論 文

HiPi 버스를 사용한 멀티프로세서 시스템에서 캐쉬 코히어런스 프로토콜의 성능 평가에 관한 연구

正會員 金 永 川* 正會員 姜 寅 坤* 正會員 黃 勝 郁** 正會員 崔 眞 圭***

A Study on the Performance Analysis of Cache Coherence Protocols in a Multiprocessor System Using HiPi Bus

Young Chon Kim,* In Kon Kang,* Seung Uk Hwang,** Jin Kyu Choi*** *Regular Members*

要 約

본 논문에서는 pended 프로토콜을 가지는 HiPi 버스와 다중 캐쉬 메모리를 사용하는 멀티프로세서 시스템을 기술하고, 캐쉬 코히어런스 프로토콜에 따라 프로세서의 효율 측면에서 시스템의 성능을 평가하였다. HiPi 버스는 ETRI에서 개발된 행정전산망용 주전산기인 TICOM II의 공유 버스로 사용되기 위하여 개발되었다. HiPi 버스는 고속의 데이타 전송 능력을 가지고 있으나, 캐쉬 간의 데이타 전송을 허용하지 못하는 단점을 가지고 있다. 캐쉬 간의 데이타 전송이 전체 시스템의 성능에 미치는 영향을 측정하고, HiPi 버스에 적합한 캐쉬 코히어런스 프로토콜을 선택하기 위하여 두가지 시뮬레이션을 실시하였다. 첫째, HiPi 버스를 사용하는 멀티프로세서 시스템에 다양한 캐쉬 코히어런스 프로토콜을 적용하고 시뮬레이션을 통하여 프로세서 효율에 따른 성능 분석을 실시하였다. 각각의 프로토콜은 상태 천이도로 나타내었으며, Markov 정적 상태도를 이용하여 각 상태의 확률 값을 구하였다. 각 상태의 확률은 시뮬레이션에서 입력 값으로 사용되었고, 모델링과 시뮬레이션은 SLAM II 심볼과 언어를 사용하였다. 둘째, 캐쉬 간의 데이타 전송을 갖는 HiPi 버스를 제안하였고, 제안된 HiPi 버스를 사용하는 멀티프로세서 시스템에 다양한 캐쉬 코히어런스 프로토콜을 적용하고 시뮬레이션을 통하여 프로세서 효율에 따른 성능 분석을 실시하였다. 고려된 캐쉬 코히어런스 프로토콜은 Write-through, Write-once, Berkely, Synapse, Illinois, Firefly, Dragon 이다.

ABSTRACT

In this paper, we describe a multiprocessor system using the HiPi bus with pended protocol and multiple cache memories, and evalute the performance of the multiprocessor system in terms of processor utilization for various cache coherence protocols. The HiPi bus is delveloped as the shared bus of TICOM II which is a main computer system to establish a nation-wide computing

*全北大學校 컴퓨터工學科
Dept. of Computer Eng., Chonbuk National Univ.
**韓國海洋大學校 制御計測工學科
***한남大學校 電子工學科
Dept. of Elec. Eng. Han-Nam Univ.
論文番號 : 93-7 (接受1992. 6. 2)

network in ETRI. The HiPi bus has high data transfer rate, but it doesn't allow cache-to-cache transfer. In order to evaluate the effect of cache-to-cache transfer upon the performance of system and to choose a best-performed protocol for HiPi bus, we simulate as follows : First, we analyze the performance of multiprocessor system with HiPi bus in terms of processor utilization through simulation. Each of cache coherence protocol is described by state transition diagram, and then the probability of each state is calculated by Markov steady state. The calculated probability of each state is used as input parameters of simulation, and modeling and simulation are implemented and performed by using SLAM II graphic symbols and language. Second, we propose the HiPi bus which supports cache-to-cache transfer, and analyze the performance of multiprocessor system with proposed HiPi bus in terms of processor utilization through simulation. Considered cache coherence protocols for the simulation are Write-through, Write-once, Berkely, Synapse, Illinois, Firefly, and Dragon.

I. 서 론

반도체 기술의 발달로 인하여 고속 및 다기능을 내장하는 고성능 프로세서가 개발되었다. 즉, 프로세서는 명령어의 파이프라인 처리 기능, MMU(Memory Management Unit), FPU(Float Pointing Unit), 캐쉬 메모리 등을 포함하고 있다. 이러한 고성능 프로세서의 출현은 컴퓨터 시스템의 성능을 계속 향상시켜 왔다. 그러나 최근 컴퓨터 시스템에서는 멀티미디어 데이타와 지식 정보 등을 효율적으로 처리할 수 있는 기능이 요구되고 있으며, 이러한 기능은 높은 수준의 연산 및 추론 능력과 방대한 데이타의 고속 연산 및 통신을 위한 구조를 갖는 시스템을 요구한다. 이러한 요구는 단일 프로세서 시스템의 한계를 넘어 다수의 프로세서가 효율적으로 병렬 처리할 수 있는 멀티프로세서 시스템의 구현에 관심을 갖게 하였다.[1,2,3,4]

일반적으로, 소규모 멀티프로세서 시스템은 구조가 간단하고, 구현과 확장이 용이한 공통 버스 구조가 채택된다. 공통 버스 구조를 갖는 멀티프로세서 시스템은 다수의 프로세서가 하나의 버스를 공유하며, 공통 버스는 프로세서와 메모리, 프로세서와 프로세서, 혹은 프로세서와 외부 디바이스 사이를 연결한다. 공통 버스 멀티프로세서 시스템에서 단일 버스의 사용은 버스를 사용하고자 하는 다수의 요구로 인하여 버스에서 병목현상을 초래한다. 또한, 메모리는 프로세서의 처리 속도에 비해 액세스 속도가 느리므로 많은 지연이 발생한다. 버스의 병목현상과 메모리의 지연은 시스템의 성능을 저해하는 가장 큰 문제점이다. 그러므로, 멀티프로세서 시스템의 성능 향상은 메모리의 지연과 버스의 병목 현상을 최소화 함으로써 기대할 수 있다.[4]

메모리의 지연을 최소화하는 방법은 고속의 메모리를 사용하는 방법, 인터리브 메모리를 사용하는 방법, 메모리 모듈에 입력 큐 버퍼를 사용하는 방법, 캐쉬 메모리를 사용하는 방법 등이 있다. 버스의 병목현상을 해결하기 위한 방법으로는 버스의 대역폭을 증가시키거나, 캐쉬 메모리를 사용하는 것이다.[3,4,5]

