메모리 상주 DBMS 기반의 OLTP 응용을 위한 다중프로세서 시스템 캐쉬 성능 분석

(Cache Performance Analysis of Multiprocessor Systems for OLTP Applications based on a Memory-Resident DBMS)

정 용 화 † 한 우 종 †† 윤 석 한 ††
(Yongwha Chung)(Woo-Jong Hahn)(Suk-Han Yoon)
박 진 원 ††† 이 강 우 †††† 김 양 우 †††††
(Jin-Won Park) (Kangwoo Lee) (Yang Woo Kim)

요 약 다중프로세서 시스템에 대한 대부분의 기존 연구는 과학계산용 응용을 중심으로 수행되어 왔으며, 또 다른 응용 분야인 상용 응용을 이용한 연구는 아직까지 초보 단계에 머물고 있는 실정이다. 이는 상용 DBMS의 소스 프로그램을 액세스하기가 쉽지 않으며, POSTGRES와 같은 공개된 소스 프로그램을 액세스 할 수 있더라도 컴퓨터 구조 설계자가 수십만 라인의 그 소스 프로그램을 이해하여 의미있는 성능 분석을 수행하기는 사실상 불가능하기 때문이다. 본 연구에서는 상용 응용을 이용하여 다중프로세서 시스템을 분석하기 위하여, SQL로 작성된 병렬 프로그램을 아키텍처 시뮬레이터 상에서 수행할 수 있는 EZDB라는 병렬 DBMS를 자체 개발하였다. EZDB가 POSTGRES와 다른점은 그 목적이 다중프로세서 시스템에서 상용 응용을 수행시키고 그 성능을 쉽게 분석할 수 있다는 점이다. EZDB의 유용함을 확인하기 위해, 본 논문에서는 다중프로세서 시스템에서 TPC-B 작업부하를 수행시켰을 때의 캐쉬 성능을 분석한다. 구축된 작업부하를 기반으로 프로그램 구동 시뮬레이션을 수행한 결과, 상용 응용에서 데이터 구조의 공유 특성이 매우 특별하며 국부성 및 작업 세트가 과학계산 응용의 경우와 매우 상이함을 확인하였다.

Abstract Currently, multiprocessors are evaluated almost exclusively with scientific applications. Commercial applications are rarely explored because it is difficult to obtain the source codes of commercial DBMS. Even when the source code is available, such as for POSTGRES, understanding the source code enough to perform detailed meaningful performance evaluations is a daunting task for computer architects.

To evaluate multiprocessors with commercial applications, we have developed our own DBMS, called EZDB. EZDB is a parallelized DBMS, loosely inspired from POSTGRES, and running on top of a software architecture simulator. It is capable of executing parallel programs written in SQL. Contrary to POSTGRES, EZDB is not intended as a prototype for a production-quality DBMS. Its purpose is to easily run and evaluate the performance of commercial applications on multiprocessor architectures.

To illustrate the usefulness of EZDB, we showed the cache performance data collected for the TPC-B benchmark on a shared-memory multiprocessor. The simulation results showed that the data structures exhibited unique sharing characteristics and that their locality properties and working sets were very different from those in scientific applications.

† 정 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
yongwha@computer.etri.re.kr
†† 비 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
wjhan@computer.etri.re.kr
shyoon@computer.etri.re.kr
††† 종신회원 : 영산대학교 컴퓨터정보공학부 교수
jinon@mail.ysu.ac.kr
†††† 비 회 원 : 동국대학교 컴퓨터정보통신학부 교수
klee@cakra.dongguk.ac.kr
††††† 정 회 원 : 동국대학교 컴퓨터정보통신학부 교수
ywkim@cakra.dongguk.ac.kr
논문접수 : 1999년 10월 27일
심사완료 : 2000년 5월 27일

1. 서 론

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)** 시스템은 대규모 계산이 수반되는 문제를 처리하기 위해 필요한 계산 능력을 효과적으로 제공하고 있으며 이러한 추세는 계속될 전망이다. 따라서 다중프로세서 시스템의 성능을 예측하기 위한 효과적인 방안을 모색할 필요성이 증대되고 있으며, 이러한 대규모 컴퓨터 시스템을 실제로 구현하기 전에 미리 제안된 아키텍처의 특성을 평가해 보는 것은 매우 중요한 일이다. 그러나, 다중프로세서 아키텍처를 갖는 컴퓨터 시스템은 순차적인 컴퓨터 시스템에 비하여 모델링하기가 훨씬 어려운데, 이는 아키텍처와 병렬 프로그램 동작간의 매우 복잡한 상호작용 때문이다.

