상용 작업부하를 이용한 다중프로세서 컴퓨터 시스템 성능 평가

Performance Evaluation for a Multiprocessor Computer System Using a Commercial Workload

박진원*, 정용화**, 박종원**, 안희일**, 이강우***, 김양우***
Jin-Won Park , Yong-wha Chung, Chong-Won Park,
Hee-Il Ahn, Kang-woo Lee, Yang-Woo Kim

Abstract

The CC-NUMA based, distributed shared memory is an emerging architecture for multiprocessor computer systems because of its scalability and easy of programming. In this paper, we analyzed performance of a ring-based, CC-NUMA multiprocessor computer system using a commercial workload targeted for popular OLTP applications. Based on the traces collected from real machines, the characteristics of the commercial workload could be obtained. The simulation results showed that the bottleneck on the ring could be effectively removed by using a dual ring structure. We believe our simulation methodology and results will help us to design better multiprocessor computer systems for commercial application domains.

* 영산대학교 컴퓨터정보공학부
** 한국전자통신연구원
*** 동국대학교 정보통신공학과

1. 서론

컴퓨터 기술은 매우 빠른 속도로 발전해 가고 있으며, 그 중에서도 가장 활발하게 연구가 진행되고 있는 분야는 병렬 처리(parallel processing) 분야이다[1]. 즉, 병렬 처리를 위한 다중프로세서(multiprocessor) 컴퓨터 시스템은 대규모 계산이 수반되는 문제를 처리하기 위해 필요한 계산 능력을 효과적으로 제공하고 있으며 이러한 추세는 계속될 전망이다. 따라서 다중프로세서 컴퓨터 시스템의 성능을 예측하기 위한 효과적인 방안을 모색할 필요성이 증대되고 있으며, 이러한 대규모 컴퓨터 시스템을 실제로 구현하기 전에 미리 제안된 아키텍춰의 특성을 평가해 보는 것은 매우 중요한 일이다. 그러나, 다중프로세서 아키텍춰를 갖는 컴퓨터 시스템은 순차적인 컴퓨터 시스템에 비하여 모델링하기가 훨씬 어려운데, 이는 아키텍춰와 병렬 프로그램 동작간의 매우 복잡한 상호작용 때문이다.

일반적으로 Uniform Memory Access(UMA) 방식의 버스에 의한 공유메모리 컴퓨터 시스템은 단일 어드레스 공간을 지원함에 따라 프로그래밍이 용이하지만, 프로세서 수를 증가시켜 시스템 성능을 향상시키는 확장성(scalability)에 제한을 받아 일반적으로 프로세서 수가 대략 10개 이상인 경우는 전체 시스템 성능이 오히려 낮아지는 현상이 발생한다. 이러한 확장성에 대한 제한에 따라 공유메모리 컴퓨터 시스템을 규모가 크고 복잡한 응용 분야에서 활용하기에는 한계를 가지고 있다. 더불어 수백 개 이상의 프로세서를 탑재하여 대규모 병렬 처리 시스템으로 사용하는 분산메모리 컴퓨터 시스템은 매우 뛰어난 확장성을 가지고 있으나, NO Remote Memory Access(NORMA) 구조에 기반을 두고 있기에 이 시스템에 운용되는 응용 프로그램을 작성하기가 매우 어려워, 주로 단순 작업이 많이 반복되는 과학계산용으로 사용되고 있다.

이러한 동향에 따라 과학계산 분야나 상용 분야에서 공통으로 활용할 수 있으며 공유메모리와 분산메모리 시스템의 혼합 형태를 갖는 병렬 처리 시스템인 분산공유메모리(distributed shared memory) 시스템들이 새롭게 등장하였다[1]. 특히, 기존의 성숙된 시장을 확보하고 있는 공유메모리 시스템에 쉽게 적용될 수 있으면서도 확장성이 뛰어난 Cache Coherent Non-Uniform Memory Access(CC-NUMA) 방식의 컴퓨터 시스템은 1990년대 중반 이후에 등장하고 있는 새로운 병렬 처리 시스템이라 할 수 있다. 특히, 인텔의 SHV와 같은 버스 기반 공유메모리 노드들이 서로 연결된 CC-NUMA 시스템은 노드 내에서의 캐쉬 일관성 유지뿐만 아니라 노드간에서의 일관성도 유지해야 하므로 복잡한 프로토콜을 필요로 한다. 이러한 CC-NUMA 시스템의 성능을 분석한 기존의 연구로는 [2-5] 등이 있으나, 대부분 과학계산용 작업부하를 이용한 연구이다. 한편 다중프로세서 시스템의 또 다른 응용 분야인 상용 응용 분야의 작업부하를 이용한 성능 연구가 최근에 발표되기 시작했지만[6-8], 그 대상 시스템이 버스 기반 공유메모리 시스템 등 단순한 다중프로세서 시스템에만 국한된 실정이다.

따라서 본 연구에서는 분산공유메모리 시스템의 또 다른 응용 분야인 상용 응용 분야의 대표적인 작업부하를 이용하여 노드 내와 노드 간에서 복잡한 캐쉬 일관성 유지 프로토콜을 갖는 CC-NUMA 다중프로세서 시스템의 성능을 비교 분석한다. 특히, 캐쉬 일관성 유지 프로토콜과 상호연결망 등 시스템 공유자원과의 관계는 시스템성능에 결정적인 역할을 하므로 이들을 통합한 성능 분석을 수행한다.

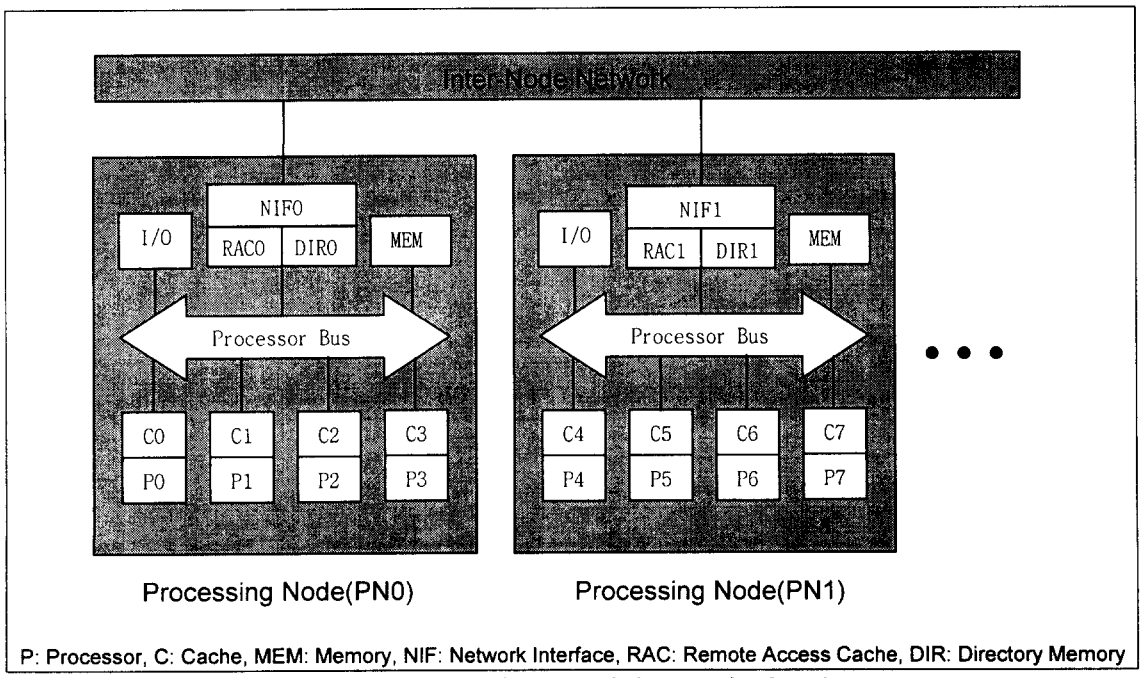
본 논문의 구성은 다음과 같다. 먼저 2장에서 대상 시스템(target system)인 CC-NUMA 방식의 다중프로세서 컴퓨터 시스템의 특징에 대해서 간단히 설명하고, 3장에서는 부과될 상용 작업부하의 특징을 기술한다. 4장에서는 본 논문에서 수행하는 시뮬레이션 방법을 설명하