멀티프로세서 시스템에서 대역폭을 증가시키기 위하여 pended 프로토콜이 사용된다. Pended 프로토콜은 버스의 동작 과정을 여러 단계로 나누고 파이프라인으로 처리한다. 이러한 프로토콜을 요구/응답 분리 프로토콜(split request/response protocol)이라고도 한다. 멀티프로세서 시스템에서 pended 프로토콜이 사용된 예로는 Multimax 시스템의 Nanobus, Balance 시스템의 공통 시스템 버스, 그리고 국내에서 개발된 TICOM의 시스템 버스인 HiPi(Highly Pipelined) 버스 등이 있다.[3,6,7]

캐쉬 메모리는 고속의 버퍼 메모리로서, 프로세서 내부나, 프로세서와 메모리 사이에 두어 계층적 메모리를 구성한다. 멀티프로세서 시스템에서 캐쉬 메모리의 사용은 성능을 크게 향상시키는 반면, 복사된 데이타 값들 사이에 불일치가 발생하는 캐쉬 코히어런스 문제를 유발시킨다. 지금까지 캐쉬 코히어런스 문제를 해결하려는 프로토콜에 대한 많은 연구가 진행되어 왔다.[8,9,10,11,12] 버스 구조 시스템에서 사용되는 캐쉬 코히어런스 프로토콜은 스누핑(snooping)을 기반으로 한다. 스누핑이란 버스의 동작을 관찰하는 것을 말하며 스누핑을 통해서 캐쉬 내의 제어기는 데이타 불럭을 갱신, 무효화(invalidate), 또는 브로드

캐스트(broadcast) 시키고 해당 상태 비트를 변경한다. 스누핑을 기반으로 하는 전략에는 쓰기가 발생할 때마다 메모리를 갱신시키는 write-through와 임의의 캐쉬에서 쓰기를 위한 요구가 발생할 때마다 메모리를 갱신시키는 write-back 전략이 있다. Write-back 전략에는 공유된 데이타를 무효화 시키는 write-invalidate 방법과 공유된 데이타를 모두 갱신 시키는 write-broadcast 방법이 있다. 프로토콜에는 write-through와 write-invalidate 방법을 사용하는 Write-Once, Synapse, Berkely, Illinois, 그리고 write-broadcast 방법을 사용하는 Firefly, Dragon 등이 있다.[5,9,11] 이러한 프로토콜은 버스의 사용 시기와 횟수, 메모리의 갱신 시기와 횟수, 그리고 다른 캐쉬 내의 블럭을 처리하는 방법들이 각각 다르므로 프로토콜의 적용에 따라 시스템의 성능은 달라진다. 따라서, 캐쉬를 사용하는 공통 버스 멀티프로세서 시스템에서는 사용되는 버스의 특성에 맞는 적절한 프로토콜을 선택·적용하는 것이 필수적이다. 적절한 프로토콜을 선택하기 위해서는 버스에 캐쉬 코히어런스 프로토콜을 적용하고, 그에 대한 성능 평가가 이루어져야 한다. 프로토콜의 성능 평가는 현재까지 적지 않은 연구가 진행되어 왔다.[9,11,12]

본 논문에서는 캐쉬 메모리를 사용하고 HiPi 버스를 갖는 멀티프로세서 시스템에서 적절한 캐쉬 코히어런스 프로토콜을 선택하기 위해서 다양한 캐쉬 코히어런스 프로토콜을 적용하고, 각각에 대해 성능 평가를 실시하였다. 성능 평가를 위하여 SLAM II 그래픽 모델링과 시뮬레이션 언어를 이용하였다.[13] 정확한 입력 값을 사용하기 위해 각 프로토콜에서 정의된 상태를 Markov 정적 상태도로 나타내고, 이를 이용하여 각 상태에 대한 확률을 구하였다. 버스에서는 캐쉬의 각 상태에 따라 서로 다른 동작을 수행하게 되는데 계산된 확률 값은 시뮬레이션에서 프로세서가 캐쉬 블럭의 입력으로 이용하였다. 시뮬레이션은 프로세서의 수, 캐쉬의 적중률, 읽기 확률에 따른 프로세서의 효율을 구하였다. 프로세서 효율은 버스의 병목현상과 시스템 성능을 고찰할 수 있으며, 이를 이용하여 최적의 시스템을 구성할 수 있으므로 시스템 성능 평가에 자주 이용된다. HiPi 버스는 캐쉬 간의 블럭 전송을 허용하지 않는 특성 때문에 적용할 수 있는 프로토콜이 제한된다.[6] 그래서, HiPi 버스에 적용될 수 있는 Write-Through, Write-Once, Synapse, Illinois 프로토콜에 따라 시뮬레이션을 실시하고, 결과에 대한 성능 분석을 하였다.

HiPi 버스의 캐쉬 간에 직접 데이타를 전송 하지 못하는 특성 때문에 적용할 수 있는 프로토콜이 제한될 뿐만 아니라 전체적인 시스템 성능을 저하시킨다. 즉, 다른 캐쉬에 데이타를 전달하기 위해서는 메모리를 경유하여야 하고, 이는 메모리를 액세스하는 시간 지연이 발생하므로 전체 시스템의 성능을 저하시키는 결과를 초래한다. 본 논문에서는 캐쉬 간의 직접 데이타 전송이 가능한 HiPi 버스를 제안하고, 제안된 HiPi 버스를 갖는 멀티프로세서 시스템에 대한 시뮬레이션을 실시하여 성능 분석을 하고, HiPi를 사용한 시스템에 비해 향상된 성능의 정도를 비교 평가하였다. 이는 차후 HiPi 버스의 개선에 있어서 선행되어야 할 연구이다. 제안된 HiPi 버스에서는 Write-Once, Berkely, Illinois, Firefly, Dragon 등의 프로토콜 적용이 가능하며, 각 프로토콜을 시뮬레이션에 적용하므로써 제안된 HiPi 버스에 적합한 프로토콜의 선택을 위한 분석을 기술하였다.

Ⅱ. 시스템 구성과 캐쉬 코히어런스

1. 공통 버스 시스템의 구성

캐쉬 코히어런스 프로토콜을 성능 평가하기 위해 사용되는 시스템은 N개의 프로세서 P와 각각의 전용 캐쉬 C, M개의 공통 메모리 모듈, 하나의 공통 버스로 구성된다. 시스템 구성에 대한 블럭도는 그림 1과 같다.

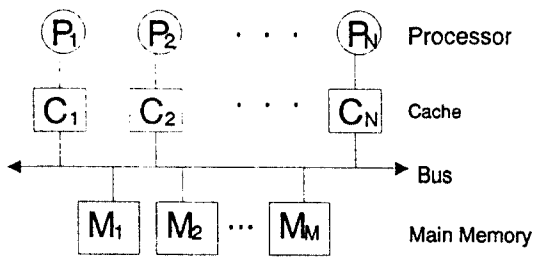


그림 1. 전용캐쉬를 갖는 멀티프로세서 시스템
Fig. 1. Multiprocessor system with private Cache

프로세서 모듈은 각각 하나의 프로세서와 전용 캐쉬를 가지며, 버스의 사용을 요구하고 어드레스 또는 데이타를 전달하기 위해서 RQ(Requester)가 존재하고, 메모리 모듈에는 RQ에 대한 응답을 위해 RP(Responder)가 존재한다. 전용 캐쉬는 캐쉬 메모리와 캐쉬 제어기, 그리고 스누프 제어기로 구성된다.

