다중버스 다중프로세서 시스템을 위한 버스 중재 방식의 성능 분석*

Performance Analysis of Bus Arbitration Schemes for Multiple-bus Multiprocessor System

김 종 현**
Jong-Hyun Kim

Abstract

In a multiple-bus multiprocessor system in which processors and memory modules are interconnected through system buses, time delay due to bus contention degrades system performance. In order to reduce such a problem, an optimal bus arbitration scheme and its hardware are neccessary. In this study, performances of four arbitration schemes are analyzed and compared: fixed-priority, equal-priority, rotating-priority and round-robin priority schemes.

For the study, the software simulator of a multiple-bus multiprocessor system is developed by using SLAM Ⅱ. Simulation results show that, when memory sccesses are evenly distributed to all memory modules, round-robin priority scheme provides the best performance. But, when a hot spot exists, the use of the fixed-priority scheme results in the shortest access time.

1. 서 론

최근 대부분의 고성능 컴퓨터에서는 다중프로세서 구조(multiprocessor structure)가 채택되고 있다. 이와 같이 한 시스템내에 여러 개의 프로세서들이 존재하는 시스템의 설계에서 가장 중요한 것은 기억장치와 프로세서들 간의 상호연결 방식이다. 상호연결 구조에는 단일 시분할 버스(single time-shared bus)로부터 완전연결 크로스바(fully-connected crossbar)에 이르기까지 매우 다양하다[1]. 이들 중에서 단일 시분할 버스가 가장 간단한 연결 방식으로써, 최근 개발된 대부분의 슈퍼미니급 컴퓨터시스템들에서 채택되고 있다. 그러나 단일 버스 시스템은 여러개의 프로세서와 기억장치 모듈들이 한개의 버스를 공유함으로써 시스템의 규모가 커질수록 버스가 시스템 성능의 병목이 되기 때문에 시스템 성능이 급격히 감소하게 된다. 반면에 크로스바 스위치는 높은 throughput을 제공하지만 구현이 복잡하고 비용이 많이 든다. 크로스바 스위치를 사용하면서 상용화된 다중프로세서 시스템의 한 예로는 IPM

* 본 논문은 1992년도 한국과학재단 연구비 지원에 의하여 수행되었음.
** 연세대학교 원주캠퍼스 전산학과

社에서 개발한 IP-1이 있다. 그러나 N*N 크로스바는 O(N^2)의 복잡도를 감수해야 한다는 단점이 있다. 다단계 상호연결망(Multistage Interconnection Network)은 프로세서와 기억장치 간에 많은 일대일 연결을 동시에 제공하며, 크로스바에 비해 복잡도를 O(NlogN)으로 줄일 수 있다. 상용화된 다단계 상호연결망으로는 BBN Butterfly가 있고, Illinois 대학의 Cedar, IBM의 RP3, Purdue 대학의 PASM 등과 같은 실험적 시스템에서 연결망으로 채택되고 있다. 다단계 상호연결망은 오류회복 기능이 약하다는 단점이 있다[1].

상호연결망들의 장점을 살리고 단점들을 보안하면서 제안된 것이 다중버스 시스템(multiple-bus system)이다[2]-[5]. 다중버스 시스템은 P개의 프로세서들과 M개의 기억장치 모듈들이 B개의 버스를 통하여 연결되어 있으며, 기억장치 모듈들은 프로세서들에 의하여 공유되는 구조를 가지고 있다. 이 구조는 버스 용량(bus capacity), 시스템의 신뢰도 및 결합 허용도를 증가시켜 주며, 프로세서의 수가 기억장치의 수보다 적은 시스템 환경에서 throughput과 비용이 단일 버스와 크로스바의 중간임이 밝혀졌다[10]. 그러나 다중버스 시스템에서도 기억장치 액세스를 원하는 프로세서의 수가 많은 경우에는 버스 경합으로 인한 시간 지연이 발생되어 시스템 성능을 저하시키게 된다. 이러한 버스 경합으로 인한 시스템의 성능 저하를 줄이기 위해서는 최적의 버스 중재 방식과 그에 대한 하드웨어 구현이 필요하다[6, 7].

다중버스 다중프로세서 시스템의 버스 중재 기능은 다음과 같은 두가지 중재기에 의하여 이루어진다[8, 9].

- 1-of-N arbiter: 여러개의 프로세서들이 동시에 공유 기억장치를 액세스하려고 할 때, 그들 중에서 한개를 선택하는 회로.
- B-of-M arbiter: 선택된 프로세서에게 버스를 할당하는 회로.

버스 중재에는 여러가지 방법이 있으며, 이들은 각각 장단점을 가지고 있고 작업의 특성에 따라 성능에 미치는 영향이 서로 다르다[12, 13]. Guibaly[12]는 프로세서들이 한개의 버스를 공유하는 단일-버스 다중프로세서 시스템에서 버스 중재 방식들이 시스템 성능에 미치는 영향을 시뮬레이션을 이용하여 분석하였다. 또한 Yang[13]은 다중-버스 다중프로세서 시스템을 위한 버스 중재 방식들의 성능을 비교하였으나, 확률적 모델(probabilistic model)을 이용하였기 때문에 많은 가정들이 전제되어 분석에 한계가 있었다.

본 연구에서는 이들의 연구에서 고려된 다음과 같은 대표적인 버스 중재 방식들이 다중버스 시스템의 성능에 미치는 영향을 시뮬레이션을 이용하여 분석하였다. 특히, 이 연구에서는 실제 시스템에서 빈번히 발생하는 상황으로서 특정 기억장치 모듈에 액세스가 집중되는 hot spot 현상이 발생하는 경우에 대한 분석도 포함하였다.