며, 이러한 방법으로 시뮬레이션된 결과를 5장에 기술하고, 마지막 6장에서 결론을 맺는다.

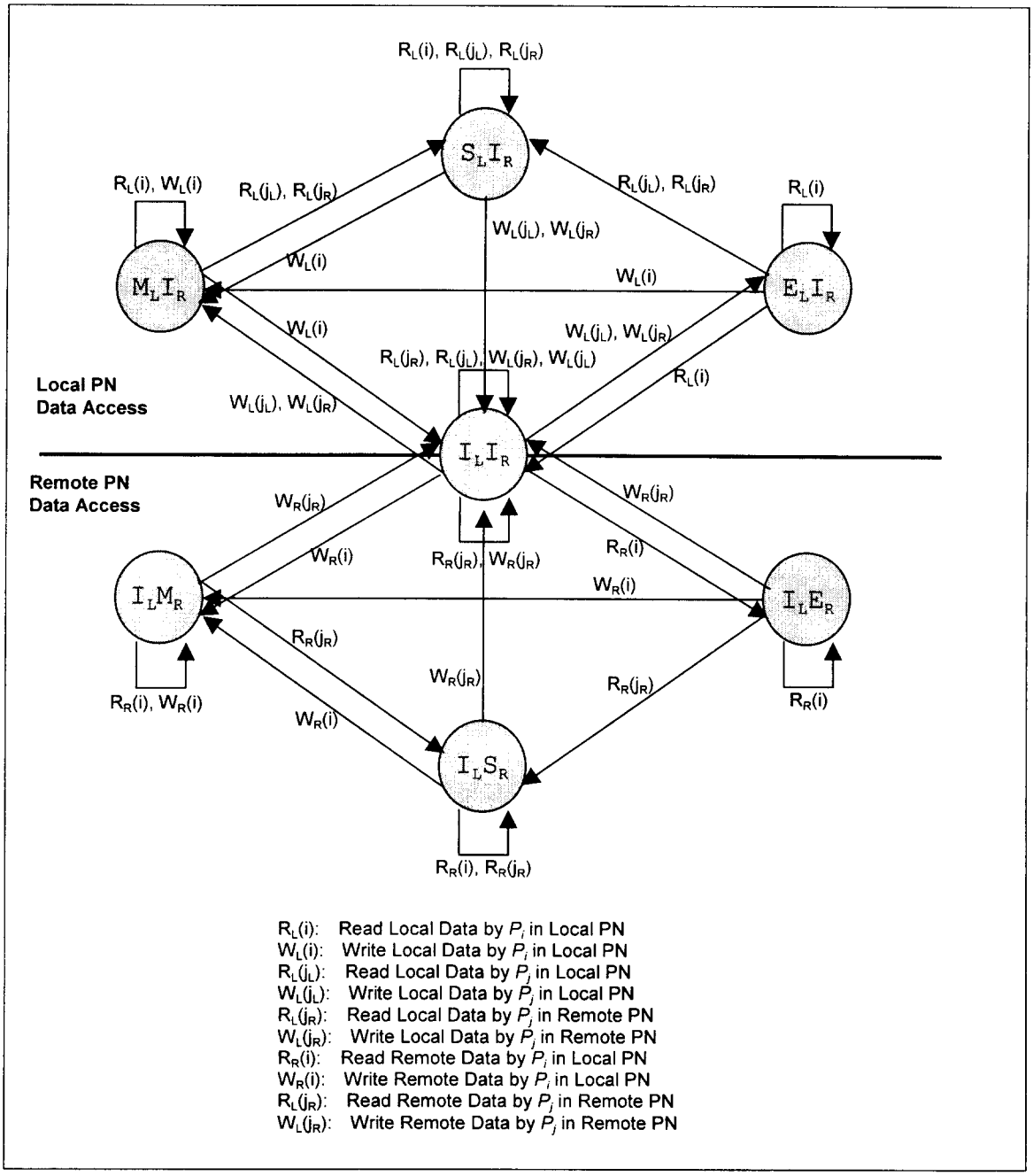
2. 다중프로세서 컴퓨터 시스템

본 논문에서는 대상 시스템으로 버스 기반 공유메모리의 Processing Node(PN)들이 상호연결망(inter-node network)으로 연결되어 분산공유메모리를 제공하는 CC-NUMA 방식의 다중프로세서 컴퓨터 시스템을 가정한다(그림 1 참조). 하나의 PN내에서 각각의 프로세서는 캐쉬 메모리를 가지며, 버스를 통하여 공유 메모리를 액세스한다. 캐쉬 메모리는 Least Recently Used(LRU) replacement policy를 갖는 4-way, set-associative 방식으로 구성되며, 캐쉬 일관성(cache coherence) 유지는 Illinois 프로토콜[1]을 따른다. Illinois 프로토콜에서는 캐쉬내의 데이터에 쓰기 동작을 수행할 경우, 공유 메모리내의 대응하는 블록 수정을 그 캐쉬 블록이 다른 블록을 위하여 방출(replacement)될 때까지 연기하는 write-back policy를 갖는다. 또한, 다중프로세서 컴퓨터 시스템에서 복수개의 캐쉬가 동일한 데이터를 보유하고 있는 경우에, 하나의 캐쉬에 쓰기 동작이 수행되면 다른 캐쉬가 보유하던 그 데이터는 invalidate됨으로써 복수 개 캐쉬간 데이터 일관성이 유지된다.