스누프 제어기는 버스의 동작 상태를 감시하여 캐쉬 코히어런스를 유지할 수 있도록 캐쉬 메모리의 내용을 메모리에 갱신(write-back) 시키거나, 무효화 시키는 동작, 또는 캐쉬 메모리의 내용을 변경(update) 시키는 기능을 한다.[6]

메모리 모듈은 일정 크기의 메모리와 메모리 제어기를 갖는다. 버스의 특성에 따라서 메모리 모듈에 입력 큐를 두는데, pended 프로토콜의 경우 입력 큐의 존재 여부는 메모리를 접근하려는 시도에서 재시도(retry)를 줄일 수 있는 좋은 방법 중의 하나이다. RQ에서 보낸 어드레스의 데이타를 제공하기 위하여 RP를 갖는데, 이는 데이타 버스의 중재와 데이타 전송 기능을 수행하는 버스와 메모리 모듈의 인터페이스 역할을 한다.[14]

공통 버스는 멀티프로세서 시스템을 지원하는 HiPi 버스이다. HiPi 버스는 pended 프로토콜, 캐쉬 코히어런스 프로토콜의 지원, 100 MB/sec의 전송률, 80 ns의 주기, 데이타와 어드레스 버스가 독립적으로 동작하는 특성을 가진다. HiPi 버스는 8개의 전송 형태를 가지며, 16개까지 확장이 가능하도록 설계되어 있다. 8가지 전송형태는 Normal Read (NRD), Normal Write (NWR), Read for Read (RFR), Read for Write (RFW), Invalidate (IVD), Write Back (WRB), Interlock Read (LCR), Interlock Write (LCW) 이다. 전송 형태 중에서 캐쉬와 관련된 것은 RFR, RFW, IVD, WRB 이다. RFR은 캐쉬로 읽기 동작을 우하여 블럭을 읽어들이는 전송 형태이고, RFW는 쓰기 동작을 위하여 읽어들이는 전송 형태이다. IVD는 다른 캐쉬에 있는 블럭의 내용을 무효화시키기 위하여 어드레스를 버스에 전송하는 것이다. WRB는 캐쉬에서 미스가 발생하여 기존에 데이타 블럭과 대치(replacement)가 발생하는데, 캐쉬에 존재하던 데이타 블럭이 갱신된 경우에는 메모리의 데이타도 갱신시켜야 하는데 이러한 경우에 사용되는 전송형태이다.[6,14,15]

본 연구에서는 HiPi 버스가 16개까지 전송형태를 확장할 수 있음을 이용하여 캐쉬 간의 데이타 전송 형태인 Cache-to-Cache(CTC)를 제안하고, 시뮬레이션을 위하여 CTC에 대한 타이밍도를 그림 2와 같이 가정하였다. CTC를 지원하기 위해서 버스의 상태 버스(status bus)에 메모리 금지(memory inhibit) 신호선을 첨가하고, 각 프로세서 모듈에서 RQ에서 요청한 어드레스의 데이타를 제공할 수 있도록 RP 기능의 첨가를 가정하였다. CTC는 스누핑에 의해 버스의 데이타를 임의의 캐쉬에서 가지고 있는 것을 감지하면, 해당 프로세서 모듈은 메모리에서 데이타를 제공하는 것을 금지하기 위해 메모리 금지선을 액티브(active) 시키고 RP를 구동하여 해당 어드레스의 데이타를 제공한다. HiPi 버스에서는 캐쉬 간의 블럭 전송이 이루어지지 않고 메모리를 경유해야 하므로 메모리를 접근하는 시간이 추가적으로 지연된다. 메모리의 접근 시간은 프로세서 및 버스의 주기에 비해 매우 느리므로 공유된 블럭이 많을수록 시스템의 성능은 저하된다. 또한, 블럭 전송을 할 때마다 메모리를 갱신하게 되므로 Shared Dirty(공유되는 수정된 캐쉬 내의 블럭 상태) 상태를 갖는 Berkely 프로토콜을 지원하지 못하고, 캐쉬 간의 전송을 기반으로 하는 write-broadcast 방법을 지원하지 못한다. 그러므로 캐쉬 간의 블럭 전송을 지원하는 버스의 수정에 대한 제안은 캐쉬 간의 블럭 전송시 메모리 지연의 감소와 많은 프로토콜을 적용할 수 있는 융통성에 비추어 필수적이다.

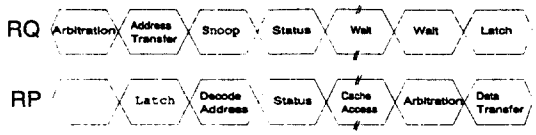


그림 2. CTC 전송의 타이밍도
Fig. 2. Timing diagram of CTC Transfer

2. 시스템 인자

본 논문에서 모델링 및 시뮬레이션에 사용될 인자를 다음과 같이 정의하였다.

N : 프로세서의 수
S_m : 메모리의 크기 (블럭 수)
S_c : 캐쉬의 크기 (블럭 수)
T_p : 프로세서 주기
T_a : 중재 시간 (어드레스와 데이타 버스 모두 같다)
T_b : 버스 주기
T_m : 메모리 접근 시간
T_c : 캐쉬 접근 시간
h : 적중률
m : 미스율 (m = 1 - h)
r : 메모리 액세스하는 경우 중 읽기 동작을 수행할 확률

w : 메모리 액세스하는 경우 중 쓰기 동작을 수행할 확률 (w=1-r)
i : 프로세서가 메모리 요구를 하지 않고 내부 동작을 수행할 확률
q : 공유된 정도
f : 메모리에 대한 캐쉬 크기의 비 ($f=S_c/S_m$)
a : 블럭 하나를 택하여 읽기 동작을 수행할 확률
b : 블럭 하나를 택하여 쓰기 동작을 수행할 확률
c : 블럭 하나를 택하였을 때, 미스될 확률
d : 두 개 이상의 캐쉬가 같은 블럭을 가질 확률
e : 다른 캐쉬에서 임의의 블럭이 존재할 확률
P_n : 캐쉬 블럭이 n 상태에 존재할 확률 (단, n은 프로토콜에서 정의된 상태)