- 고정 우선순위(fixed priority) 방식
- 동등 우선순위(equdl priority) 방식
- 회전 우선순위(rotating priority) 방식
- 라운드-로빈 우선순위(round-robin priority) 방식

고정 우선순위 중재 방식에서는 각 프로세서들이 고정된 우선 순위를 가지고 있다. 이 순위는 일반적으로 프로세서들이 버스에 접속된 순서에 의하여 정해지며, 하드웨어에 의하여 고정되므로 시스템이 동작하는 동안에 변하지 않는다. 이 방식은 특정 프로세서들이 다른 프로세서들보다 우선적으로 버스를 사용할 수 있도록 해야 할 경우에 유용하다.

그러나 대부분의 다중프로세서 시스템은 대칭적 시스템(symmetric system)으로서 모든 프로세서들에게 균등하게 작업들이 할당되므로 공유자원의 할당도 공정(fair)해야 한다. 이에 따라 설계된 방식이 동등 우선순위 방식이다. 이 방식에서는 시스템내의 모든 프로세서들이 각 중재 사이클 동안에 시스템 버스를 할당받을 수 있는 기회가 균등하게 주어진다. 예를 들어, B개의 버스를 가진 시스템에서 중재 사이클이 시작되는 시점에 k개의 프로세서들이 버스를 요구하였다면, 각 프로세서들이 버스 사용권을 획득할 수 있는 확율은 B/k(단, K 〉 B)가 된다.

동등 우선순위 방식을 사용하더라도 실제로는 모든 프로세서들이 동등한 우선순위를 가지도록 하는 것은 불가능하다. 왜냐하면 엄밀한 의미에서의 공정한 프로토콜은 N개의 프로세서들을 가진 시스템에서 버스 사용을 원하는 모든 프로세서들이 N 버스 사이클내에 한번씩 버스를 사용할 수 있어야 하기 때문이다. 이에 따라 설계된 방식이 회전 우선순위 방식이다. 이 방식에서는 각 프로세서의 우선순위가 시간에 따라 변한다. 즉, 버스 사이클이 끝나면 각 프로세서의 우선순위가 1씩 높아지고, 이 동작이 우선순위의 최고값이 될 때까지 반복하며, 다시 최하값으로 되돌아가서 사이클을 반복한다.

그러나 회전 우선순위 방식도 완전한 공정성을 제공하지는 못하는데, 특히 각 프로세서의 작업부하가 동일하게 분포되어 있지 않은 경우에 그러하다. 작업부하가 많이 걸리는 프로세서도 우선순위가 높아질 때까지는 버스 사용권을 획득하기가 어렵기 때문이다. 이에 따른 보완을 위하여 제안된 것이 라운드-로빈 우선순위 방식이다. 이 방식에서도 프로세서의 우선순위는 시간에 따라 계속 변하지만, 우선순위가 사이클마다 1씩 변하지 않고 시스템 부하에 따라 동적으로(dynamically) 변한다. 각 중재 사이클이 끝날 때마다 바로 전 사이클에서 버스를 사용한 프로세서의 다음 프로세서에게 가장 높은 우선순위를 주는 것이다. 따라서 각 프로세서의 우선순위는 매 사이클마다 시스템 부하에 따라 높아진다.

이러한 여러가지 중재 방식들의 성능을 비교하기 위하여는 모든 중재 회로가 시스템에 구현될 필요가 있는데, 이 과정을 실제 시스템에서 수행하는 것은 많은 시간과 비용이 소요된다. 따라서, 본 연구에서는 융통성이 높고 비용이 적게 드는 소프트웨어 시뮬레이션을 이용하여 각 중재 방식들의 성능을 비교하였다. 제2장에서는 다중버스 시스템의 구조에 대하여 설명하였고, 제3장에서는 소프트웨어 시뮬레이터의 개발에 대하여 기술하였다. 제4장에서는 시뮬레이션 및 분석의 결과를 설명하였고 마지막으로 제5장에서는 전체적인 결론을 정리하였다.

2. 시스템 모델

본 연구에서 선택한 다중버스 시스템의 구조는 〈그림 1〉과 같다. N개의 프로세서들은 B개의 버스, B_1, B_2…B_B에 의해 공유 기억장치와 연결되어 있다. 공유 기억장치는 인터리브된 기억장치 모듈, M_1, M_2…M_M로 구성되어 있으므로 여러 프로세서들이 동시에 공유 기억장치를 액세스할 수 있다. 프로세서가 기억장치 모듈을 액세스하려면 B개의 버스들 중의 하나를 할당받아야 한다. 〈그림 1〉에서 볼 수 있듯이, 각 프로세서와 기억장치 사이의 경로는 여러 개가 존재하므로 다른 프로세서들이 버스들을 사용하는 동안에도 사용 가능한 버스가 남아 있는 경우에는 기억장치 액세스가 가능해진다. 이에 따라 전체 버스 대역폭도 상승한다. 또한 어떤 버스에 동작상의 오류가 발생한 경우에는 다른 버스들을 사용하여 시스템이 계속 동작할 수 있으므로 결함 허용도도 높아진다.

이 구조에서 여러 개의 프로세서들이 동시에 공유 기억장치 모듈을 액세스하려는 경우에는 사용 가능한 버스들을 프로세서에 할당하는 기법이 필요하다. 특히 프로세서의 수가 버스의 수보다 많은 경우에는 버스의 중재 방식이 시스템 성능에 큰 영향을 주게 된다. 버스 중재에 소요되는 시간에 따른 시스템의 성능 저하를 방지하기 위해서 일반적으로 일반적으로 중재기는 하드웨어로 구현된다. 하드웨어 중재기는 다중버스 연결망의 복잡도(complexity)를 다소 증가시키게 된다.

