는 프로세서에서 발생되는 잠금읽기(XR : eXchange Read), 잠금쓰기(XR : eXchange Write), 해제쓰기(RW : Release Write) 신호와 각 비교기에서 출력되는 match 신호 (A : 번지, D : 데이타, I : ID)와 그리고 버스 스누핑과 관련된 어트리

뷰트 신호 (ADD_Q : 큐에 새로 가입, REM_Q : 큐 head를 제거함, NORMAL : 큐와 무관)에 의해서 결정된다.

XR은 프로세서가 아토믹 명령어 수행중 잠금변수를 읽을 때 발생하고, XW는 잠금변수를 쓸 때 발생한다. 그리고 RW는 잠금변수를 해제할 때 발생한다.

3. Caching 및 Queuing 알고리즘

잠금변수의 caching 알고리즘은 그림 2의 잠금캐쉬(LC), 잠금데이타 비교(DC), 잠금번지 비교(AC), 그리고 버스 스누핑을 통해서 수행된다. 한번 읽혀진 잠금변수는 잠금캐쉬에 저장되며 스핀 프로세서는 공유메모리 대신 잠금캐쉬를 접근하여 caching 효과를 얻을 수 있다. 잠금데이타 비교는 잠금변수의 잠금 상태를 판단하고, 잠금번지 비교는 같은 번지의 잠금변수에 대해서 잠금 동작이 발생했는지 판단한다. 잠금변수의 데이타 비교 결과를 이용하면 프로세서의 잠금 성공여부를 자체적으로 판단할 수 있다.

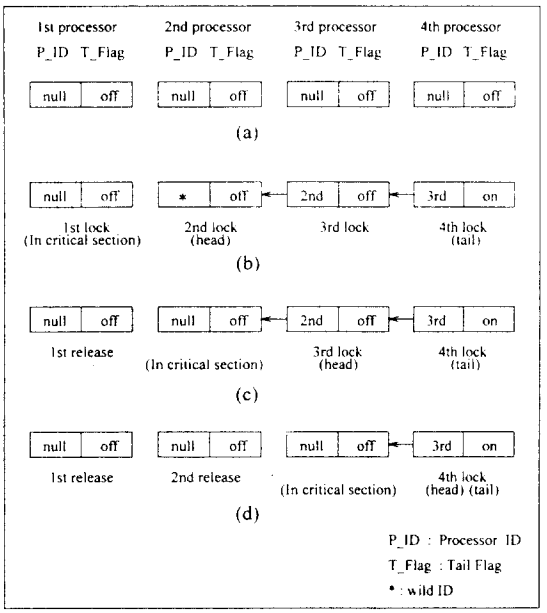

그림 3. Queuing 예
Fig. 3. An Example of the Queuing.

스핀 프로세서의 queuing 알고리즘은 그림 2의 FIFO 큐(Q), 테일 플래그(TF), 프로세서 ID 비교(IC), 그리고 버스 스누핑을 통해서 수행된다. 그림 3은 queuing 알고리즘의 예를 보여주고 있다. 큐 버퍼(그림3의 Q)는 스핀 프로세서 큐를 형성하기 위해 선행 프로세서 ID를 갖고 있고, 테일 플래그(그림 3의 TF)는 큐의 마지막 프로세서임을 나타낸다. 이 플래그가 on 된 프로세서는 잠금을 시도하는 후행 프로세서를 큐 tail에 연결하기 위해 자신의 프로세서 ID를 보내주어야 하는 책임이 있다. 그리고 프로세서 ID 비교는 잠금변수를 해제하는 프로세서가 자신의 선행 프로세서인지 비교하여 큐의 head가 될 수 있는지 결정한다. 큐의 head가 되는 조건은 잠금변수를 해제하는 프로세서가 선행 프로세서이거나, 선행 프로세서가 와일드 ID(맨처음 큐를 형성하는 프로세서는 선행 프로세서가 없기 때문에 항상 스스로 와일드 ID를 갖는다.)일 때이다. 그리고 버스 스누핑은 프로세서 ID를 얻고 잠금변수에 대한 동작을 감지한다. 이런 기본 구조를 이용한 잠금시 queuing 알고리즘은 아래와 같고, 그 예는 그림 3의 LOCK에서 보여주고 있다.