또한 각 PN내에는 다른 PN의 데이터를 액세스하기 위한 원격접근 캐쉬(Remote Access Cache: RAC)와 이러한 RAC간의 일관성을 유지하기 위한 디렉토리가 존재한다. 본 논문에서는 full-map 방식[1]의 디렉토리를 가정한다. 여기서 full-map 방식의 디렉토리는 각 PN당 1비트의 엔트리를 갖는데, PN의 수가 증가함에 따라 디렉토리의 크기가 증가한다는 단점이 있지만, 임의 개수의 캐쉬 복사본을 동시에 관리할 수 있다는 장점 때문에 수백 개 규모의 프로세서로 구성된 시스템에서 많이 이용되고 있기 때문이다.



<그림 1> 다중프로세서 컴퓨터 시스템



$R_L(i)$: Read Local Data by P_i in Local PN
$W_L(i)$: Write Local Data by P_i in Local PN
$R_L(j_L)$: Read Local Data by P_j in Local PN
$W_L(j_L)$: Write Local Data by P_j in Local PN
$R_L(j_R)$: Read Local Data by P_j in Remote PN
$W_L(j_R)$: Write Local Data by P_j in Remote PN
$R_R(i)$: Read Remote Data by P_i in Local PN
$W_R(i)$: Write Remote Data by P_i in Local PN
$R_L(j_R)$: Read Remote Data by P_j in Remote PN
$W_L(j_R)$: Write Remote Data by P_j in Remote PN

<그림 2> 캐쉬 일관성 유지 프로토콜의 상태 천이도

각 프로세서의 캐쉬 메모리와 각 PN내 RAC간의 데이터 일관성 유지를 위해 본 논문에서 가정한 프로토콜의 상태 천이도(state diagram)를 <그림 2>에 나타낸다. <그림 2>에서 첨자 L로 표시된 것은 Local을 의미하고, R은 Remote를 의미한다. 먼저, PN내 프로세서 캐쉬간의 데이터 일관성을 유지해 주는 Illinois 프로토콜은 **M**(Modified), **E**(Exclusive Clean), **S**(Shared), **I**(Invalid) 4개의 상태로 구성된다. 여기서 M은 하나의 프로세서만이 자신의 캐쉬 내에 유효한 블록을 가지고 있고 메모리의 데이터는 유효하지 않으며 어떤 다른 캐쉬에도 유효한 블록이 존재하지 않는 상태를 의미한다. 또한 E는 하나의 캐쉬만이 유효한 블록을 가지나 메모리의 데이터 또한 유효한 상태를 의미하고, S는 두개 이상의 프로세서가 캐쉬 내에 이 블록을 변경되지 않은 상태로 보유할 수 있는 상태를 의미한다. 이 프로토콜의 특징은 추가적인 버스 신호(shared 신호, SH)를 이용하여 BusRd시 다른 캐쉬가 현재 그 블록을 보유하고 있는지 여부를 결정하는 점이다.

예를 들어 다음의 간단한 시나리오를 살펴보자. <그림 1>의 프로세서 *P1*에서 최초의 u 값 읽기(PrRd) 동작을 수행하면, 버스상에는 BusRd 동작이 발생하여 공유 메모리에 있는 u 값("5"라고 가정)은 캐쉬 *C1*으로 로드되고 캐쉬의 상태는 I에서 E로 변경된다. 이때 프로세서 *P3*에서 u 값의 읽기(PrRd) 동작을 수행한다면, 버스상에는 BusRd 동작이 발생하여 공유 메모리의 u 값 "5"가 캐쉬 *C3*로 로드된다. 그러나 최초의 읽기 동작과 달리, 버스를 모니터하던 *C1*이 버스상에 shared 신호 SH를 보냄으로써 *C3*의 상태는 I에서 S로 변경된다. 물론 *C1*의 상태도 E에서 S 상태로 변경된다. 그리고 *P3*가 u 값을 "7"로 쓰기(PrWr) 동작을 수행한다면, 버스상에는 BusRdX 동작이 발생하여 *C1*의 복제본은 invalidate되고 *C3*의 상태는 S에서 M로 변경된다. 그러나 write-back policy에 의해 공유 메모리의 u 값은 변경되지 않은 상태("5")로 남아있다. 만약 *P1*이 u 값의 읽기(PrRd) 동작을 다시 수행한다면, 버스상에는 BusRd 동작이 발생하여 *C3*에 있는 M 상태의 u 값("7")이 *C1*에 로드되는 동시에 공유 메모리 값이 수정된다. 이때 *C1*과 *C3*의 상태는 모두 S 상태로 변경됨으로써 캐쉬간 일관성이 유지된다.

다음은 로컬 PN에 할당된 데이터에 대한 read miss인 경우를 예로 들어 PN간 캐쉬 일관성 유지 시나리오를 살펴보자. <그림 1>에서 프로세서 *P0*의 local PN *PN0*에 할당된 데이터 읽기가 실패하면, 디렉토리 *DIR0*를 확인하여 그 데이터의 복사본이 remote PN *PN1*에 M 상태로 존재함을 알 수 있다. *PN0*에서 이 복사본에 대한 write-back 요청이 *PN1*으로 보내지면, *PN1*의 *RAC1*에 저장된 복사본은 M 상태에서 S 상태로 변경되고 그 복사본을 *PN0*로 전송한다. 복사본을 받은 *PN0*에서는 *RAC1*의 상태 변경을 반영하기 위해 *DIR0*를 수정하고, 요청한 프로세서 *P0*에 그 복사본을 제공하며, 캐쉬 *C0*는 S 상태로 설정한다. 대상 시스템의 캐쉬 일관성 유지 프로토콜에 대한 좀 더 자세한 내용에 대해서는 [9]를 참조하기 바란다.

3. 작업부하

본 논문에서는 상용 작업부하로 On-Line Transaction Processing(OLTP) 응용을 목표로 한 TPC-B[10]를 사용한다. TPC 벤치마크는 데이터베이스 시스템의 성능을 비교하는데 많이 이용되는 표준 벤치마크로서, 현재 공식적으로는 TPC-C가 TPC-B를 대체하였지만, TPC-B가 구현이 간단하고 프로세서/메모리 관점에서는 TPC-C의 경우와 유사하여 다중프로세서 컴퓨터 시스템용 작업부하로 많이 이용된다[7,8].

```
BEGIN TRANSACTION
    Update Account where Account_ID = Aid
        Read Account_Balance from Account
        Set Account_Balance = Account_Balance + Delta
        Write Account_Balance to Account
    Write to History
        Tid, Bid, Aid, Delta, Time_stamp
    Update Teller where Teller_ID = Tid
        Set Teller_Balance = Teller_Balance + Delta
        Write Teller_Balance to Teller
    Update Branch where Branch_ID = Bid
        Set Branch_Balance = Branch_Balance + Delta
        Write Branch_Balance to Branch
COMMIT TRANSACTION
```

<그림 3> TPC-B에서의 트랜잭션[4]

TPC-B는 은행의 뱅킹 시스템을 모델링한 것으로, 각각의 트랜잭션은 임의로 선택된 계좌의 잔고를 수정하는 것이다. 즉, TPC-B에서는 트랜잭션 생성기에 의해서 Teller ID *Tid*, Branch ID *Bid*, Account ID *Aid*, Account 변동액 *Delta*가 주어지면, <그림 3>과 같은 트랜잭션이 수행된다. 이때 Teller, Branch, Account, History의 4개 테이블이 사용되는데, Account, Teller, Branch의 레코드 수 비율은 100,000:10:1이 된다. Teller 레코드는 *Tid*, *Bid*, Teller_Balance 필드로 구성되며 적어도 100바이트의 크기를 갖는다. 또한 Branch 레코드에는 *Bid*와 Branch_Balance로 구성되며 적어도 100바이트의 크기를 가지며, Account 레코드는 *Aid*, *Bid*, Account_Balance로 구성되며 적어도 100바이트의 크기를 갖는다. 마지막으로 History 레코드는 *Aid*, *Tid*, *Bid*, *Delta*, Time_stamp로 구성되며, 적어도 50바이트의 크기를 갖는다.