이와 같은 다중버스 시스템에서는 프로세서들이 기억장치와 버스를 공유함으로써 발생되는 충돌(conflict)이 있다. 첫째 여러 프로세서들이 동시에 동일한 기억장치 모듈을 액세스하려고 할 때 발생되는 기억장치 충돌, 둘째, 프로세서에 의해 발생되는 기억장치 요구에 비해 버스의 수가 충분하지 않을 때 발생되는 버스 경합(bus contention)이 있다.

공유자원(기억장치와 버스) 사용에서 발생되는 충돌에 대한 중재는 다음과 같은 두 단계로 이루어진다. 각 기억장치 모듈당 한개씩 전체 M개의 1-of-N 중재기들이 기억장치 충돌을 중재한다. 여기서 M은 기억장치 모듈의 수이고, N은 프로세서의 전체 갯수이다. 1-on-N 중재기는 자신이 담당한 기억장치 큐로부터 버스 중재 프로토콜에 의해 한개의 프로세서를 선택한다. 다음 단계로서, 한개의 B-of-N 중재기가 버스 경합을 중재하는데, 여기서 B는 사용 가능한 버스의 최대 갯수, M은 그 순간의 기억장치 액세스 요구의 수이다.

본 연구에서는 제1장에서 설명한 네가지 중재 방식들이 위의 두가지 중재기에서 적용될 때의 시스템 성능(기억장치 액세스 처리율)을 분석하는 것이다. 이 분석에서는 범위를 정하기 위하여 다음과 같은 가정을 두었다[3, 4, 5].

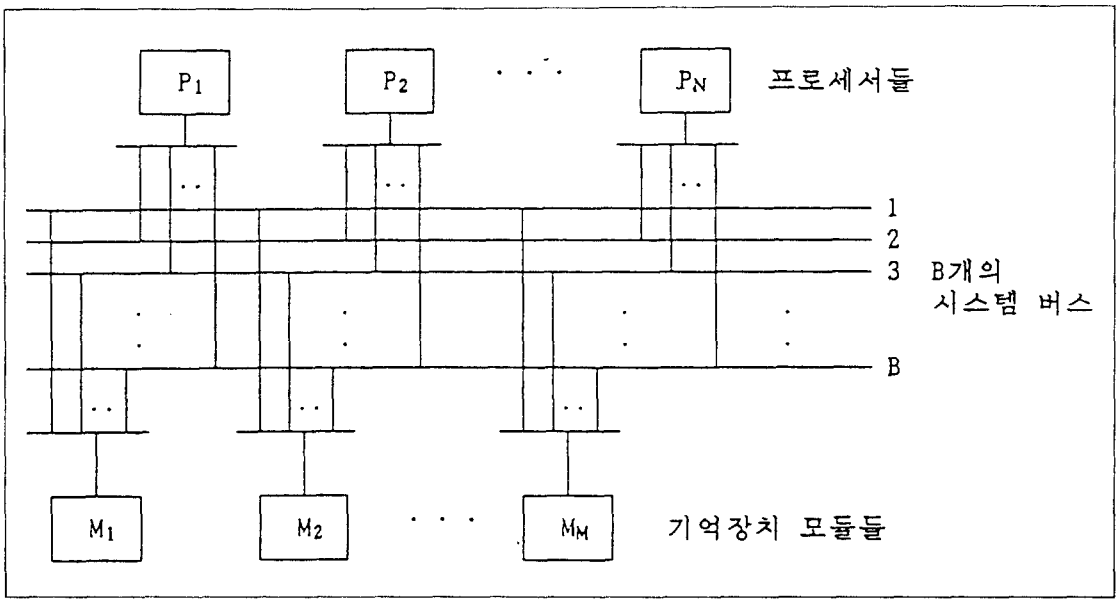
(1) 각 프로세서로부터 발생되는 기억장치 요구는 프로세서들 간에 독립적이다.

(2) 주소 해독(address decoding) 시간은 중재 시간에 포함되는 것으로 가정한다.

(3) 버스 중재기는 읽기와 쓰기 요구를 동일하게 취급하며, 이들은 모두 같은 서비스 시간 분포와 평균값을 갖는다.

(4) 모든 기억장치 모듈에 대한 서비스 시간은 독립적이고 동일한 분포를 갖는다.

(5) 주소 해독이 끝난 후에 기억장치 요구는 해당 기억



〈그림 1〉 다중버스 시스템의 구조

장치 큐에 들어가며, 요구된 기억장치 모듈이 idle 하고 사용 가능한 버스가 있으면 큐에서 즉시 제거된다. 따라서 이 경우에는 큐잉 지연 시간 없이 경로가 직접 형성된 결과가 된다.

(6) 특정 기억장치 모듈에 대한 요구가 즉시 서비스를 받을 수 없는 경우에는 기억장치 서비스가 끝날 때까지 큐에서 대기한다.

(7) 기억장치 엑세스가 완료되면, 기억장치 모듈과 버스는 지연 시간 없이 해체된다.

3. 소프트웨어 시뮬레이터의 개발

본 연구에서 개발한 시뮬레이터는 제2장에서 제시된 시스템 모델에 따라 시뮬레이션 전용 언어인 SLAM Ⅱ를 이용하여 구현되었다[14, 15]. 즉, 이산-사건 시뮬레이션 기법이 사용되었으므로, 시뮬레이터는 시스템의 주요 요소들의 동작과 기능들을 시뮬레이트하는 사건 루틴(event routine)들로 구성되어 있다. 시스템 모델의 동작과 기능에 따라 분류되어 작성된 사건 루틴들은 다음과 같다.