잠금시 알고리즘

Step 1:

첫번째 잠금을 시도한 프로세서는 잠금변수를 획득한다. 첫번째 프로세서는 잠금변수를 획득하였기 때문에 큐를 만들지 않는다.

Step 2:

두번째 잠금을 시도한 프로세서는 잠금 실패가 발생하여 버스를 스누핑한다. 이때는 아직 tail 프로세서가 생성되지 않은 상태로 ID를 보내주는 선행 프로세서가 없기 때문에 큐의 head가 되기 위해 스스로 와일드 ID(그림 3에서 *로 표시)를 갖는다. 그리고 동시에 tail 플래그를 on시켜 큐의 tail이 되고 후행 프로세서에게 자신의 ID를 전송할 책임을 진다.

Step 3:

세번째 잠금을 시도한 프로세서 역시 잠금 실패가 발생하여 버스를 스누핑한다. 이때는 두번째 프로세서가 큐의 tail이기 때문에 자신의 프로세서 ID를 세번째 프로세서에게 보내주고 tail 플래그를 off 시켜 tail에서 벗어난다. 세번째 프로세서는 버스 스누핑을 통해 받은 ID를 큐 버퍼에 저장하여 두번째 프로세서가 선행 프로세서가 되도록 큐를 형성한다. 그리고 tail 플래그를 on시켜 후행 프로세서에게 자신의 ID를 전송할 책임을 진다.

Step 4:

네번째 잠금을 시도한 프로세서는 세번째 프로세서와 Step 3.에서 일어난 동작을 수행한다.

이런 연속적인 잠금 동작은 그림 3의 Lock (After 4 locks)과 같이 linked list 형태의 FIFO 큐가 만들어 진다. 그림 3에서 두번째 프로세서의 Q가 와일드 ID(*로 표시)로 되어 있는 것은 맨처음으로 큐를 형성한 프로세서이기 때문이다.

잠금 변수 해제시 queuing 알고리즘은 아래와 같고, 그 예는 그림 3의 Release에서 보여주고 있다.

해제시 알고리즘

Step1:

잠금변수를 해제하는 프로세서는 큐에서 대기하고 있는 모든 프로세서에게 잠금해제 신호와 함께 자신의 ID를 보내준다.

Step 2:

큐에 있는 프로세서는 잠금해제 프로세서의 ID와 자신의 선행 프로세서 ID를 비교하여 같으면 큐에서 탈퇴한다. 이때 와일드 ID는 어떤 프로세서 ID와도 match가 일어나기 때문에 와일드 ID를 갖고 있는 프로세서는 항상 큐에서 탈퇴한다(앞에서 기술한 잠금시의 알고리즘이 지켜지면 와일드 ID match와 프로세서 ID match가 동시에 발생하는 경우는 없다.). 큐에서 탈퇴한 프로세서는 큐를 null로 만들고 잠금변수를 획득한다.

Step 3:

이때 프로세서 ID match가 일어나지 않은 프로세서들은 아무 동작하지 않고 큐에서 대기한다.

4. QCX 동기화 프로토콜

그림 4는 QCX 동기화 프로토콜의 상태천이를 나타내고 있다. SM은 I)nitial, A)tomic, T)ail in queue, Q)ueue, H)ead in queue 상태를 갖고 있다.

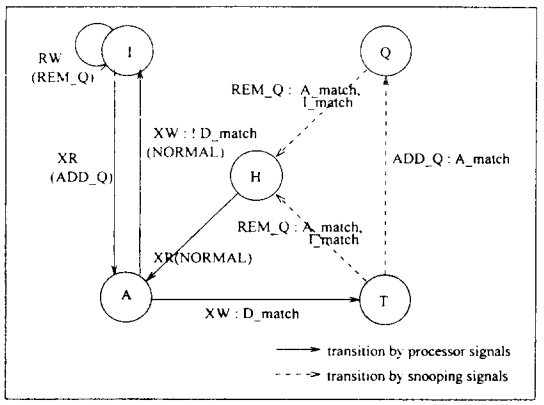

그림 4. QCX 프로토콜의 상태 천이도
Fig. 4. State Transition Diagram of QCX Protocol.