메모리에서 하나의 블럭을 취하여 캐쉬에 가져다 놓을 확률 f는 메모리 블럭의 수에 대한 캐쉬 블럭의 수의 비로 표현된다. 공유된 정도를 나타내는 q는 두 개 이상의 캐쉬에 공유된 블럭의 수를 나타낸다. 그러므로, 두 개의 캐쉬가 같은 블럭을 공유할 확률은 f와 q의 곱으로 정의할 수 있다. 이를 확장하여 N 개의 캐쉬 중에서 두 개 이상의 캐쉬에 같은 블럭이 존재할 확률은 식 (1)과 같다.

$$d = q\sum_{i=2}^{N}\begin{bmatrix} N \\ i \end{bmatrix} f^{(i-1)} \simeq f\cdot q\cdot(N-1) \qquad (1)$$

또한 캐쉬에서 읽기, 쓰기, 미스가 일어날 확률을 a, b, c로 정의하였고, 각각은 캐쉬에서 한 블럭이 선택된 후 발생하므로 $a=r/S_c$, $b=w/S_c$, $c=m/S_c$로 정의된다.

3. 캐쉬 코히어런스 프로토콜

멀티 프로세서 시스템에서 각각의 프로세서마다 캐쉬 메모리를 사용할 경우, 캐쉬 메모리에 복사된 데이타를 다른 캐쉬 메모리에 있는 동일한 주소의 내용과 불일치하는 문제가 발생하는데, 이를 캐쉬 코히어런스(cache coherence) 문제라고 한다. 버스 기반 멀티프로세서 시스템에서 이를 해결하기 위한 프로토콜은 write-invalidate 및 write-broadcast에 기반을 둔 6가지 프로토콜이 사용된다.[8,9,11]

Write-invalidate 방법을 기반으로 하는 프로토콜은 Write-Once, Synapse, Berkely, Illinoise 등이 있고, write-broadcast 방법을 기반으로 하는 프로토콜은 Firefly, Dragon 등이 있다. 그림 3은 Illinois 프로토콜의 상태 천이도를 나타내었고, 다른 프로토콜에 대해서도 상태 천이도를 동일한 방법으로 나타낼 수 있다. 상태 천이도에서 N(NOTIN)은 캐쉬의 초기 상태나 대치로 인해 비어 있는 상태, I(INVALID)는 무효한 상태, V(VALID)는 수정되지 않은 상태, S(SHARED)는 공유된 상태, D(DIRTY)는 수정된 상태를 나타낸다.[9,11,12]

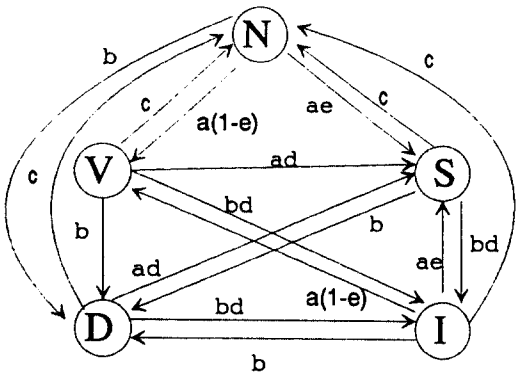


그림 3. Illinois의 상태 천이도
Fig. 3. The state diagram of Illinois

그림 3의 Illinois 프로토콜에서 각 상태의 확률을 Markov 정적 상태의 확률을 이용하여 구하면, 식 (2), (3), (4), (5), (6)과 같다. 다른 프로토콜에 대해서도 아래의 식과 같은 방식으로 각각의 상태에 대한 확률을 계산하였으며, 이 확률 값을 시뮬레이션의 입력 값으로 이용하였다.

$$P_N = \frac{c}{a+b+c} \qquad (2)$$

$$P_V = \frac{a(1-e)(P_N+P_I)}{ad+b+bd+c} \qquad (3)$$

$$P_S = \frac{ad(1-P_N-P_I)+ae(P_N+P_I)}{ad+b+bd+c} \qquad (4)$$

$$P_D = 1-P_N-P_V-P_S-P_I \qquad (5)$$

$$P_I = \frac{bd(1-P_N)}{a+b+bd+c} \qquad (6)$$

윗 식에서 e는 요구한 블럭이 다른 캐쉬에 존재할 확률이다. 즉, 다른 캐쉬에서 N 상태가 아닌 다른 상태로 존재할 확률이다. e를 구하면, 식 (7)과 같다.

$$e = f \cdot g \cdot (N-1) \cdot (1-P_N) \quad (7)$$

Ⅲ. 모델링 및 시뮬레이션

본 연구에서는 SLAM II 그래픽 모델을 이용하여 시스템을 모델링 하였으며, 혼합된(mixed) 네트워크 모델링을 이용하였다. 모델링에 대한 검증은 프로세서 각 동작 상태에 따른 동작과정을 추적(trace)하여 모델링과 비교 검증하였다. 시뮬레이션은 모델에 근접한 환경을 구축할 수 있는 장점이 있으나, 시뮬레이션에 대한 가정과 인자의 정확한 설정에 의존적으로 결과가 출력된다. 따라서 시뮬레이션에서는 정확한 가정과 인자를 설정하는 것이 중요하다. Ⅱ장에서 계산된 각 프로토콜의 상태 확률은 프로세서의 수, 읽기를 수행할 확률, 캐쉬에서 히트할 확률에 따라 달라진다. 올바른 가정의 확립을 위하여 본 연구에서는 변수에 따른 각 상태의 값을 시뮬레이션의 입력으로 이용하였다. 본 연구의 시뮬레이션 환경은 UNIX 운영체제를 사용하는 SUN SPARC 워크스테이션에서 SLAM II 컴파일러를 이용하였다.[13]

1. 시뮬레이션을 위한 가정

시스템의 성능 평가를 위하여 시뮬레이션에 사용되는 인자는 프로세서, HiPi 버스, 메모리 모듈 등을 고려하여 다음과 같이 가정 하였다.

1) 버스는 캐쉬 데이타 블럭의 상태에 따라 다른 동작을 수행하는데, 이러한 특성을 모델링하기 이해 2장에서 계산된 각 상태의 확률 값, P_n을 이용하였다.
2) 프로세서는 항상 처리할 작업(job)이 있고, T_p는 50 ns 이다. 프로세서가 내부 명령을 수행할 확률, i는 0.692이다. 이는 MC68030 마이크로프로세서에 대한 성능을 참조하였다.[14]
4) 캐쉬의 접근 시간, T_c는 50 ns 이고, 프로세서 주기 내에 접근이 가능하다.
5) 메모리의 접근 시간, T_m은 320 ns 이다.
6) 중재 시간, T_a는 80 ns 이다.
7) 버스의 주기, T_b는 80 ns 이다.
8) 시뮬레이션 시간은 100,000 ns 이다.
9) 메모리는 2개의 입력 큐를 가지고 있고, 16개의 모듈로 이루어져 있다.
10) 캐쉬 간의 전송이 가능한 경우, 캐쉬의 액세스에 대한 우선 순위는 외부의 요구에 있다.
11) 메모리에 대한 캐쉬 크기의 비, f는 0.000125이다.
12) 캐쉬 간의 공유 정도, q는 100 이다.