(1) SCP 사건루틴

SCP(System Control Process) 사건 루틴에서는 시스템의 주요 요소들(프로세서, 기억장치, 버스)의 갯수를 결정하고, 시스템 플래그들을 초기화한다. 또한, 시간 0에 PROC 사건이 프로세서 수만큼 실행되도록 스케쥴한다.

(2) PROC 사건 루틴

프로세서의 동작을 시뮬레이트하는 사건으로서, 난수 발생기에 의해서 생성되는 시간 간격에 따라 기억장치 액세스 요구를 발생하고 해당 프로세서의 BUSY 프래그를 세트시킨다. 또한 기억장치 요구를 발생한 즉시 주소 해독이 시작되도록 ADRDEC 사건을 스케쥴한다.

(3) ADRDEC 사건 루틴

공유 기억장치의 액세스 요구가 발생했을 때 기억장치 요구의 주소를 해독하는 루틴이다. 정해진 해독 시간이 지난 후에 ENDDEC(End of ARDDEC)사건이 실행되도록 스케쥴한다. 주소 해독 시간은 중재 시간에 포함시켰기 때문에 0ns로 세트하였다. 또한 주소 해독이 시작될 때 프로세서 BUSY 프래그를 0으로 리셋하여 프로세서가 일정 시간 후에 다음 기억장치 요구를 발생할 수 있게 하였다.

(4) ENDDEC 사건 루틴

해당 기억장치 큐에 기억장치 요구를 넣고, 즉시(스케쥴링 시간=0) ARBIT 사건 루틴이 실행되도록 스케쥴한다. 이것은 큐에 들어간 직후에 지연 시간 없이 중재가 일어남을 의미한다.

(5) ARBTR 사건 루틴

중재기의 동작을 시뮬레이트하는 사건으로서, 중재 시간으로 가정된 80ns 후에 ENDARB(End of ARBTR) 사건이 실행되도록 스케쥴한다.

(6) ENDARB 사건 루틴

중재기의 기능을 실제 수행하는 루틴으로서 1-on-N 중재기와 B-of-M 중재기를 모두 구현한다. 중재기 루틴에서는 먼저 사용 가능한 버스가 있는 지를 확인한다. 이때 버스를 찾는 방법은 각 버스 선에 번호를 지정하고, 번호가 작은 것부터 사용 여부를 타진한다. 사용 가능한 버스의 조사는 최악의 경우에 버스의 갯수만큼(버스가 모두 사용 중일 때는 시스템 내의 버스 수 만큼 조사한 후에 마치게 되므로) 반복하게 된다. 사용 가능한 첫번째 버스를 확보한 후에는 중재 프로토콜에 따라 적절한 프로세서 큐를 선택한다.

이 과정에서, 현재 조사하는 프로세서 큐에 대기하고 있는 요구가 있고 동시에 그 요구가 원하는 기억장치가 액세스 가능한 경우에는 그 큐로부터 요구를 읽어서 기억장치 액세스를 시작한다. 만약 프로세서 큐에 요구가 없거나, 있더라도 해당 기억장치가 다른 프로세서에 의해서 사용되고 있는 경우에는 다음 프로세서 큐를 검사하게 된다. 이와 같이, 기억장치 요구의 선택이 버스를 확보한 후에 이루어지지만, 이미 특정 중재 프로토콜에 근거하여 기억장치 요구가 프로세서 큐에 들어간 상태이므로 사실상 프로세서들에 대한 선택이 먼저 이루어졌다고 할 수 있다. 즉, 프로세서의 요구가 큐에 들어가는 순간에 기억장치 충돌은 해결되고(1-of-N 중재기), 버스 경합을 해결하여 버스를 확보한 후(B-of-M 중재기)에 기억장치 액세스가 시작되는 것이다. 중재가 모두 끝난 후에는 지연 시간 없이 기억장치 액세스를 시작할 수 있도록 즉시 MACC 사건을 스케쥴한다. 이 과정은 사용 가능한 버스가 없을 때까지 계속된다.

(7) MACC 사건 루틴

기억장치 액세스 동작을 시뮬레이트하는 사건으로서, 액세스 시간으로 가정된 320ns 후에 ENDMAC(End of MACC) 사건을 스케쥴한다.

(8) EMACC 사건 루틴

기억장치 액세스 동작을 시뮬레이트하는 사건으로서, 사용된 기억장치 모듈과 버스를 free 상태로 바꾸어준다. 서비스가 완료되었으므로 큐에서 프로세서 번호를 제거한다. 또한, 서비스를 받은 프로세서가 다음 요구를 위하여 활성화되어야 하므로 즉시 PROC 사건을 스케쥴한다. 그리고 만약 시스템내에 서비스를 받지 못하고 큐에서 기다리는 요구가 있으면 다시 중재 동작을 수행하도록 ARBIT 사건도 스케쥴한다.

각 사건 루틴들 간의 파라메터 전달은 ATRIB 어레이를 통하여 이루어지며, 각 ATRIB의 내용은 다음과 같다.