일반적으로 공유메모리(shared memory) 다중프로세서 시스템은 단일 주소 공간을 제공하여 매우 편리한 프로그래밍 환경을 제공한다. 이러한 공유메모리 다중프로세서 시스템에서 높은 성능을 제공하기 위해서는 공유 데이타(shared data)에 대한 액세스 패턴을 이해하는 것이 필수적이다. 이러한 의미에서 과학계산 응용은 과학계산용 벤치마크[2,3]가 개발된 이후로 충분히 이해되었다고 할 수 있다. 그러나 컴퓨터 아키텍처 설계자에게 또 다른 응용인 **상용 응용(commercial applications)**에 대한 동작 특성은 아직 충분히 이해되지 못하고 있다. 이는 다중프로세서 시스템을 평가하기 위해 필요한 DBMS의 소스 프로그램을 일반적으로 액세스하기 어렵기 때문이다. 설사 *POSTGRES*[4]와 같은 공개 프로그램을 입수한다 하더라도 복잡한 데이타 구조, 록킹(locking) 방법, 수십만 라인의 코드는 컴퓨터 아키텍처 설계자가 성능을 평가하는데 어려움을 겪는다. 따라서 컴퓨터 아키텍처 설계자에게 POSTGRES와 같은 DBMS 소스를 연구 툴로 활용하는 것은 매우 부담스러운 일이다.

이러한 이유로 다중프로세서 시스템의 상용 응용 분야 작업부하를 이용한 성능 연구가 별로 이루어지지 않았다. 다만 최근들어 몇 편의 연구결과[5-8]가 발표되었는데, 예를 들어 [6]에서는 분포 구동 방식을 이용하여 클러스터 시스템을 성능을 분석하였고, [7]에서는 Oracle DBMS와 AltaVista 탐색 엔진을 이용한 다중프로세서 시스템 시뮬레이션 결과가 발표되었다. [8]에서는 POSTGRES DBMS를 이용하여 다중프로세서 시스템의 메모리 성능을 분석하였다. 그러나, 이러한 성능 결과를 컴퓨터 아키텍처 설계자가 보다 쉽게 추출하기 위해서는 보다 간단한 DBMS 프로그램의 개발이 필요하다.

본 연구에서는 이러한 목적에서 컴퓨터 아키텍처 분석용 DBMS를 자체 개발하였다. 즉, EZDB라는 Structured Query Language(SQL)로 작성된 병렬 프로그램을 수행 할 수 있는 병렬 DBMS를 개발하였다. 이는 POSTGRES 처럼 메모리 상주[9], 관계형 DBMS이고, 운영체제 기능을 사용하지 않는다. 또한 EZDB는 소프트웨어 시뮬레이터상에서 동작하여, 시뮬레이트되는 아키텍처상에서 수행될 때 발생하는 모든 상황이 트레이스될 수 있다. 현재 기본 모듈의 설계가 완료되어 TPC-B 벤치마크와 같은 간단한 상용 응용을 수행시킬 수 있다. EZDB의 목적은 완벽한 DBMS를 제공하는 것이 아니라, 각각의 공유 데이타에 대한 메모리 동작 특성과 같은 다중프로세서 시스템의 성능에 관한 자세한 정보를 제공하기 위함이다. 본 논문에서는 먼저 EZDB의 구조와 설계에 대하여 언급하고, TPC-B 응용을 이용하여 여러 가지 메모리 동작 특성을 분석한다. 특히, Oracle이나 POSTGRES 등의 실제 DBMS를 이용한 기존의 시뮬레이션 결과[7,8]에서는 아직 발표되지 않은 각각의 데이타 구조에 대한 캐쉬 성능을 분석하였다. 즉, 캐쉬 크기, 블록 크기, 프로세서 수를 변화시키면서 각각의 데이타 구조에 대한 캐쉬 미스 영향을 분석하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 EZDB에 대해서 간단히 설명하고, 3장에서는 TPC-B 벤치마크 프로그램을 서술한다. 4장에서는 시뮬레이션 환경과 분석 대상 시스템에 대해 설명하며, 시뮬레이션 결과를 5장에 기술하고, 마지막 6장에서 결론을 맺는다.