2. SLAM II 모델링과 시뮬레이션

시뮬레이션은 먼저, HiPi 버스에 대하여 적용할 수 있는 Write-Through, Write-Once, Synapse, Illinois를 적용한 후, 각각을 모델링하고 시뮬레이션하였다. 다음으로, 캐쉬 간의 블럭 전송이 가능하도록 HiPi 버스를 수정 제안하고, Write-Once, Berkely, Illinois, Firefly, Dragon 프로토콜을 제안된 버스에 각각을 적용하여 모델링과 시뮬레이션 하였다. SLAM II 그래픽 모델링은 그림 4와 같다.

시뮬레이션에서 변수로 사용된 값은 프로세서의 수, 읽기를 수행할 확률, 그리고 적중률이다. 변수로 사용되지 않을 경우, 프로세서는 20개, 읽기를 수행할 확률은 0.8, 적중률은 0.95로 디폴트(default) 값을 설정하였다.

프로세서는 가정에서 정해진 확률에 따라 read hit, read miss, write hit, write miss 엔트리를 발생시키고, 각 엔트리는 구분 인자로서 어트리뷰트 값을 가진다. 메모리 모듈의 선택은 일향 분포(uniform distribution)을 사용하였다.

제안된 버스의 모델링은 메모리 모듈을 선택하는 부분에서 HiPi 버스와 다르다. 즉, 스누핑으로 인해서 필요한 경우에는 메모리를 선택하는 대신 다른 프로세서의 모듈에 있는 캐쉬를 선택하게 된다. 그러므로 메모리의 지연은 없고, 단지 캐쉬의 지연만 존재하므로, 응답 시간은 줄어들게 된다. 전용 캐쉬의 요구와 외부에서의 요구가 동시에 이루어 질 때, 외부 요구에 우선권이 주어지므로 캐쉬에서는 단지 캐쉬 액세스 주기 만을 소요하게 된다.

시뮬레이션은 HiPi 버스에서 Write-Through, Write-Once, Synapse, Illinois 프로토콜에 대한 시뮬레이션을 수행하였고, 제안된 HiPi 버스에서 Write-Once, Berkely, Illinois, Firefly, Dragon 프로토콜에 대한 시뮬레이션이 이루어졌다.

Ⅳ. 결과 및 고찰

프로세서의 수, 적중률, 읽기를 수행할 확률에 따른 시뮬레이션을 통하여 프로세서의 효율을 결과로 구하였다.

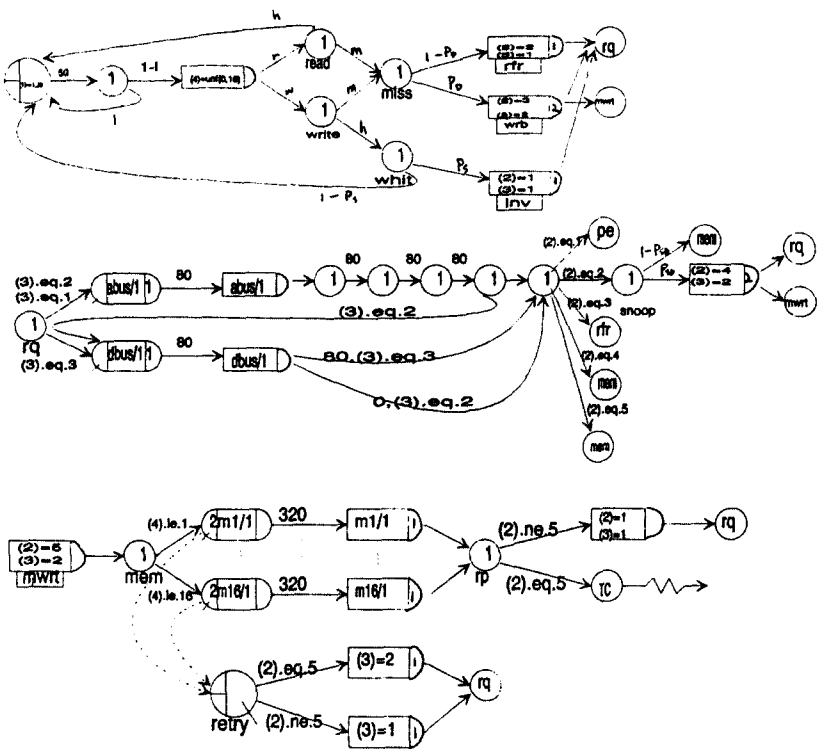


그림 4. SLAM II 그래픽 모델링
Fig. 4. Graphic modeling by SLAM II

1. HiPi 버스에서 시뮬레이션의 결과 및 고찰

HiPi 버스에서 가능한 Write-Through, Write-Once, Synapse, Illinois 프로토콜에 대한 시뮬레이션을 하였다. N=20, r=0.8 일때, 적중률에 따른 결과는 그림 5와 같다. 프로세서의 효율은 적중률이 증가함에 따라서 급격히 향상되었다.

Write-Through와 Write-Once가 성능이 나쁜 이유는 쓰기 동작을 수행할 때마다 버스를 요구함으로 버스에서 충돌에 의한 지연이 많기 때문이다. 좋은 성능을 보이고 있는 프로토콜은 Synapse와 Illinois인데, 그중 Illinois가 가장 좋은 성능을 보이는 것을 알 수 있다. 적중률이 높아질수록 캐쉬에서 대치가 적게 발생하고, 메모리를 갱신하는 횟수가 줄어드므로 Synapse나 Illinois의 성능은 비슷해 진다.

h=0.95, r=0.8 일때, 프로세서의 수에 따른 결과는 그림 6과 같다. 프로세서의 수가 증가함에 따라 각 프로세서의 효율은 버스에서의 충돌로 인하여 점차