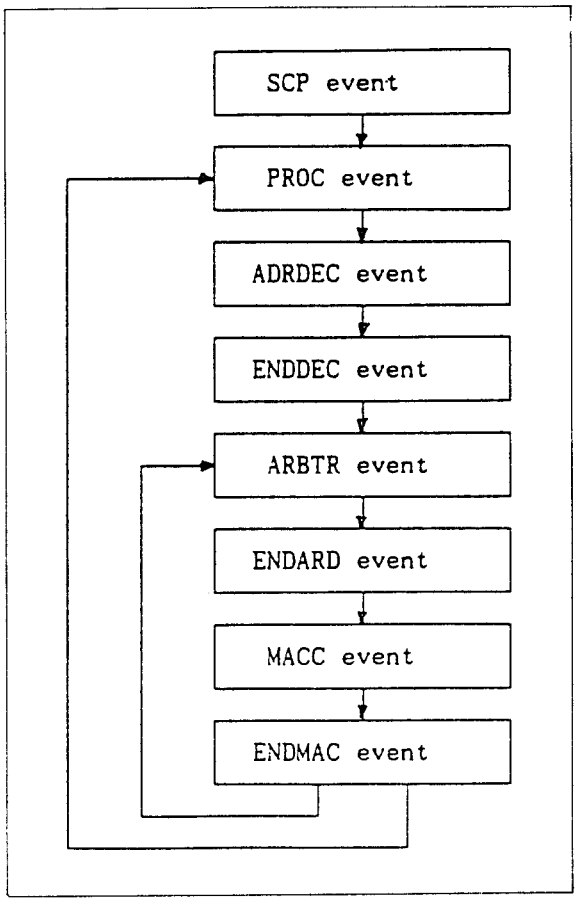
ARTIB(1) : 사건 발생 시간
ARTIB(2) : 기억장치 액세스를 요구한 프로세서 번호
ARTIB(3) : 요구한 기억장치 모듈의 번호
ARTIB(4) : 프로세서가 할당받은 버스의 번호
ARTIB(5) : 프로세서가 기억장치를 요구한 시간

사건 루틴들이 관련된 다른 사건 루틴들을 스케쥴하는 관계는 〈그림 2〉와 같다.

4. 시뮬레이션 및 결과 분석

시스템 버스는 동기식(synchronous) 혹은 비동기식(asynchronous)으로 동작한다. 동기식 시스템에서는 모든 프로세서-기억장치간의 동작이 하나의 공통 클럭(global clock)에 동기되어 일어난다. 비동기식 시스템에서는 프로세서가 임의의 시간에 기억장치 요구(memory request)를 발생시키며, 기억장치 액세스 시간은 기억장치 혹은 버스의 경합에 따른 지연 시간 때문에 상황에 따라 변하게 된다.

본 실험에서는 프로세서와 기억장치를 각각 8개로 가정하고, 버스의 수는 다양하게 변화시키면서 각 버스 중재 프로토콜의 성능과 특성을 측정하였다. 시뮬레이션에서 사용한 시간 변수들은 프로세서가 발생한 기억장치 주소의 해독(decode)시간과 중재에 걸리는 시간을 합하여 80nsec, 기억장치 서비스 시간은 320nsec로 가정하였다.



〈그림 2〉 사건 루틴들 간의 스케줄링 관계

이러한 분석에서 구분되어야 할 시스템 구성으로는 각 프로세서가 캐쉬를 가지는 경우와 가지지 않는 경우이다. 본 연구에서는 기억장치 요구의 빈도를 높이기 위하여 캐쉬가 없는 경우를 고려하였다. 캐쉬가 없는 경우에 프로세서는 기억장치 요구를 발생한 순간부터 대기(wait) 상태에 들어가며, 기억장치로부터 데이타가 도착한 후에 다시 활성화되어 다음 기억장치 요구를 발생한다. 다음 기억장치 요구를 발생하는 것도 데이타 도착 즉시 발생하는 경우와 임의의 시간이 지난 후에 발생하는 경우가 있으나, 본 연구에서는 실제 시스템과 유사한 환경을 만들어 주기 위하여 두번째 경우만 고려하였다.

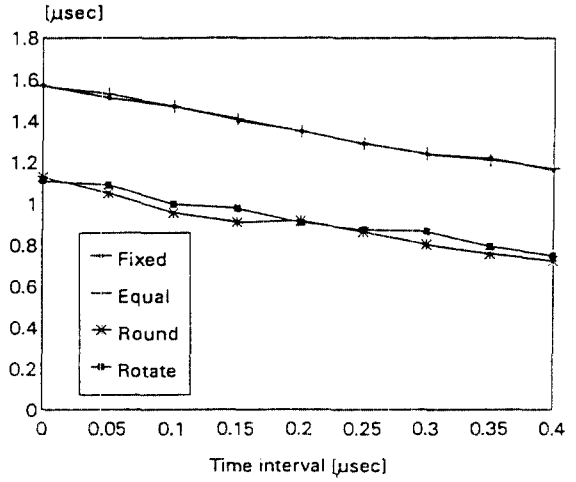
본 연구에서는 (1) 프로세서가 모든 기억장치 모듈들을 균등한 빈도로 액세스하는 경우와, (2) 특정 기억장치 모듈로 액세스 요구가 집중되어 hot spot이 존재하는 경우에 대한 분석을 하였다. 성능 비교의 척도로는 평균 기억장치 액세스 시간을 사용하였다. 여기서 기억장치 액세스 시간이란 기억장치 충돌이 발생하거나 모든 버스들이 busy일 때 기억장치 큐에서 대기하는 시간과 기억장치 서비스 시간을 합한 것이다.