그림 4에서 실선은 프로세서의 읽기/쓰기 신호(XR,XW,RW)에 의해서 상태천이가 일어나는 것을 의미하고, 점선은 버스 스누핑 신호(ADD_Q, REM_Q)에 의해서 상태천이가 일어나는 것을 의미한다. 그리고 각 비교기의 match 신호는 상태천이가 일어날 조건으로 이용된다. 다음은 SM의 각 상태에서 일어나는 동작을 설명한 것이다.

I 상태:

잠금을 시도하지 않은 상태 혹은 잠금을 완료한 상태이다. 이 상태에서는 잠금 시도와 잠금 해제(잠금을 완료한 상태이기 때문에 잠금을 해제할 수 있다.)를 할 수 있다. 잠금 시도가 일어나면 잠금 변수에 대한 잠금읽기(XR)를 수행하며 잠금을 실패할 경우 큐에 들어간다는 신호(ADD_Q)를 발생시키고 A 상태로 들어간다. 한편 잠금 해제가 일어나면 잠금 해제쓰기(RW)를 수행하면서 후행 프로세서를 큐의 head가 되도록 만들어 주는 신호(REM_Q)를 발생시킨다. 이 신호를 받은 후행 프로세서(T 혹은 Q 상태에 있음)는 H 상태로 된다.

A 상태:

잠금변수에 대한 아토믹 읽기/쓰기 수행을 보장하는 상태이다. 잠금변수에 대한 잠금읽기와 잠금쓰기의 값이 다르면(XW: !D_match) 잠금을 성공하였기 때문에 '잠금'으로 만들고 I 상태로 되돌아간다. 값이 같으면(XW : D_match) 잠금을 실패하였기 때문에 T 상태로 들어간다. 즉, 가장 최근에 잠금을 시도하여 실패한 프로세서이기 때문에 큐의 tail 이 된다.

T 상태:

큐의 tail 상태이다. 후행 프로세서가 같은 번지의 잠금변수에 대해 잠금시도를 실패(ADD_Q : A_match)하면 후행 프로세서에게 큐의 tail을 넘겨주고 자신은 Q 상태로 들어간다. 한편 같은 번지의 잠금변수에 대해 자신의 선행 프로세서가 잠금해제(REM_Q : A_match, I_match)하면 Q 상태를 거치지 않고 바로 H 상태로 들어가 큐에서 탈퇴한다.

Q 상태:

큐의 가운데 있는(head도 tail도 아닌) 상태이다. 같은 번지의 잠금변수에 대해 자신의 선행 프로세서가 잠금해제 (REM_Q : A_match, I_match)를 하면 H 상태로 들어가 큐에서 탈퇴한다. 위의 조건이 만족하지 않으면 Q 상태에서 계속 대기한다.

H 상태:

큐에서 탈퇴하는 상태이다. 잠금변수를 획득할 차례이므로 잠금변수를 읽고(XR) A 상태로 들어 간다. A 상태로 들어간 후 잠금변수를 획득했다는 신호(XW : !D_match)를 보내고 I 상태로 들어간다. 다른 상태와 달리 H 상태는 잠금을 수행하는 과정에

있기 때문에 다른 프로세서의 잠금 변수 접근은 허용되지 않는다.

Ⅳ. 성능 시뮬레이션

1. 시뮬레이션 모델

QCX 기법의 동기화 성능을 평가하기 위해 시뮬레이션을 수행하였다. 시뮬레이션에서 사용할 하드웨어 모델과 작업부하 모델을 설정하였다. 하드웨어 모델은 공유버스를 사용하는 전형적인 다중 프로세서 시스템이고 표 1의 작업부하 모델은 다중 프로세서 시스템의 캐쉬 일관성 성능을 평가하기 위해 많이 사용되고 있는 Archibald 모델[15]이다. 시뮬레이션 수행은 작업부하 모델을 이용하여 i) 프로세서 갯수를 변화시키고, ii) 잠금변수에 대한 경합을 변화시켰을 때 성능을 측정하였다. 그리고 QCX의 동기화 성능을 비교 평가하기 위해 QOLB와 LBP에 대해서도 동일한 환경에서 동일한 방법으로 시뮬레이션하였다.