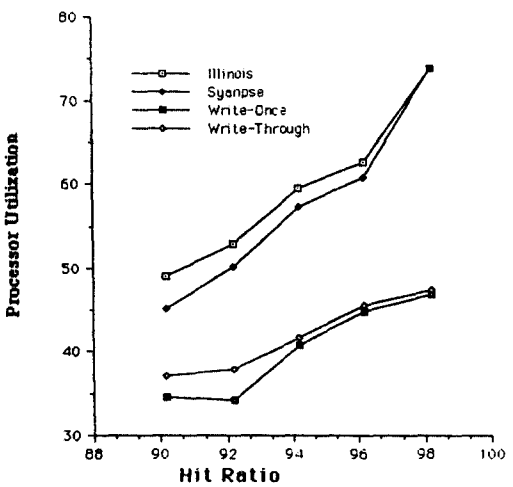


그림 5. 히트율에 따른 프로세서의 이용률
Fig. 5. Processor utilization vs. hit ratio

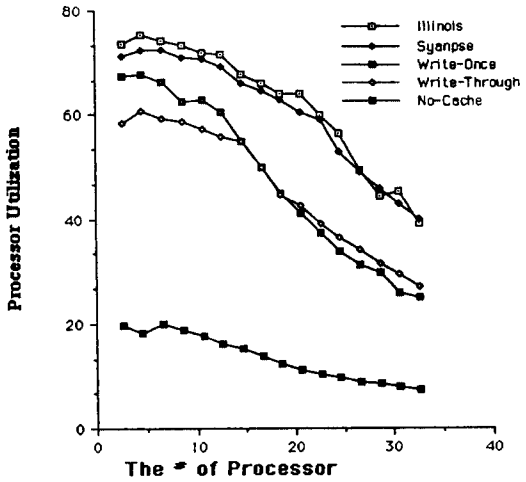


그림 6. 프로세서 수에 따른 프로세서의 이용률
Fig. 6. Processor utilization vs. the number of processors

감소한다.

프로세서의 수에 따른 결과를 살펴보면, write-through와 write-back 전략에 있어서 큰 성능 차이가 있음을 발견할 수 있다. Illinois 프로토콜이 가장 좋은 성능을 나타낸다. 각 프로토콜마다 프로세서의 수가 8개 이상일 경우 버스의 병목현상으로 급격히 프로세서 효율이 저하됨을 알 수 있다.

프로세서의 수가 2개에서 4개로 증가할 때, 캐쉬가 없는 경우에는 캐쉬 코히어런스가 적용되지 않으므로 버스에서의 충돌이 증가하여 프로세서의 효율이 감소한다. 그러나, 캐쉬를 사용하여 캐쉬 코히어런스 프로토콜을 적용하게 되면 캐쉬 코히어런스를 유지하는데 소요되는 오버헤드(overhead)가 존재하게 된다. 프로세서의 수가 2개나 4개일 경우는 오버헤드의 양은 비슷한데 프로세서의 수가 증가하여 각 프로세서에 대한 오버헤드의 양이 감소하므로 각 프로세서의 효율은 Illinois 프로토콜의 경우에 0.02 정도 증가하게 된다. 프로세서의 4개 이상이 되면, 오버헤드에 의해서 미치는 영향보다 버스에서 사용 요구의 충돌에 의해서 지연되는 영향이 증가하여 다시 프로세서의 효율은 감소한다.

N=20, h=0.95 일때, 읽기를 수행할 확률에 따른 결과는 그림 7과 같다. 읽기의 확률을 높이면, 캐쉬에서 미스가 발생하여 대치가 일어날 때 메모리의 갱신을 줄일 수 있으므로 성능이 향상된다. 읽기의 확률은 하드웨어에 의존적이 아니라 어플리케이션에 의존적이다. HiPi 버스에서 WRB 전송 형태는 메모리를 갱신하기 위하여 사용되는데, 쓰기 동작의 확률이 높으면 WRB의 요구가 증가되어 프로세서의 효율을 저하시킨다. 그러나 읽기 확률이 적었을 경우에도 60-70% 사이에서 보다 프로세서 효율이 좋다. 그 이유는 HiPi 버스가 파이프라인 동작을 하기 때문이다.

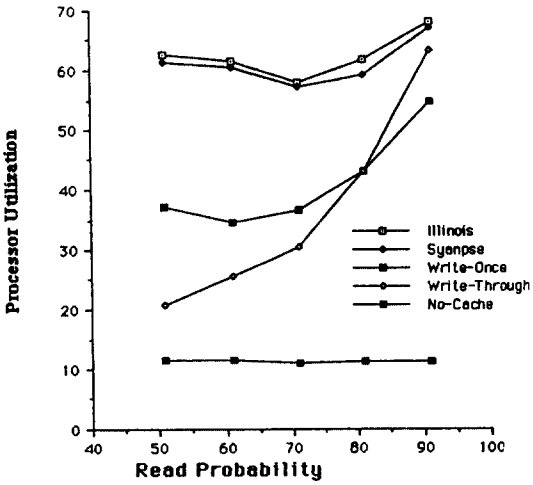


그림 7. 읽기 확률에 따른 프로세서의 이용률
Fig. 7. Processor utilization vs. read probability

시뮬레이션의 결과 HiPi 버스에서는 Illinois 혹은 Synapse 프로토콜을 선택 적용하는 것이 가장 좋은 성능을 유지할 수 있다. 20개의 프로세서, 0.95의 적중률, 0.8의 읽기를 수행할 확률의 조건에서 Illinois와 Synapse의 시스템 성능은 각각 12.56, 11.86을 나타내었다.

2. 제안된 버스에서 시뮬레이션의 결과 및 고찰

제안된 버스는 캐쉬 간의 블럭 전송을 지원하므로, 스누핑을 통하여 다른 캐쉬 내의 블럭이 히트하면 메모리의 지연보다 적은 시간으로 지정된 블럭을 가져올 수 있다. Write-Through와 Synapse 프로토콜은 블럭간의 전송이 성능에 변화를 주지 못하므로, 제안된 버스에서 시뮬레이션은 생략하였다. 제안된 버스에서의 시뮬레이션은 캐쉬 간의 블럭 전송이 유효한 Write-once, Berkely, Illinois, Firefly, Dragon에 대하여 프로토콜에 대한 시뮬레이션을 하였다.

N=20, R=0.8 일때, 적중률에 따른 각 프로토콜

의 결과는 그림 8과 같다. Write-Once는 HiPi 버스와 비교해서 차이가 거의 없다. 그 이유는 R 상태가 되기 위하여 메모리를 접근하기 때문에 캐쉬를 접근하는 횟수가 적다. 이는 캐쉬 간의 전송에 관계가 없으므로 성능 향상이 없는 것이다. 그러나 Illinois 프로토콜의 경우는 캐쉬 간의 전송이 가능할 때 프로세서 효율이 0.1-0.2 정도 향상됨을 알 수 있었다. 프로토콜에 따른 성능 분석에 대한 해석적 방법에서는 write-broadcast 방법인 Firefly와 Dragon이 Illinois 보다 좋은 성능을 나타내었다.[11] 그러나 본 시뮬레이션에서는 쓰기 동작이 발생하였을 때에 버스의 동작이 무효화(invalidate) 시키는 동작 보다 많은 시간이 소요되므로 write-broadcast 방법보다 write-invalindate 방법이 더욱 좋은 성능을 나타내었다. 그림 10에서 쓰기 동작을 수행할 확률이 높아질수록 write-broadcast 방법을 적용한 경우의 시스템 성능이 더욱 저하되는 것은 HiPi 버스에서 write-broadcast가 부적합하다는 것을 알 수 있다.