4-1. 기억장치 모듈들이 균등하게 액세스되는 경우

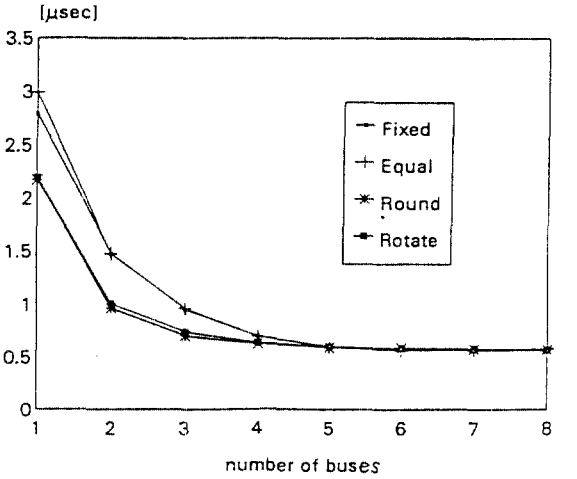
첫번째 실험은 프로세서들에 의하여 기억장치 모듈들이 액세스되는 빈도가 균등한 경우에 대하여 분석하였다. 버스는 회선 교환(circuit switching) 방식으로 동작하는 것으로 가정하였으며, 따라서 프로세서는 기억장치 액세스 요구를 발생한 때부터 기억장치 서비스가 완료될 때까지 idle 상태가 된다. 시뮬레이션에서 버스의 수는 1개부터 8개까지 변화시키고, 요구간 시간 간격(inter-request time interval)은 0에서 400nsec까지 50nsec 간격으로 변화시키면서 각각에 대한 성능을 측정하였다.

〈그림 3〉은 버스의 수가 2개로 고정된 상태에서 프로세서가 기억장치 액세스 요구를 발생시키는 시간 간격의 변화에 따른 평균 기억장치 액세스 시간을 보여주고 있다. 여기서 시간 간격이 0nsec이면, 프로세서가 이전의 요구에 대한 처리가 완료된 즉시 다음 요구를 발생하는 경우를 의미한다. 〈그림 3〉에서 보는 바와 같이, 요구간 시간 간격이 커지면(즉, 기억장치 요구 빈도가 낮아지면), 모든 프로토콜에서 평균 기억장치 액세스 시간이 줄어든다. 그 이유는 시간 간격이 짧은 경우에는 큐에서 대기하는 요구의 수가 증가하여 지연 시간이 길어지며, 요구의 빈도가 낮아지면 큐잉 지연이 줄어들기 때문에 평균 액세스 시간은 점차적으로 감소하는 것이다. 중재 방식들 간의 성능을 비교해보면, 라운드-로빈 방식의 경우에 기억장치 액세스 시간이 가장 짧아지며, 회전 우선순위 방식이 비슷한 결과를 보여주고 있다. 고정 우선순위 방식과 동등 우선순위 방식은 위의 두 방식보다 성능이 떨어지는 것을 알 수 있었다.

〈그림 4〉는 요구간 시간 간격이 일정할 때 버스 수의 변화에 따른 성능 비교의 결과를 보여주고 있다. 이 분석에서는 버스의 수를 1개부터 8개까지 변화시켰으며, 요구간 시간 간격은 기억장치 액세스가 완료된 100nsec 후에



〈그림 3〉 기억장치 액세스 요구간 시간 간격의 변화에 따른 평균 액세스 시간



〈그림 4〉 버스 수의 변화에 따른 평균 기억장치 액세스 시간

발생하는 경우로 고정시켰다. 전체적으로 보면, 버스의 수가 증가함에 따라 기억장치 액세스 시간이 줄어든다. 그러나 그림에서 보는 바와 같이, 버스의 수가 어느 정도 증가하면 성능에는 더 이상 차이가 나타나지 않음을 알 수 있다. 그 이유는 버스의 수가 많을 때는 기억장치 액세스 시간의 지연 요인이 버스 경합이 아니라 기억장치 충돌(memory conflict)이 되기 때문이다. 또한, 버스의 수가 적을 때는 라운드-로빈 중재 방식이 가장 우수하며, 동등 우선순위 방식이 가장 성능이 낮은 것으로 나타났다. 그러나 중재 방식에 따른 이러한 성능상의 차이는 버스의 수가 증가함에 따라 감소하여, 버스 수가 5개 이상일 때는 네가지 중재 방식들의 성능 차이가 거의 없다. 그 이유는 버스의 수가 충분할 때는 버스 경합에 의한 시간 지연은 거의 없어지고 기억장치 충돌에 의한 지연만 발생하기 때문이다. 즉, 기억장치 모듈들에 대한 액세스 빈도가 동등하므로 액세스 지연 시간은 버스의 수와 중재 방식과는 무관함을 알 수 있다.

4-2. 기억장치 모듈들에 대한 액세스 빈도가 불균등한 경우

이 실험은 프로세서로부터 발생되는 기억장치 요구들의 1/2 정도가 특정 기억장치 모듈로 집중되는, 즉, 극심한 hot spot이 존재하는 경우에 대한 중재 방식의 성능을 분석 비교하였다. 이것은 프로세서들에 의하여 공유되는 프로그램 코드와 데이타들의많은 부분이 특정 기억장치 모듈에 저장되어 있을 때 발생하는 실제의 상황을 반영하는 것이다. 여기서 hot spot 집중율을 1/2로 높게 정한 이유는 hot spot이 발생하는 상황에서 각 버스 중재 방식의 성능 차이를 분명하게 구분시키기 위한 것이다.

본 연구에서는 8개의 기억장치 모듈들 중에서 한개의 기억장치 모듈이 hot spot이 되는 경우를 고려하였으며, 이에 대한 버스 중재 방식의 성능을 비교하기 위하여 hot spot 모듈의 위치를 변경하면서 실험하였다. 버스의 수는 2개, 4개 및 8개로 증가시키면서 평균 기억장치 액세스 시간을 측정하였다. 또한, 기억장치 요구들간의 시간 간격은 0부터 600ns까지 변경시켰다.