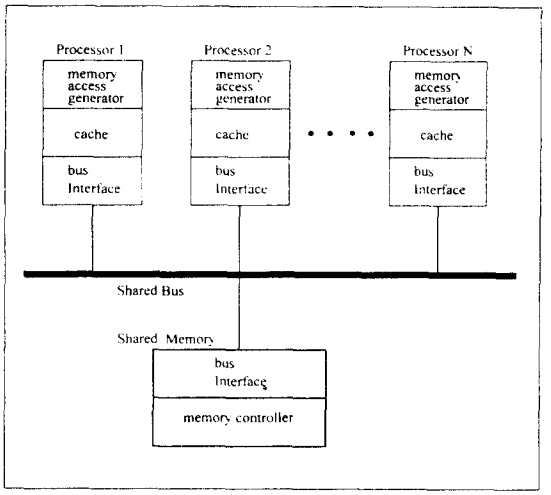

그림 5. 하드웨어 시뮬레이션 모델
Fig. 5. Hardware Simulation Model.

시뮬레이션에서 사용한 하드웨어 모델은 그림 5에서 보는 것과 같이 여러개의 프로세서가 공유버스를 통해 공유메모리를 접근하는 다중 프로세서 구조이다. 이 모델에서 프로세서는 메모리 접근 발생기 모듈과 캐쉬 모듈과 그리고 버스 인터페이스 모듈로 구성되며, 공유메모리는 버스 인터페이스와 메모리 제어기 모듈로 구성되어 있다. 프로세서들에서 발생하는 메모리 접근은 요구 순서에 따라 처리된다. 메모리 접근 발생기는 작업부하에 따라 메모리 요구를 발생하며, 발생된 메모리 요구는 먼저 캐쉬를 접근하고 캐쉬 미쓰가 발생하면 다시 버스를 통해 메모리를 접근한다. 캐쉬 모듈은 캐쉬 일관성 프로토콜과 동기화 프로토콜을 갖고 있으며 QCX, QOLB, LBP 프로토콜 중에서 선택할 수 있도록 시뮬레이터를 만들었다. 캐쉬 라인의 크기는 64바이트이고, 버스 전송 폭은 16바이트이기 때문에 캐쉬 라인을 채울 때 4번의 데이타 전송이 버스에서 일어난다.

표 1은 시뮬레이션에서 사용한 작업부하 모델과 파라미터들을 보여주고 있다. 각 파라미터의 의미는 다음과 같다. 시간적 국부성(locality)을 반영하기 위해 LRU 스택을 사용하는 32개의 공유데이타를 두었다. 프로세서에서 발생하는 메모리 접근 중에 이 공유데이타를 접근할 확률은 shared이다. 메모리 접근 중에 읽기 확률은 read이고, 메모리 접근시 캐쉬에 적중할 확률은 hit이다. 캐쉬 대체가 일어날 때 대체될 캐쉬 라인이 수정 상태일 확률은 modified이다.(캐쉬 대체가 일어날 때 캐쉬가 수정 상태이면 write-back이 일어난다.) 메모리 접근을 마친 프로세서는 wait 클럭 후에 다시 새로운 공유메모리 접근을 시작한다.

표 1. 작업부하 모델과 파라미터
Table 1. Workload Model and Parameters.

파라미터	사용한 값	범위
shared	1%, 3%, 5%	0.1% - 5%
read	80%	70% - 80%
hit	95%	95% - 98%
modified	30%	30% - 40%
wait	1	0 - 5
no. of shared lines	32	16 - 1024
line size	16 words (64바이트)	16 words (64바이트)
cache size	1024 lines	128 - 1024 lines
task size	100 access	
workload	120 tasks	
no. of processors	4,5,6,10,12,15,20,25,30	

캐쉬 line size는 64바이트이며 각 프로세서마다 1024개 라인의 cache size를 갖고 있다. task size는 메모리를 100번 접근하도록 만들었으며, 시뮬레이션시 사용한 workload는 120개의 task를 각 프로세서에게 균등하게 할당하고 이를 병렬로 수행하게 만들었다. 시뮬레이션시 두가지 파라미터를 변화시켰다. 하나는 공유데이타 비율을 1%, 3%, 5%로 변화시키고, 다른 하나는 프로세서 갯수를 4, 6, 8, 10, 12, 15,