4. 시뮬레이션 방법

본 논문에서는 성능분석의 방법으로 CSIM[11]을 이용한 컴퓨터 시뮬레이션을 채택하였다. CSIM은 분포 구동 방식[12]의 시뮬레이션 모델을 작성할 수 있는 C 기반의 high-level 언어로서, 작업부하를 생성하기 위하여 통계적 분포 함수를 활용한다. 또한 본 논문에서 살펴본 CC-NUMA 시스템의 성능지수로는 공유자원인 PN내의 상호연결망과 PN간의 상호연결망에서의 이용률 및 큐 길이를 설정하였다. 이는 사용자가 경험하는 궁극적인 시스템 성능지수인 응답시간, 처리율 등에 이들이 가장 큰 영향을 미치기 때문이다.

우선 CSIM으로 작성된 CC-NUMA 시뮬레이터에 부가되는 TPC-B 작업부하를 생성키 위해, EZDB[13]라는 DBMS에 TPC-B에서 정의한 트랜잭션을 CacheMire[3]상에서 실행시켜 나타난 메모리 액세스 카운트를 <표 1>에 나타내었다. 여기서 Account, Teller, Branch 테이블의 레코드 수는 각각 400,000, 40, 4를 가정하였다. 또한 프로세서 캐쉬의 크기가 1MB일 때 하나의 트랜잭션당 약 26개의 읽기 캐쉬 접근 실패와 4개의 쓰기 캐쉬 접근 실패가 관

찰되었다. 이는 Data General사에서 제시한 결과[14]와 매우 유사한 값임을 알 수 있다.

<표 1> TPC-B 트랜잭션의 평균 액세스 카운트 [13]

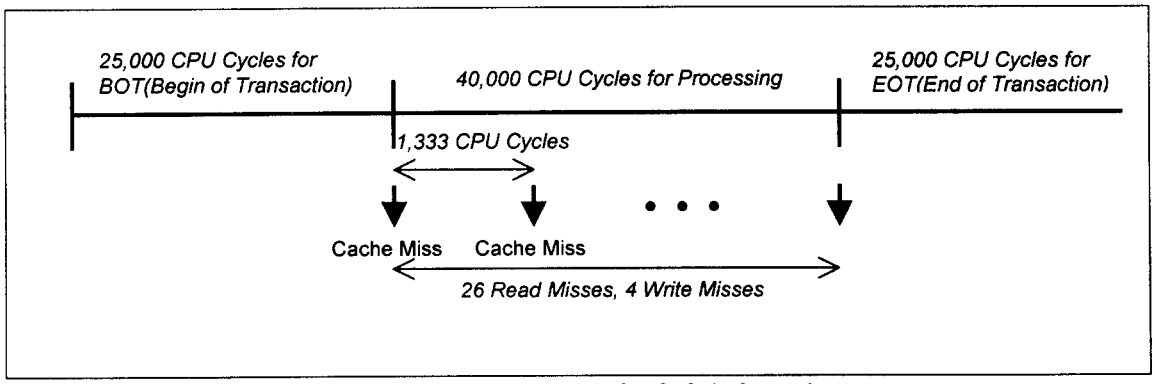
Transaction Processing					
Instruction Fetch	Private Data		Shared Data		
	Read	Write	Read	Write	Lock
34,258	2,063	291	261	41	51

다음에 대상 시스템인 CC-NUMA 시스템에 대해 다음과 같은 가정을 한다. 여기서 네번째와 다섯번째 가정은 매우 일반적인 것으로써, 이는 모든 인스트럭션과 사유 데이터에 대해서는 프로세서간 상호작용이 없기 때문이다.

- 하나의 인스트럭션은 1 CPU 싸이클을 필요로 한다.
- 캐쉬 메모리로부터의 데이터 읽기는 2 CPU 싸이클을 필요로 한다.
- 캐쉬 메모리로의 데이터 쓰기는 3 CPU 싸이클을 필요로 한다.
- 모든 인스트럭션 접근(fetch)은 캐쉬 접근 성공(cache hit)이다.
- 모든 사유 데이터(private data)에 대한 접근(fetch)은 캐쉬 접근 성공(cache hit)이다.
- 모든 캐쉬 접근 실패(cache miss)는 공유 데이터(shared data)에 대한 접근에서만 발생한다.

이러한 CC-NUMA 시스템에 대한 가정과 <표 1>을 기준으로 TPC-B에 대해 다음과 같이 가정하고, 하나의 트랜잭션을 처리하는 과정을 요약하면 <그림 4>와 같다. 즉, 각각의 프로세서에서는 <그림 4>로 표현되는 TPC-B 작업부하들이 독립적으로 수행되며, 이 작업부하 모델은 대상 시스템의 프로세서 수에 무관하다고 가정한다.

- TPC-B 질의의 BOT(Begin of Transaction) 및 EOT(End of Transaction)에 약 25,000 CPU 싸이클이 필요하며[6], 이 기간동안 다른 coherence 트랜잭션은 발생하지 않는다.
- TPC-B 질의의 처리에 약 40,000 CPU 싸이클이 필요하며, 이 기간동안 평균 26개의 읽기 캐쉬 접근 실패와 4개의 쓰기 캐쉬 접근 실패가 발생한다. 즉, 두개의 캐쉬 접근 실패 간격은 약 1,333 CPU 싸이클이다.



<그림 4> TPC-B 트랜잭션의 작업부하 모델

또한, 하나의 읽기 또는 쓰기 캐쉬 접근 실패와 쓰기 캐쉬 접근 성공이 발생하면 2장에서 언급한대로 데이터의 일관성을 유지하기 위한 동작을 수행해야 하는데, 이를 위하여 다음과 같이 가정한다. 즉, 어떤 한 데이터에 대한 쓰기 캐쉬 접근 성공이 발생할 때 시스템 내에 존재하는 이 데이터의 다른 사본들은 무효화되어야 하는데, 분포 구동 시뮬레이션에서는 몇 개의 사본이 어느 프로세서 캐쉬 또는 RAC에 존재하는지를 확률분포로 정의해야 하는데, 본 논문에서는 [5,8]에서와 같이 이를 선형분포로 가정한다.