먼저, 첫번째 기억장치 모듈(#1)이 hot spot인 경우에 대하여 측정된 결과는 〈표 1〉과 같다. 결과에 따르면, 고정 우선순위 방식이 가장 성능이 좋은 것으로 나타났는데, 그 이유는 hot spot인 기억장치 모듈이 가장 우선순위가 높으므로 많은 요구들을 신속히 처리해줄 수 있기 때문이다. 특기할 사항은 버스의 수가 늘어나도 액세스 시간은 줄어들지 않는데, 그 이유는 대부분의 요구가 한 기억장치 모듈로 집중되기 때문에 지연 요인은 기억장치 충돌이고, 버스의 수는 별 영향을 미치지 못한다는 것을 알

수 있다.

〈표 1〉 첫번째 기억장치 모듈이 hot spot인 경우

(a) 버스의 수=2

중재방식	요구간 시간간격[μsec]						
	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Fixed	1.62	1.66	1.54	1.44	1.37	1.34	1.22
Equal	2.67	2.48	2.27	2.18	1.99	1.82	1.69
Round	2.82	2.57	2.35	2.12	1.99	1.85	1.63
Rotate	2.75	2.56	2.31	2.14	1.99	1.86	1.62

(b) 버스의 수=4

중재방식	요구간 시간간격[μsec]						
	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Fixed	1.66	1.67	1.55	1.49	1.38	1.39	1.34
Equal	2.76	2.55	2.14	2.12	1.90	1.77	1.61
Round	2.76	2.54	2.30	2.12	1.90	1.88	1.72
Rotate	2.72	2.55	2.28	2.16	1.89	1.82	1.66

(c) 버스의 수=4

중재방식	요구간 시간간격[μsec]						
	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Fixed	1.68	1.66	1.58	1.43	1.35	1.38	1.25
Equal	2.71	2.31	2.28	2.00	1.99	1.87	1.54
Round	2.84	2.55	2.33	2.16	1.85	1.88	1.54
Rotate	2.79	2.53	2.32	2.11	1.88	1.89	1.51

두번째 실험은 hot spot을 세번째와 마지막 기억장치 모듈로 변경시키면서 각 중재 방식에 대한 평균 기억장치 액세스 시간을 측정하였다. 첫번째 실험에서 버스의 수가 성능에 거의 영향을 미치지 않는 것이 확인되었으므로, 이 실험에서는 버스의 수를 2개로 고정시켰다. 실험 결과는 〈표 2〉 및 〈표 3〉과 같다. 결과에 따르면, 두 경우에 모두 고정 우선순위 방식이 가장 성능이 좋은 것으로 나타났다. 또한, 고정 우선순위 방식에서 hot spot이 세번째 기억장치 모듈인 경우 보다 마지막 모듈인 경우에 성능이 가장 낮은 것으로 나타났고, 다른 중재 방식에서는 차이가 없었다.

Hot spot 모듈의 우선순위가 낮기 때문에 평균 액세스 시간이 길어질 것으로 예상하였으나, 대부분의 기억장치 요구들이 한 기억장치에 집중됨으로써 그 기억장치 큐에서 대기하고 있으며 프로세서들은 그 요구에 대한 서비스가 종료될 때까지 다음 요구를 발생하지 않는다. 따라서 요구 발생 수가 적어져서 전체적인 서비스 수는 줄어든다. 그러나 상대적으로 다른 기억장치 모듈에 대한 요구의 수가 적기 때문에 평균 액세스 시간은 그러한 현상이 가장 심한 고정 우선순위 방식의 경우가 가장 짧게 나타나는 것이다.

〈표 2〉 세번째 기억장치 모듈이 hot spot인 경우

중재방식	요구간 시간간격[μsec]						
	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Fixed	1.65	1.64	1.54	1.43	1.40	1.36	1.25
Equal	2.66	2.51	2.30	2.10	1.98	1.85	1.61
Round	2.82	2.48	2.32	2.15	2.00	1.81	1.64
Rotate	2.67	2.51	2.32	2.22	1.97	1.83	1.63

〈표 3〉 세번째 기억장치 모듈이 hot spot인 경우

중재방식	요구간 시간간격[μsec]						
	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Fixed	1.67	1.66	1.61	1.56	1.47	1.42	1.34
Equal	2.64	2.49	2.31	2.18	1.99	1.87	1.60
Round	2.80	2.45	2.33	2.21	1.99	1.80	1.66
Rotate	2.79	2.51	2.38	2.25	1.99	1.80	1.61

5. 결론

프로세서들과 기억장치 모듈들이 여러 개의 시스템 버스들을 통하여 연결된 다중버스 구조에서는 버스 경합으로 인한 시간 지연이 발생되어 시스템 성능을 저하시키게 된다. 이러한 성능 저하를 줄이기 위해서는 최적의 버스 중재 방식과 그에 대한 하드웨어 설계가 필요하다. 버스

중재에는 여러가지 방법이 있으며, 이들은 각각 장단점을 가지고 있고 작업의 특성에 따라 성능에 미치는 영향이 서로 다르다. 본 연구에서는 대표적인 네가지 버스중재 방식들의 특성을 분석하고, 각 방식들에 대한 시스템 성능을 비교하였다.

중재 방식들의 성능을 비교하기 위하여는 모든 중재 회로들이 시스템에 구현되어야 하는데, 이 과정을 실제 시스템에서 수행하는 것은 많은 시간과 비용이 소요된다. 따라서, 본 연구에서는 융통성이 높고 비용이 적게드는 소프트웨어 시뮬레이션을 이용하여 각 방식들의 성능을 비교하였다. 연구의 결과에 따르면, 본 연구에서 사용한 시뮬레이션 방법은 다양한 시스템 환경(버스의 수, 기억장치 요구들 간의 시간 간격, 기억장치 액세스 패턴)에서의 각 버스 중재 방식의 성능을 측정할 수 있음을 입증하였다.