20, 25, 30개로 변화시켰다. 그리고 동기화 성능을 비교하기 위해 하나의 임계영역에서 잠금변수 경합이 모두 발생하도록 만들었다. 다시 말하면, 정상적인 상태에서는 병렬성을 극대화하기 위해 공유데이타를 가능한 세분화(fine grain)시켜야 하나, 시뮬레이션에서는 잠금변수 경합을 발생시키기 위해 하나의 임계영역내에 모든 공유데이타가 존재하도록 만들었다.

2. 시뮬레이션 결과

앞에서 설명한 하드웨어 모델과 작업부화 모델을 기준으로 QCX와 QOLB와 LBP의 동기화 성능을 시뮬레이션을 통해 측정하였다.

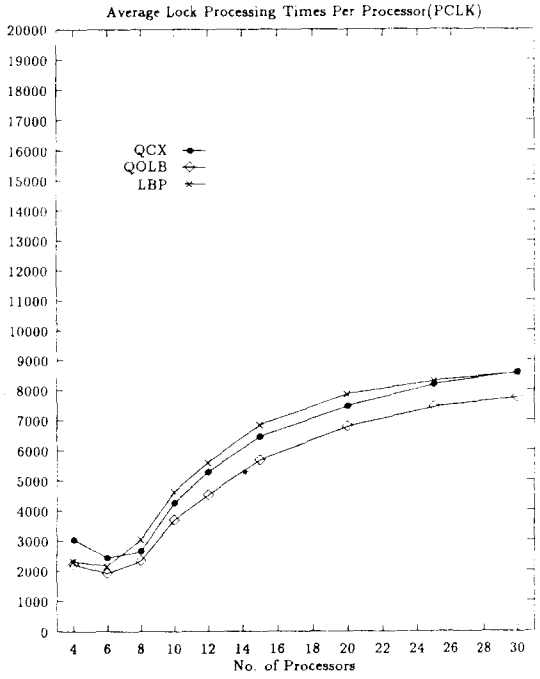

그림 6. 평균 잠금처리 시간(1% 공유데이타)
Fig. 6. Average Lock Processing Time(1% Shared Data).

그림 6과 그림 7과 그림 8은 공유데이타 비율이 각각 1%, 3%, 5%일 때 프로세서 갯수를 30개까지 증가시켜면서 동기화 성능을 측정한 것이다. 그림 6은 공유데이타 비율이 1%로 비교적 경합이 적은 경우로 동기화 성능은 QOLB가 가장 좋은 것으로 측정되었다. 그림 7은 공유데이타 비율이 3%로 경합이 중간 정도로 세 기법의 동기화 성능은 거의 차이가 없다. 그림 8은 공유데이타의 비율이 5%로 비교적 경합이 심한 경우로 동기화 성능은 LBP가 가장 좋은 것으로 측정되었다.

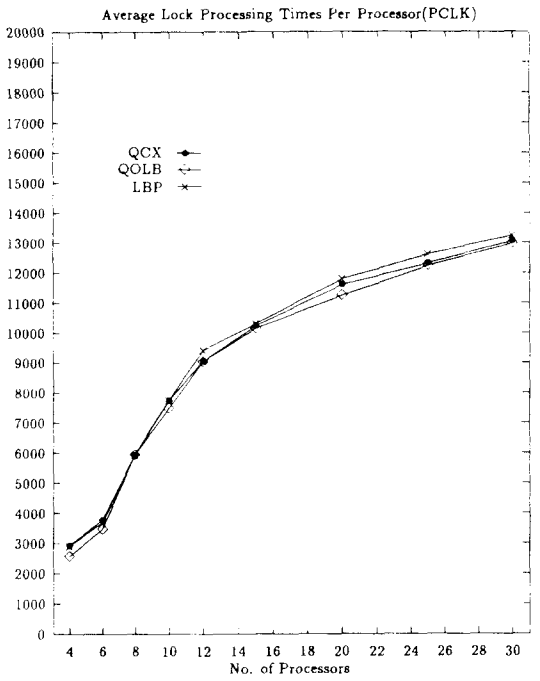

그림 7. 평균 잠금처리 시간 (3% 공유데이타)
Fig. 7. Average Lock Processing Time (3% Shared Data).