예를 들어, N개의 PN과 $4N$개의 프로세서로 구성된 시스템에서 쓰기 연산의 대상이 되는 데이터가 하나의 프로세서 캐쉬 내에 존재할 경우, 이 데이터는 그 프로세서가 속한 PN의 MEM에 존재한다. 이때 PN의 다른 프로세서 캐쉬에 존재하는 이 데이터 사본의 개수는 1, 2, 또는 3이며, 이에 대한 분포를 균일하게 가정함으로써 발생된 임의의 숫자로 결정한다. 이러한 수의 사본을 포함하는 프로세서도 임의의 숫자를 발생시켜 결정한다. 또한 이 데이터의 사본이 $N-1$개의 다른 PN의 RAC에 존재할 수도 있으며, 이때 RAC의 개수 및 위치도 동일한 방식으로 결정한다.

5. 시뮬레이션 결과

본 논문에서는 PN간 상호연결망으로 SCI[15]와 같은 단방향 링(unidirectional ring)을 설정하였으며, 시뮬레이션에서 사용된 변수의 기본 값은 <표 2>와 같다.

먼저 전체 프로세서 수를 증가시킬 때 버스 및 링에서의 이용률(utilization) 및 평균 큐 길이의 변화를 나타내면 <그림 5>와 같다. 여기서 이용률이란 단위시간에 공유자원인 버스나 링이 활용되는 확률을 의미한다. 버스의 이용률 및 큐 길이는 매우 낮고 짧은 반면 링의 이용률과 큐 길이는 매우 높고 큰 것을 알 수 있다. 이는 링 토폴로지(topology)의 특성상 통신량이 많이 부과되기 때문이라고 해석된다. 또한 PN내 프로세서 수는 4로 고정시키고 PN의 개수를 증가시킴에 따라, 링의 이용률이나 큐 길이가 증가하지만 버스의 경우는 이 값들이 오히려 감소한다. 이는 링의 특성에 의한 것으로, PN의 숫자가 증가함에 따라 통신 대상이 되는 두 PN간의 평균 길이가 증가하며, 결국 링은 임의의 지점에서 보면 그 지점을 통과하는 통신량이 증가하기 때문이다. 그러나 PN내 프로세서 수가 4로 고정된 경우, 버스가 점유되는 시간은 동일하더라도 링에서의 통신량 증가에 따라 전체적인 수행시간이 증가하여 버스의 이용률은 다소 떨어지는 결과를 얻는다.

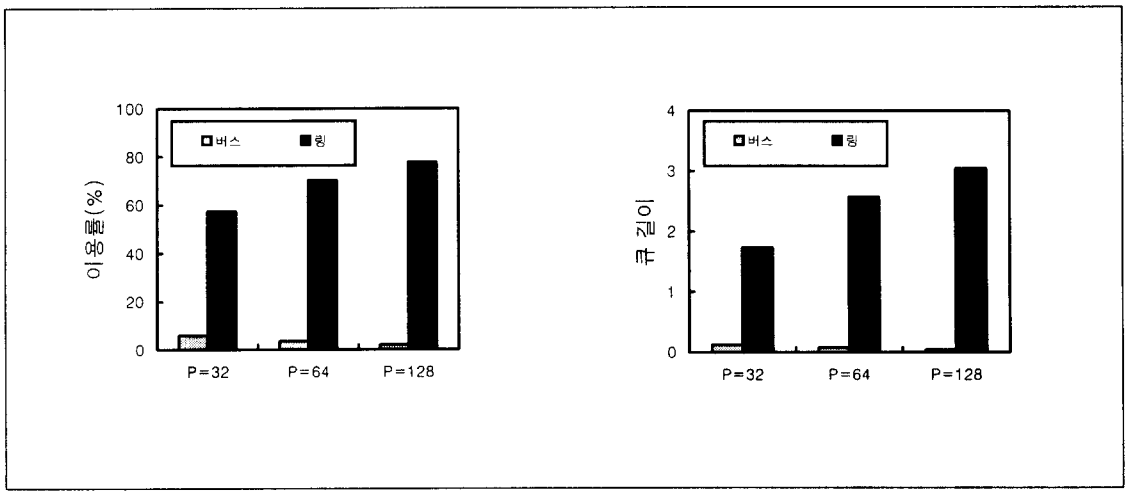
<표 2> 시뮬레이션 변수의 기본 값

시뮬레이션 변수	값
전체 프로세서의 수	32, 64, 128개
PN내 프로세서의 수	4개
프로세서의 속도	400 MHz
버스의 데이터 폭	8 Byte
버스의 속도	100 MHz
링의 데이터 폭	2 Byte
링의 속도	125 MHz
캐쉬 블록의 크기	32 Byte
RAC 블록의 크기	128 Byte