시뮬레이션에서는 프로세서와 기억장치 모듈이 각각 8개씩인 시스템에서, 버스의 수를 1개부터 8개까지 변화시키고, 기억장치 요구들간의 시간 간격도 0에서 400nsec까지 50nsec 간격으로 변화시킬 수 있도록 하였다. 성능 비교의 결과에 따르면, 라운드-로빈 방식을 이용하면 기억장치 액세스 시간이 가장 짧아지며, 회전 우선순위 방식도 비슷한 결과를 보여주고 있다. 고정 우선순위 방식과 동등 우선순위 방식은 위의 두 방식보다 성능이 떨어지는 것을 알 수 있었다. 그러나 중재 방식에 따른 이러한 차이는 버스의 수가 증가함에 따라 감소하여, 버스 수가 5개 이상일 때는 네가지 중재 방식들의 성능 차이가 거의 없었다. 그 이유는 버스가 충분하기 때문에 기억장치 충돌에 의한 지연만 발생하며, 지연 시간은 버스의 수 및 중재 방식과는 무관함을 알 수 있다.

기억장치 모듈들에 대한 액세스 빈도가 불균등한 경우에 대한 분석에서는 8개의 기억장치 모듈들 중에서 한개의 기억장치 모듈이 hot spot이 되는 경우를 고려하였으며, 이에 대한 버스 중재 방식의 성능을 비교하기 위하여 Hot spot 모듈의 위치를 변경하였다. Hot spot 집중율은 모든 실험에서 1/2로 고정시켰고, 버스의 수는 2개, 4개 및 8개로 증가시키면서 평균 기억장치 액세스 시간을 측정하였다.

결과에 따르면, 고정 우선순위 방식이 가장 성능이 좋은 것으로 나타났는데, 그 이유는 hot spot인 기억장치 모듈이 가장 우선순위가 높으므로 많은 요구들을 신속히 처리해줄 수 있기 때문이다. 특기할 사항은 버스의 수가 늘어나도 액세스 시간은 줄어들지 않는데, 그 이유는 대부분의 요구가 한 기억장치 모듈로 집중되기 때문에 지연 요인은 기억장치 충돌이고, 버스의 수는 별 영향을 미치지 못하기 때문이다.

[참 고 문 헌]

[1] T. Feng, "A Survey of Interconnection Networks," Computer, Dec. 1981, pp. 12-27.

[2] T. Mudge, J. Hayes, and D. Winsor, "Multiple Bus Architectures," IEEE Computer, June 1987, pp. 42-48.

[3] T. Lang, M. Valero, and I. Alegre, "Bandwidth of Crossbar and Multiple-Bus Connetions for Multiprocessors," IEEE Trans. on Computers, vol.c-31, No.12, Dec. 1982, pp. 1227-1234.

[4] T.N. Mudge, J.P. Hayes, G.D. Buzzard, and D.C. Winsor, "Analysis of Multiple-bus Interconnection Networks," Journal of Parallel and Distributed Computing 3, 1986, pp. 328-343.

[5] Q. Yang, "Performance of Multiple-Bus Interconnections for Multiprocessors," Journal of Parallel and Distributed Computing 8, 1990, pp. 267-273.

[6] R.C. Pearce, J.A. Field, and W.D. Little, "Asynchronous Arbiter Module," IEEE Trans. on Computers, Sept. 1975, pp. 931-932.

[7] E. Petriu, "N-channel asynchronous arbiter resolves resource allocation conflicts," Computer Design, Aug. 1980, pp. 126-132.

[8] W. Plummer, "Asynchronous arbiters," IEEE Tran. on Computers, vol.C-21, Jan. 1972, pp.37-42.

[9] T. Lang and M. Valero, "M-Users B-Servers Arbiter for Multiple-Bus Multiprocessors," Microprocessing and Microprogramming 10, J. Euromicro, 1982, pp.11-18.

[10] C.R. Das and L.N. Bhuyan, "Bandwidth Availability of Multiple-Bus Multiprocessors," IEEE Trans. on Computers, vol.C-34, No.10, Oct. 1985, pp.918-926.

[11] W.T. Chen and J.P. Sheu, "Performance Analysis of Multiple Bus Interconnection Networks with Hierarchical Requesting Mode," IEEE Trans. on Computers, vol.

40, No.7, July 1991, pp.834-842.

[12] F. Guibaly, "Design and Analysis of Arbitration Protocols," IEEE Trans. on Computers, vol.38, No.2, Feb.1989, pp.161-171.

[13] Q.Yang, "Effects of Arbitration Protocols on the Performance of Multiple-bus Multiprocessors," 1991 Int. Conf. on Parallel Processing, pp.600-603.

[14] A.B. Pritsker, Introduction to Simulation and SLAM Ⅱ, A Halsted Press Book, John Wiley & Sons, New York, 1986.

[15] 김선의, 김종현, 한탁돈, "시뮬레이션을 이용한 다중 버스 다중프로세서 시스템의 성능분석," 한국정보과학회 논문지, 제20권 제7호, 1993년 7월, pp. 1005-1008.

저자소개

김종현

1976년 연세대학교 공과대학 전기공학과 졸업(공학사)
1981년 연세대학교 대학원 전기공학과 졸업(공학석사)
1988년 Arizona State Univ. 컴퓨터공학과 졸업(Ph.D)

1976년~1982년 국방과학연구소 연구원
1982년~1983년 금오공과대학 전자공학과 전임강사
1988년~1990년 한국전자통신연구소 실장
1990년~현재 연세대학교 원주캠퍼스 전산학과 부교수