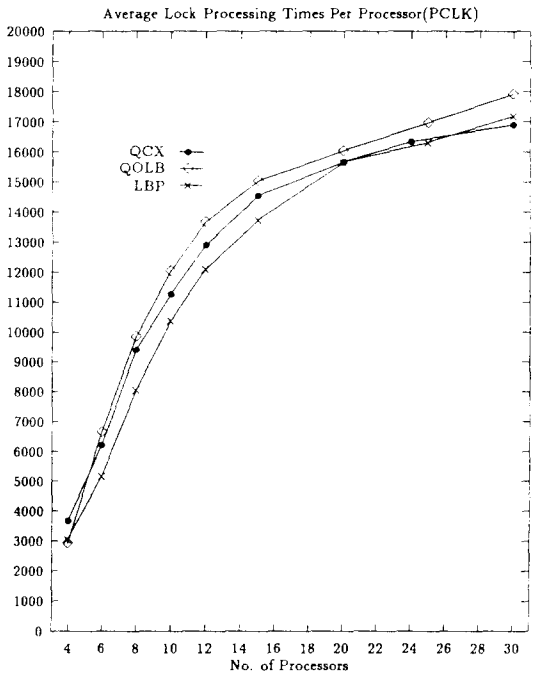

그림 8. 평균 잠금처리 시간 (5% 공유데이타)
Fig. 8. Average Lock Processing Time (5% Shared Data).

특징적인 것은 각각의 경우에 QCX는 상대적으로 일정한 성능을 유지하고 있는 반면 QOLB와 LBP는 공유데이타의 경합 정도에 따라 성능 차이가 심하게 나타났다. 이는 QCX의 caching과 queuing 구조가 캐쉬와 분리되어 있기 때문에 수행 환경에 대해 민감하지 않은 것으로 판단된다. 그리고 성능 측면만 보면 QCX가 두번째이지만 하드웨어 구현 가능성과 경제성을 고려하여 종합적으로 평가할 때 QCX가 가장 효율적이다. 즉, LBP는 매우 복잡한 캐쉬 구조를 갖고 있기 때문에 구현하기가 매우 힘들어 현실적이지 못하며, 경합이 심할 때는 QCX가 QOLB 보다 우수하기 때문에 QCX가 가장 효율적인 동기화 기법으로 평가할 수 있다.

V. 결 론

스핀록 동기화 기법은 구조가 간단하고 임계영역이 짧은 응용에서는 매우 효과적이기 때문에 많은 시스템에서 사용되고 있지만, 프로세서간 공유데이타에 대한 경합이 심할 때 시스템 성능이 갑자기 저하되는 현상을 초래한다. 이 단점을 해결하는 방법은 잠금변수를 caching 하고, 스핀 프로세서를 queuing 해야한다. 본 논문에서는 이를 효율적으로 처리할 수 있는 새로운 하드웨어 동기화 기법인 QCX의 하드웨어 설계, 알고리즘, 그리고 프로토콜을 제안하였다. 그리고 기존에 제안된 QOLB, LBP 기법과 성능을 비교하기 위해서 시뮬레이션을 수행하였다. 시뮬레이션 결과에 의하면 동기화에 관련된 성능은 하드웨어 구현성과 경제성을 고려할 때 QCX가 가장 우수한 것으로 평가된다. QCX는 QOLB나 LBP 보다 두가지 측면에서 매우 유리하다. 첫째, QCX는 기존의 메모리 구조나 캐쉬 구조의 변경없이 간단한 하드웨어(잠금캐쉬, 잠금변수번지, 큐 버퍼, 3개의 비교기, 그리고 스테이트 머신)로 구현이 가능하기 때문에 경제적이다. 반면에 QOLB와 LBP는 기존의 메모리 구조와 캐쉬 구조를 변경해야하고, 또한 매우 복잡한 캐쉬 구조를 갖고 있기 때문에 구현이 힘들고 비용이 증가한다. 둘째, QCX는 마이크로프로세서에서 제공하는 아토믹 명령어를 사용하여 설계되어 있기 때문에 일반성이 높아 많은 시스템에 적용할 수 있을 것이다. 반면에 QOLB 기법에서는 동기화 프리미티브인 QOLB를 시스템내에 구현할 수 있어야하고, LBP 기법에서는 캐쉬내에서 동기화 프로토콜을 처리할 수 있도록 구현해야 한다.