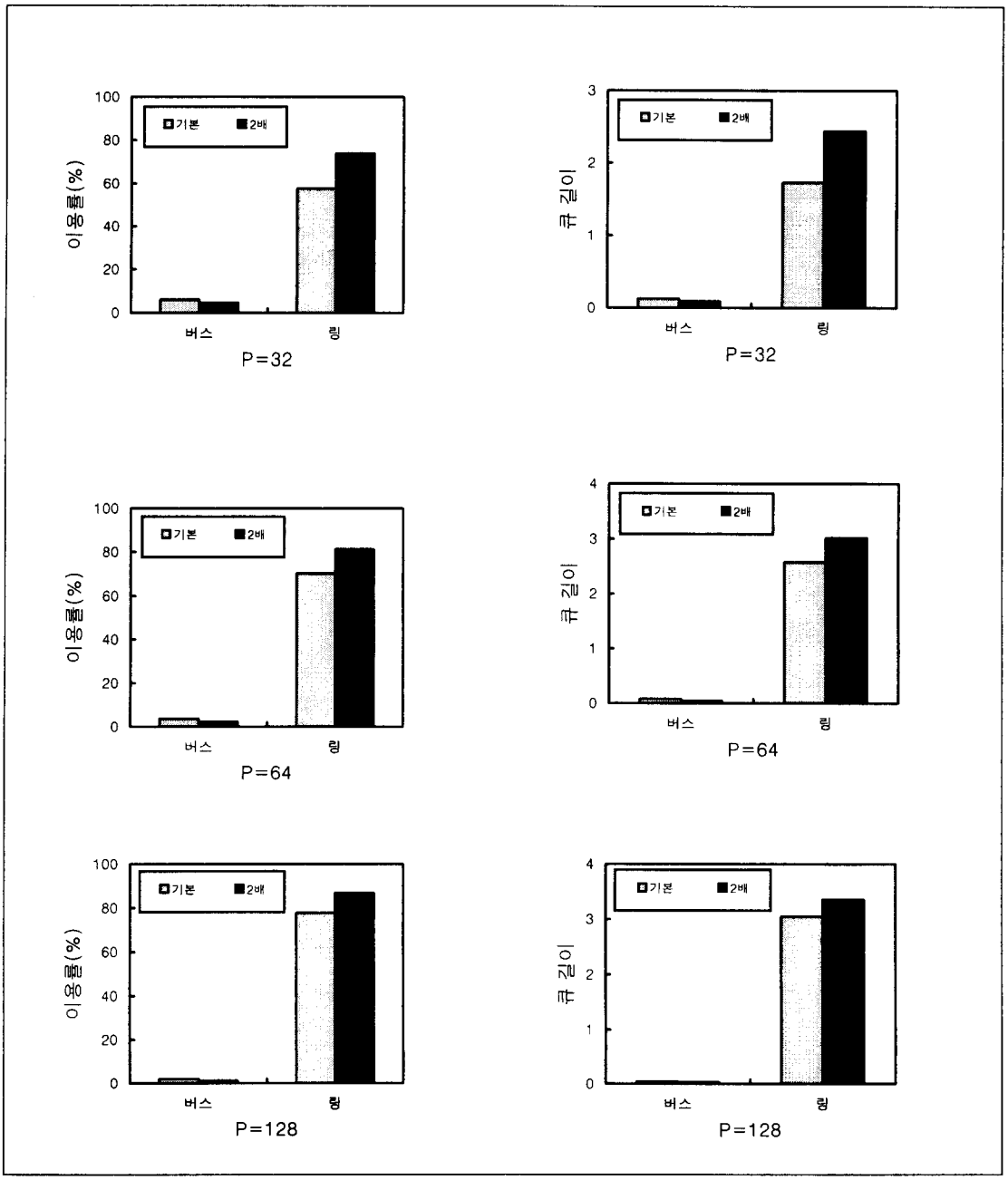
다음은 보다 많은 양의 프로세서간 통신을 부과할 때 버스나 링에 어떠한 변화가 발생하는가를 살펴보기 위해 데이터 캐쉬 접근 실패율을 <그림 4>에 나타낸 트랜잭션당 30개에서 2배인 60개로 증가시켜 보았다. 실제로 4장에서 언급한 TPC-B 작업부하의 특성은 400,000 Account, 40 Teller, 4 Branch 가정하에서 관찰된 결과로써, Account, Teller, Branch 테이블의 레코드 수를 증가

시키면 캐쉬 접근 실패율도 증가한다고 가정할 수 있다. <그림 6>에 나타난 바와 같이 캐쉬 접근 실패율이 증가함에 따라 프로세서간 통신양이 증가하며, 여기서도 위에서 설명한 이유와 같이 링의 이용률 및 큐 길이는 증가하는 반면 버스의 이용률 및 큐 길이는 감소함을 알 수 있다.

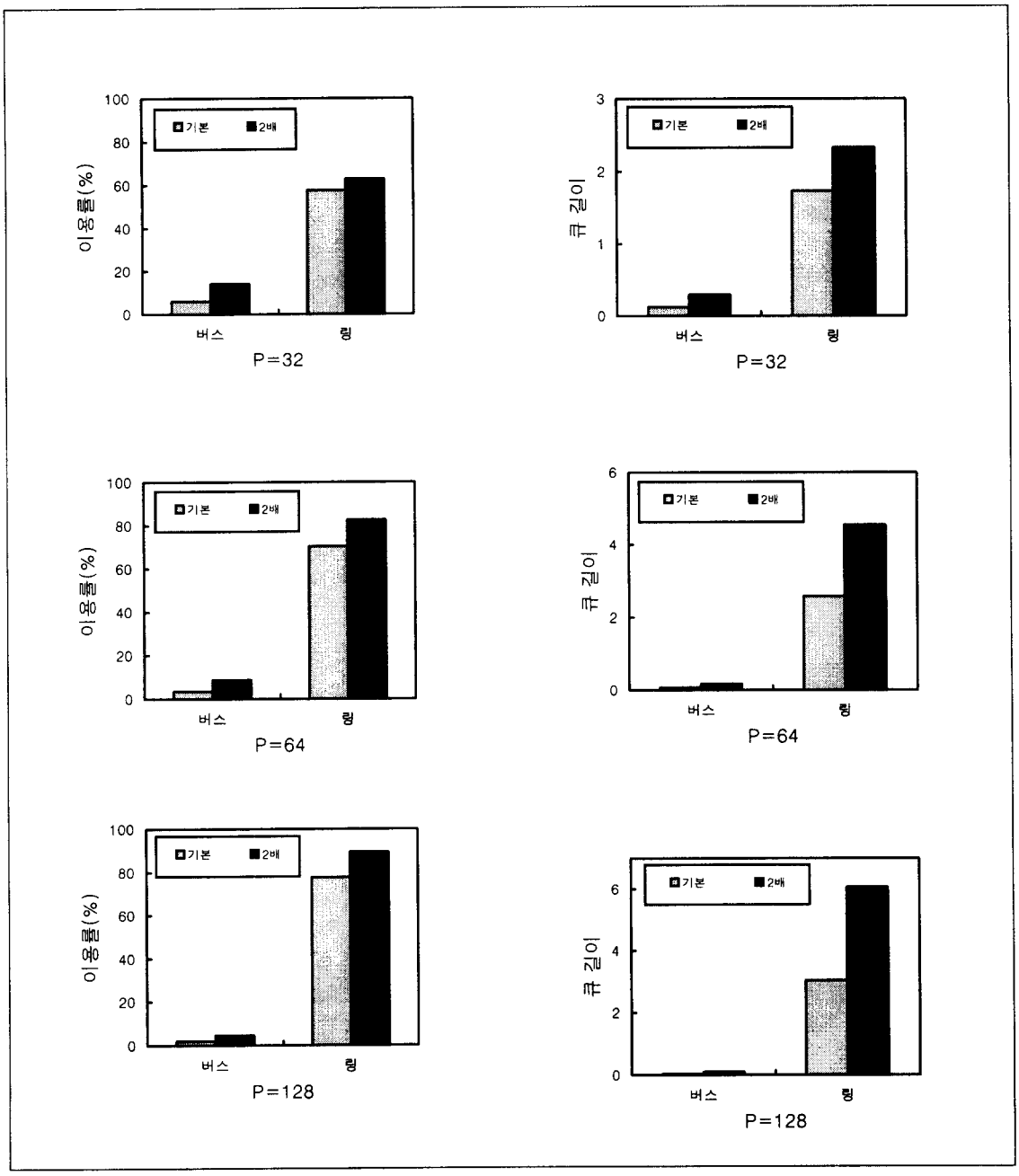
또한 <그림 5> 및 <그림 6>은 PN내 프로세서 수를 4로 고정시킨 시뮬레이션 결과이며, PN내 프로세서 수를 증가시킴에 따라 PN내 버스와 PN간 링에 부가되는 통신량도 변화함을 알 수 있다. 예를 들어 <그림 7>은 총 프로세서 수가 주어졌을 때 PN내 프로세서 수를 2배로 증가시킨 결과를 나타내며, PN내 버스의 이용률 및 큐 길이가 증가함을 알 수 있다. 특히 PN간 링의 값이 증가하는 이유는 총 프로세서 수가 고정된 경우에 PN내 프로세서 수를 증가시키면 PN의 수 및 링을 구성하는 PN간 링크(link)의 수가 감소하여, 결국 감소된 수의 링크 이용률 및 큐 길이는 증가한다.