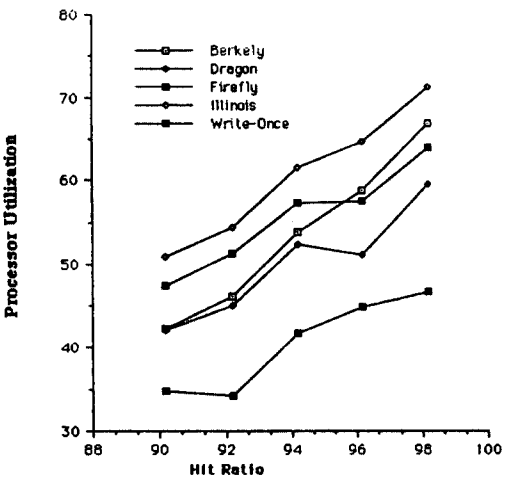


그림 8. 히트율에 따른 프로세서의 이용률
Fig. 8. Processor utilization vs. hit ratio

h=0.95, r=0.8 일때, 프로세서의 수에 따른 결과는 그림 9와 같다. 프로세서의 수가 증가하면, 프로세서의 효율은 감소하게 된다. Illinois. Firefly, Dragon 프로토콜은 프로세서의 수가 적을 경우에는 비슷한 성능을 보였으나, 프로세서의 수가 많아지면 프로세서 간의 상호 전송이 증가하게 되므로 Firefly와 Dragon은 성능이 현저하게 감소된다. 프로세서의 수가 32개일 때, Illinois의 프로세서 효율은 0.38이고, Dragon은 0.28로 0.1 정도의 차이가 나타난다.

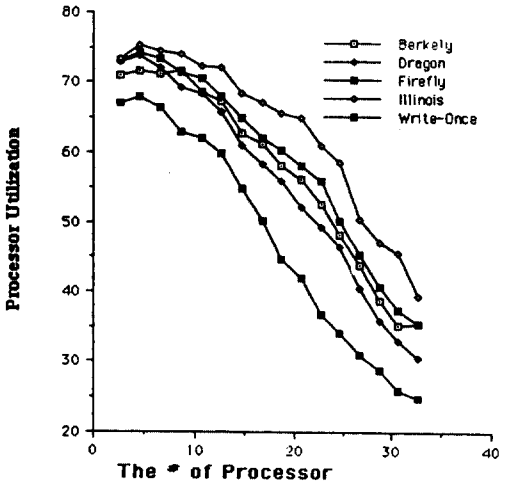


그림 9. 프로세서 수에 따른 프로세서의 이용률
Fig. 9. Processor utilization vs. the number of processors

N=20, h=0.95 일때, 읽기를 수행할 확률에 따른 결과는 그림 10과 같다. 캐쉬 간의 전송은 프로세서의 효율을 향상시키고, 프로세서의 대기 시간을 줄임으로써 효율을 향상 시킨다. 쓰기 동작에 broadcast를 하는 Firefly와 Dragon은 읽기를 수행할 확률에 민

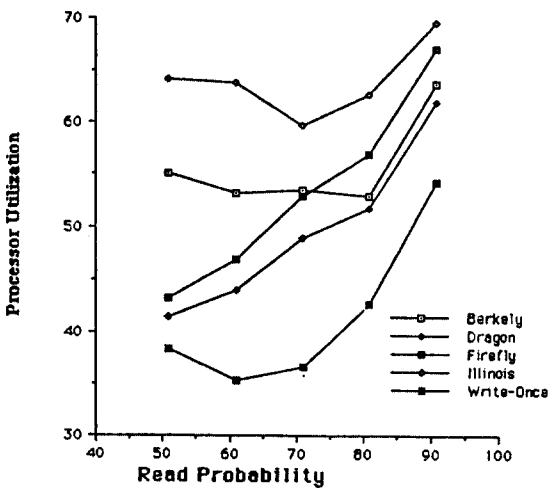


그림 10. 읽기 확률에 따른 프로세서의 이용률
Fig. 10. Processor utilization vs. read probability

감하게 작용한다. 즉, 제안된 HiPi 버스에서 T_{IVD}보다 T_{CTC}가 더욱 많은 시간이 소요되므로, 쓰기가 많을수록 프로세서의 효율은 다른 프로토콜에 비하여 급격히 저하된다.

결론적으로 제안된 버스는 시뮬레이션 결과에서 알 수 있듯이 프로세서의 효율을 향상시킬 뿐만아니라, write-broadcast 프로토콜을 지원하는 다양한 융통성을 제공한다. 그러나 write-broadcast는 다중 쓰기의 오버헤드로 인하여 어플리케이션이나 프로세서의 수에 따라 민감하게 성능이 결정된다. Illinois 프로토콜은 HiPi 버스에서나 제안된 버스에서 가장 좋은 성능을 나타내었고, 제안된 HiPi 버스에서는 HiPi 버스에서보다 프로세서 효율이 0.01 내지 0.02 정도 높게 나타났다.

V. 결 론

공통 버스를 갖는 멀티프로세서에서 성능을 저해하는 가장 큰 요소는 버스의 병목현상과 메모리의 지연이다. 이를 해결하기 위한 방법은 pended 프로토콜 버스를 사용하는 것과 캐쉬 메모리를 사용하는 것이다.

HiPi 버스는 pended 프로토콜을 채택한 버스이다. HiPi 버스는 캐쉬 코히어런스를 유지하기 위하여 Write-Through, Write-Once, Synapse, Illinois 등의 프로토콜을 지원할 수 있다. 그러나 캐쉬 간의 데이타 블럭의 전송을 지원하지 않는다. 캐쉬 간의 블럭 전송은 메모리의 사용을 줄임으로써 메모리에서 발생하는 지연을 줄일 수 있다. 제안된 HiPi 버스는 각 프로세서 모듈에 요구/응답(RQ/RP) 기능을 갖게하고, 상태 버스에 메모리 금지 신호선을 첨가하여 캐쉬 간의 블럭 전송이 가능하도록 제안한 버스이다. 캐쉬 간의 블럭 전송이 가능해지면 Write-Once, Berkely, Illinois, 그리고 Firefly, Dragon 등의 프로토콜을 지원할 수 있다.