본 연구에서는 공유메모리를 사용하는 다중처리 시스템에 적합한 동기화 기법에 대해서 연구하였지만, 앞으로 연구 과제는 대규모 프로세서가 연결되는 병렬처리 구조의 상호연결망에서 효율적인 프로세서 동기화 기법을 연구하는 것이다.

참 고 문 헌

[1] T. E. Anderson, "The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol.1, no.1, pp.6-16, Jan. 1990.

[2] G. Graunke, S. Thakkar, "Synchronization Algorithms for Shared-Memory Multiprocessors," *IEEE Computer*, pp.60-68, June 1990.

[3] L. Lamport, "A Fast Mutual Exclusion Algorithm," *ACM Transaction Computer Systems*, vol.5, no.1, pp.1-11, 1987.

[4] J. L. Baer, et al., On Synchronization Patterns in Parallel Programs, Tech. Report No.91-04-01, Univ. of Washington, Apr. 1991.

[5] J. R. Goodman, M. K. Vernon, and P. J. Woest, "Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessor," *Proc. of Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS III)*, pp.64-75, Apr. 1989.

[6] P. J. Woest and J. R. Goodman, "An Analysis of Shared-Memory Synchronization Mechanisms," *Shared Memory Multiprocessing*, The MIP Press, pp.407-436, 1992.

[7] J. Lee and U. Ramachandran, "Synchronization with Multiprocessor Caches," *Proc. of 17th Int'l Symposium on Computer Architecture*, pp.27-37, May 1990.

[8] TICOM : Technical Overview Rev. 1.1, 한국전자통신연구소, Nov. 1988.

[9] Encore Computer Co., *Multimax Technical Summary*, 1987.

[10] T. Lovette, S. Thakkar, "The Symmetry Multiprocessor System," *Proc. of Int'l Conf. parallel Processing*, vol.1, pp. 303-310, Aug. 1988.

[11] Intel Semi. Co., *Pentium Processor User's Manual, Vol.1:Pentium Processor Data Book*, 1993.
[12] L. Rudolph and Z. Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," *Proc. of 11th Int'l Symposium on Computer Architecture*, pp.340-347, Jun. 1984.
[13] A. Gottlieb, et al., "The NYU Ultracomputer-Designing an MIMD, Shared Memory Parallel Machine," *IEEE Trans. on Computers*, pp.175-189, Feb. 1983.
[14] B. Smith, "Architecture and Applications for the HEP Multiprocessor Computer System," *SPIE, Vol.298 Real-Time Signal Processing IV*, pp.241-248, 1981.
[15] J. Archibald, J. L. Baer, "An Evaluation of Cache Coherence Solutions in Shared-bus Multiprocessors," Univ. of Washington Seattle, Technical Report 85-10-05, Oct. 1985.

저 자 소 개

尹 碩 漢(正會員)
1954년 6월 16일생. 1977년 고려대학교 전자공학과 졸업. 1986년 한국과학기술원 전산학과 졸업(석사). 1995년 고려대학교 전자공학과 졸업(박사). 1977년 - 1985년 한국전자기술연구소 선임연구원. 1985년 - 현재 한국전자통신연구소 책임연구원 프로세서연구실 실장. 주관심분야는 컴퓨터구조 및 병렬처리구조임.

金 惠 鎭(正會員) 제 29권 A편 제 9호 참조
현재 고려대학교 교수

元 鐵 虎(正會員)
1962년 11월 25일생. 1985년 한국항공대학교 항공전자공학과 졸업. 1987년 한구과학기술원 전기 및 전자 공학과 졸업(석사). 1987년 현재 한국전자통신연구소 선임연구원 프로세서연구실 근무. 주관심분야는 캐쉬 메모리 및 컴퓨터 구조임.