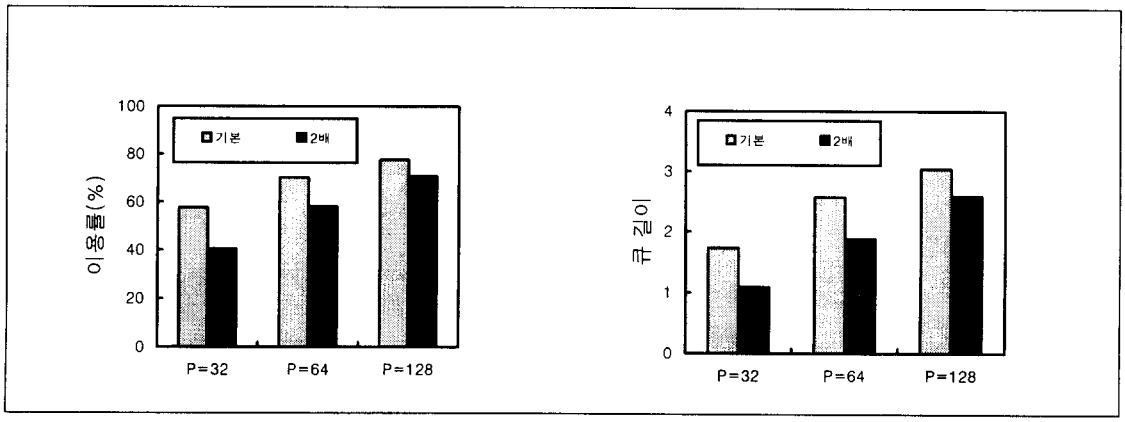
<그림 5> 전체 프로세서 수 증가에 따른 버스/링 이용률 및 큐 길이 변화



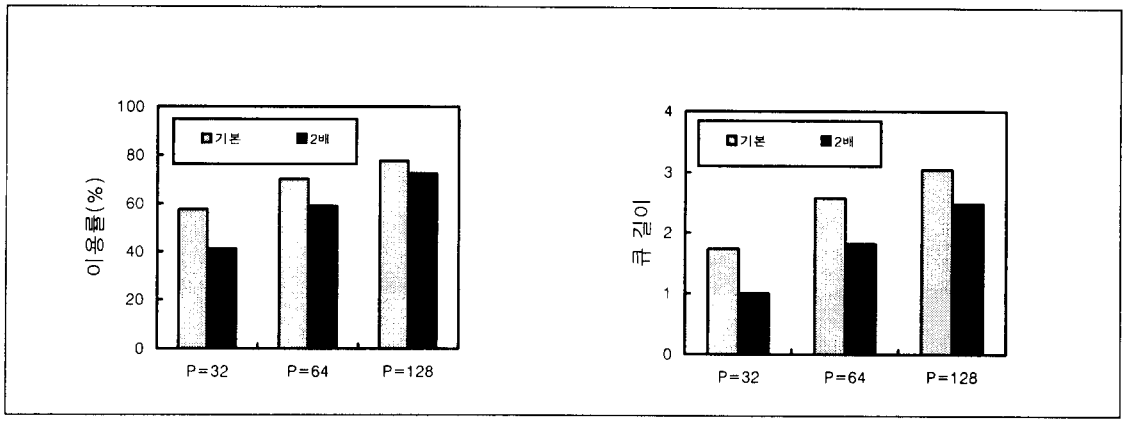
<그림 6> 캐쉬 접근 실패율 증가에 따른 버스/링 이용률 및 큐 길이 변화



<그림 7> PN내 프로세서 수 증가에 따른 버스/링 이용률 및 큐 길이 변화

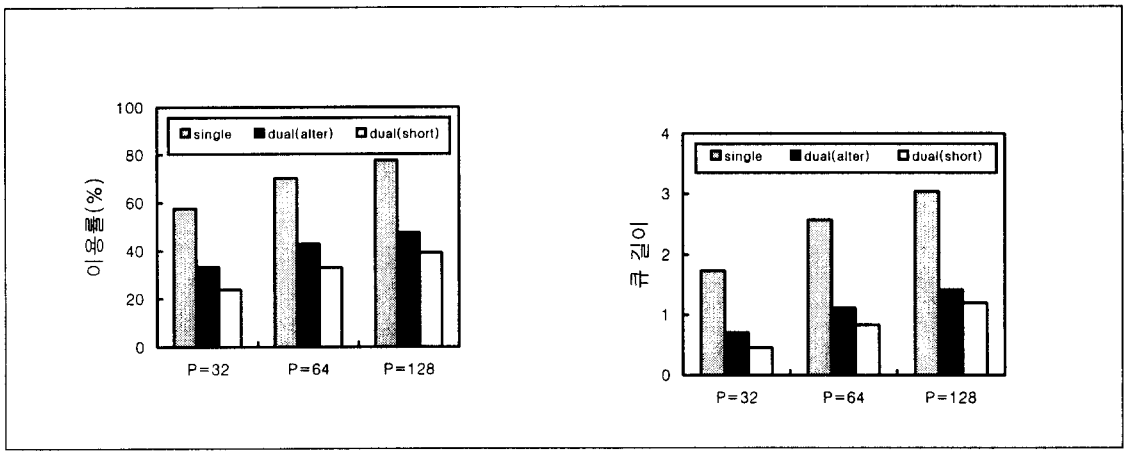


<그림 8> 링 속도 증가에 따른 링 이용률 및 큐 길이 변화



<그림 9> 링의 데이터 폭 증가에 따른 링 이용률 및 큐 길이 변화

이상의 실험에서 PN간 링에서 병목이 발생함을 알 수 있었는데, 이를 해소하기 위하여 링의 속도, 데이터 폭, 링의 개수를 각각 2배로 할 때 어떠한 결과가 나타나는지 살펴보자. <그림 8>과 <그림 9>에 나타난 바와 같이 링의 속도나 데이터 폭을 2배로 증가시키면 유사한 성능 개선을 기대할 수 있는데, 이용률과 큐 길이가 각각 7~42%, 18~73% 개선됨을 확인할 수 있다. 반면 링의 개수를 2배로 하면 링에서의 병목현상이 보다 효과적으로 해소될 수 있음을 알 수 있는데, <그림 10>에서 이용률은 약 64~142%, 큐 길이는 약 114~277% 개선되는 것으로 나타나 있다. 특히 dual ring에서는 sender에서 데이터를 전송할 때 2개의 링 중 하나를 선택해야 하는데, <그림 10>에서 "dual(alter)"란 2개의 링을 교대로 선택하는 간단한 방법을 나타내고, "dual(short)"는 sender에서 receiver로의 최단 경로를 제공하는 링을 선택하는 방법을 의미한다. 시뮬레이션 결과는 dual(short)가 dual(alter)에 비하여 병목현상을 약 54% 까지 더 해소시킬 수 있음을 보여준다.