버스의 동작은 캐쉬 코히어런스 프로토콜의 적용으로 인하여 오버헤드가 발생한다. 이 오버헤드의 양은 write-invalidate 방법을 기반으로 하는 프로세서에서는 프로세서의 수가 증가하는 것에 무관하게 오버헤드의 양이 거의 동일하다. 그러므로 프로세서의 수가 2개에서 4개로 증가하는 경우에는 오버헤드의 양은 일정한데 프로세서의 수가 증가해서 각 프로세서의 오버헤드가 감소하여 성능이 향상된다. 프로세서의 효율은 0.02 정도 향상된다.

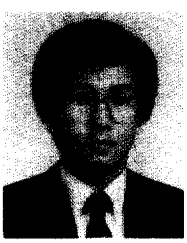
캐쉬 코히어런스에 따른 성능 분석에 대한 기존 연구의 수치적이나 해석적 접근에서는 Write-broadcast 방법인 Firefly와 Dragon이 성능이 좋은 것으로 나타났으나, pended 프로토콜을 사용하는 HiPi 버스에서 시뮬레이션을 통한 각 캐쉬 코히어런스 프로토콜에 대한 성능 분석에서는 write-invalidate 방법인 Illinois 프로토콜이 가장 성능이 좋은 것으로 나타났다. 그 이유는 HiPi 버스에서 write-broadcast 방법을 수행하기 위해서 소요되는 시간이 write-invalidate 방법을 수행하는 시간보다 많이 소요되기 때문이다. 적중률이 0.95, 프로세서가 20개, 읽기의 확률이 0.8일 때, 제안된 HiPi 버스에서의 시뮬레이션 결과 Illinois, Firefly, Dragon의 프로세서 효율은 각각 0.62, 0.56, 0.51 이었다. HiPi 버스에서나 제안된 버스에서 시스템 성능은 Illinois 프로토콜이 가장 적절한 것으로 판정된다.

적중률이 0.95, 프로세서가 20개, 읽기의 확률이 0.8일때, HiPi 버스에서 Write-through, Write-once, Synapse, Illinois 프로토콜을 적용하였을 때에 프로세서의 효율은 각각 0.1, 0.413, 0.398, 0.593, 0.628이고, 시스템의 성능은 2, 8.26, 7.96, 11.86, 12.56이다. 제안된 HiPi 버스에서 Write-once, Berkely, Illinois, Firefly, Dragon 프로토콜을 적용하였을 때에 프로세서의 효율은 각각 0.411, 0.551, 0.640, 0.571, 0.512이고, 시스템 성능은 8.22, 11.02, 12.8, 11.42, 10.24이다. 이는 HiPi 버스에 CTC 라는 캐쉬 간의 데이타 전송이 가능하게 하므로써 전체 시스템의 성능을 증가시킬수 있으므로, 차후 HiPi 버스에 개선에 있어서 성능 향상을 위해 캐쉬 간의 데이타 전송 기능을 고려해야 할 것이다.

참 고 문 헌

1. D. D. Gajski and J. K. Pier, "Essential Issues in Multiprocessor system," IEEE Transaction on Computer, Vol.18, June 1985, pp.9-27.
2. T. L. Harman, The Motorola MC68020 and MC68030 Microprocessors, Prentice-Hall, 1989.
3. E. F. Gehringer, J. Abullarade, and M. H. Gulyn, "A Survey of Commercial Parallel Processors," Computer Architecture News, 1989.
4. K. Hwang and F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

5. A. J. Smith, "Cache Memories," ACM Computing Surveys, Vol.14, No.3, Sep. 1982, pp. 473-530.
6. An Do Ki, Byung Kwan Park, Won Sae Sim, Kyeng Yong Kang, Yong Ho Yoon, "Highly Pipelined Bus : HiPi-Bus," The ETRI Journal 전자통신, Vol.13, No.3, Oct. 1991, pp.33-50.
7. J. Sanguinetti, "Performance of a Massage-Based Multiprocessor," IEEE Comp., Sep. 1986, pp.47-55.
8. P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," IEEE Computer, Jun. 1990, pp.12-24.
9. J. Archibald and J. L. Baer, "Cache Coherence Procotols : Evaluation Using a Multiprocessor Simulation Model," ACM Transactions on Computer Systems, Vol.4, No.4, Nov. 1986, 273-298.
10. S. V. Adve, V. S. Adve, M. D. Hill, and M. K. Vernon, "Comparison of Hardware and Software Cache Coherence Schemes," 18th International Symp. Comp. Architecture, May 1991, pp.298-308.
11. Q. Yang, L. N. Bhuyan, and B. C. Liu, "Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," IEEE Transactions on Computers, Vol.38, No.8, Aug. 1989, pp.1143-1153.
12. Q. Yang and L. N. Bhuyan, "A Queueing Network Model For A Cache Coherence Protocol on Multiple-bus Multiprocessors," Proc. Int'l. Conf. Parallel Processing, Vol.1, Aug. 1988, pp.130-137.
13. A. Alan B. Pritsker, Introduction to Simulation and SLAM II, John Wiley & Sons, 1986.
14. 최창렬 외 3인, "공유 메모리와 단일 버스로 구성되는 다중프로세서의 하드웨어 성능 분석," 한국정보과학회 논문지, Vol.16, No.5, Sep. 1989, pp.399-409.
15. 윤용호, 안희일, 임기욱, "주전산기 하드웨어 개발," 대한전자공학회 전자계산연구회 컴퓨터기술, Vol.7, No.1, Dec. 1990, pp.96-108.



金 永 川(Young Chon Kim) 정회원
1956년 12월 10일생
1980년 2월 : 고려대학교 전자공학과 졸업
1982년 2월 : 고려대학교 전자공학과(공학석사)
1987년 2월 : 고려대학교 전자공학과(공학박사)
1989년 : 전자계산기 기술사
1989년 8월~1990년 8월 : University of California Irvine. Post-Doc.
1986년 9월~현재 : 전북대학교 공과대학 컴퓨터공학과 부교수

姜 寅 坤(In Kon Kang) 정회원
1966년 4월 22일생
1990년 2월 : 전북대학교 컴퓨터공학과 졸업
1992년 2월 : 전북대학교 컴퓨터공학과(공학석사)
1992년 3월~현재 : 전북대학교 컴퓨터공학과 박사과정

黃 勝 郁(Seung Uk Hwang) 정회원
1958년 12월 25일생
1984년 8월 : 고려대학교 전자공학과 졸업
1986년 8월 : 고려대학교 전자공학과(공학석사)
1992년 8월 : 고려대학교 전자공학과(공학박사)
1986년 9월～1992년 8월 : 한국전자통신연구소
1992년 9월～현재 : 한국해양대학교 제어계측공학과

崔 眞 圭(Jin Kyu Choi) 正會員
1958년 9월 20일생
현재 : 한남대학교 전자공학과 조교수
제14권 제12호 참조