<그림 10> 링의 수 증가에 따른 링 이용률 및 큐 길이 변화

6. 결론

본 논문에서는 상용 작업부하를 이용하여 다중프로세서 컴퓨터 시스템의 성능을 비교 분석하였다. 대상 시스템으로는 Illinois 프로토콜로 캐쉬 일관성을 유지하는 버스 기반 공유메모리의 PN들이 링으로 연결되어 분산공유메모리를 제공하는 CC-NUMA 방식의 다중프로세서 컴퓨터 시스템을 설정하였다. 작업부하로는 상용 분야의 대표적인 응용인 OLTP를 모델링하여 부과하였다.

먼저 상용 작업부하를 대상 시스템 시뮬레이터에 적용한 결과, PN내 버스의 평균 큐 길이 및 이용률이 매우 낮은 반면 PN간 링의 평균 큐 길이 및 이용률은 매우 높음을 알 수 있었다. 다음에 PN간 링에서의 병목점을 해소하기 위해 링의 속도 및 데이터 폭을 2배로 증가시켜 본 결과, 약 7~73%의 성능개선을 얻을 수 있었다. 또한 dual ring을 사용하는 것이 PN간 링에서의 병목점을 해소하는데 매우 효과적임을 확인할 수 있었는데, 시뮬레이션 결과 약 64~277%의 성능개선을 얻을 수 있었다. 끝으로 본 논문에서는 CC-NUMA 방식의 다중프로세서 컴퓨터 시스템에 OLTP용의 작업부하를 부과한 결과를 나타내었는데, Decision Support System(DSS) 등 다른 상용 응용을 모델링한 작업부하를 부과한 시뮬레이션도 흥미로운 연구분야가 될 것으로 기대된다.

참고문헌

[1] Culler, D., Singh, J., and Gupta, A., *Parallel Computer Architecture – A Hardware/Software Approach*, Morgan Kaufmann, Pub., 1998.

[2] Stenstrom, P., et al., "Boosting the Performance of Shared-Memory Multiprocessors," *IEEE Computer*, Vol. 30, No. 7(1997), pp. 63-70.

[3] Brorsson, M., et al., "The CacheMire Test Bench – A Flexible and Effective Approach for Simulation of Multiprocessors," *Proc. of Int'l Simulation Symp.*, 1993, pp. 41-49.

[4] Nanda, A. et al., "Coherence Controller Architecture for SMP-Based CC-NUMA Multiprocessors," *Proc. of Int'l Symp. On Computer Architecture*, 1997, pp. 219-228.

[5] Barroso, L. and Dubois, M., "Performance Evaluation of the Slotted Ring Multiprocessor," *IEEE Tr. on Computer*, Vol. 44, No. 7, 1995, pp. 878-890.

[6] *First Workshop on Computer Architecture Evaluation using Commercial Workloads*, HPCA-4, 1998.

[7] Barroso, L., Gharachorloo, K., and Bugnion, E., "Memory System Characterization of Commercial Workloads," *Proc. of Int'l Symp. on Computer Architecture*, 1998.

[8] Hahn, W., et al., "Evaluation of a Cluster-Based System for the OLTP Application," *ETRI Journal*, Vol. 20, No. 4(1998), pp. 301-326.

[9] Ki, A., "RACE Protocol: Remote Access Cache Coherency Enforcement Protocol," *Technical Report, ETRI*, 1998.

[10] Gray, J., *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, Pub., 1993.

[11] Schwetman, H., "CSIM: A C-based Process -oriented Simulation Language," *Proc. of '86 Winter Simulation Conf.*, 1986, pp. 387-396.

[12] 정용화, 박진원, 윤석한, "병렬컴퓨터 시스템 성능평가를 위한 컴퓨터 시뮬레이션 기술 동향," 「한국전자통신연구원 전자통신기술동향」, 13권, 5호(1998), pp. 1-10.

[13] Lee, K., et al., "Simulation Environment for Performance Evaluation of Commercial Applications," 「한국정보과학회 컴퓨터시스템연구회 추계학술대회 논문집」, 1998, pp. 21-32.

[14] Suggs, D. and Reynolds, R., "Constructing Multiprocessor Workload Characterizations," *Proc. of ACM Annual Southeast Conf.*, 1995, pp. 3-12.

[15] James, D., "The Scalable Coherent Interface: Scaling to High-Performance Systems," *Proc. of COMPCON'94*, 1994, pp. 64-71.

저자소개

박진원
1975년 서울대학교 산업공학과 졸업 (공학사)
1982년 Ohio State University 산업공학과 졸업 (M.S.)
1987년 Ohio State University 산업공학과 졸업 (Ph.D)
1977년~1980년 한국개발연구원
1988년~1999년 한국전자통신연구원 책임연구원
1999년~현재 영산대학교 컴퓨터정보공학부 교수
관심 분야 : 성능분석, 시뮬레이션, 컴퓨터구조

정용화
1984년 한양대학교 전자통신공학과 졸업 (공학사)
1986년 한양대학교 전자통신공학과 졸업 (공학석사)
1997년 University of Southern California 컴퓨터공학과 졸업(Ph.D)
1986년~현재 한국전자통신연구원 선임연구원
관심 분야 : 시뮬레이션, 컴퓨터구조, 병렬알고리즘

박종원
1981년 한양대학교 전자통신공학과 졸업 (공학사)
1983년 한양대학교 전자통신공학과 졸업 (공학석사)
1984년~현재 한국전자통신연구원 선임연구원
관심 분야 : 성능분석, 시뮬레이션, 컴퓨터통신

안희일
1973년 서울대학교 전자공학과 졸업 (공학사)
1996년 충북대학교 전자공학과 졸업 (공학석사)
1976년~1978년 한국과학기술연구소 연구원
1978년~1980년 한국전자기술연구소 연구원
1982년~현재 한국전자통신연구원 책임연구원
관심 분야 : 시뮬레이션, 컴퓨터구조, Genetic Algorithm

이강우
1985년 연세대학교 전자공학과 졸업 (공학사)
1991년 University of Southern California 컴퓨터공학과 졸업(M.S.)
1997년 University of Southern California 컴퓨터공학과 졸업(Ph.D)
1998년~현재 동국대학교 정보통신공학과 교수
관심 분야 : 컴퓨터구조, 시뮬레이션, 성능분석

김양우
1984년 연세대학교 전자공학과 졸업 (공학사)
1986년 Syracuse University 컴퓨터공학과 졸업(M.S.)
1992년 Syracuse University 컴퓨터공학과 졸업(Ph.D)
1992년~1996년 한국전자통신연구원 선임연구원
1996년~현재 동국대학교 정보통신공학과 교수
관심 분야 : 컴퓨터구조, 데이터베이스