2. EZDB

일반적으로 데이타베이스 시스템내의 데이타베이스는 데이타 레코드의 집합이다. 데이타베이스 동작은 데이타베이스내의 레코드를 어떻게 정의하고 생성하고 취급하는지를 지정한다. DBMS는 여러 가지 데이타베이스 제약을 만족시키면서 데이타베이스 레코드에 대한 데이타베이스 동작을 수행한다. 이때 두 종류의 사용자가 있을 수 있는데, 첫째는 일반 사용자이고 두번째는 DataBase Administrator(DBA)이다. DBA는 메타데이타에 포함된 데이타베이스 정의를 직접 액세스 할 수 있는 특권을 가지고 있다. EZDB에서는 스크립트 파일(script file)을 통해서 제공되는 일반 사용자의 질의(query)만을 시뮬

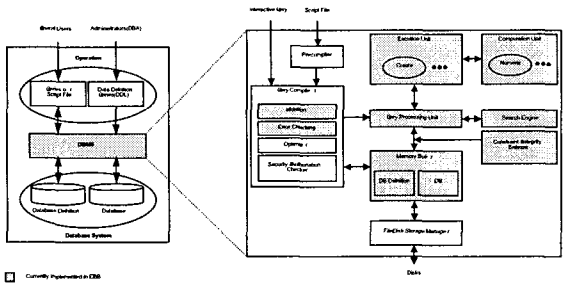

그림 1 자체 개발 DBMS인 EZDB의 전체 모습

레이트한다. <그림 1>에 나타낸 EZDB의 구조에서 회색부분이 현재 구현 완료된 부분이다.

<그림 1>에 나타난 것과 같이, 연속적인 질의로 구성된 스크립트 파일을 사용자가 작성하면 질의 컴파일러가 이를 컴파일한다. 그렇게 컴파일된 프로그램은 질의 처리기(Query Processing Unit), 실행기(Execution Unit), 계산기(Computation Unit), 탐색 엔진(Search Engine) 등에 의해 수행된다. 여기서 데이타 구조는 메타데이타(metadata), 데이타베이스 데이타, 인덱스(index) 데이타를 포함한다.

자체 개발한 EZDB에 대해 자세히 살펴보자. 여기서 질의문, 질의 명령어, 질의 동작은 각각 스크립트 파일에 나타난 프로그램문, 질의문에서 추출된 SQL과 유사한 명령어, 질의 명령어내의 활동을 의미한다. 일반적으로 질의 명령어는 테이블 이름, 조건, 계산 종류, 질의 결과에 대한 임시저장소 등을 지정하기 위해 몇 개의 질의 변수를 포함한다.

2.1 스크립트 파일

스크립트 파일은 일련의 질의문이 포함된 C 프로그램으로 *gcc*에 의해 컴파일된다. 예를 들어, <그림 2>의 2번째 줄에서 내재함수 PQexec()에 의하여 하나의 질의문이 DBMS로 전달되고, 질의 컴파일러에 의하여 컴파일된다.

```
for (i=0; i<N; i++)
  PQexec(INSERT INTO table_A VALUES (:i,0););
```

그림 2 스크립트 파일의 예

2.2 질의 컴파일러

DBMS에서는 일반적으로 3개의 컴파일러가 필요하다. 하나는 DBA 질의에 사용되는 Data Description Language(DDL)를 위한 것인데, EZDB에서는 포함되지 않았다. 다른 컴파일러는 사용자 스크립트 파일에 있는 프로그래밍 언어를 위한 것으로, EZDB에서 스크립트 파일은 C로 작성되었으며 gcc로 컴파일된다. 또한, 스크립트 파일에 있는 질의문을 질의 변수를 갖는 질의 명령어로 변환하기 위하여 질의 컴파일러를 개발하였다. 이 컴파일러는 검증을 위해 메타데이타에 저장된 정보를 액세스한다. 질의 컴파일러의 결과는 질의를 실제로 수행하는 질의 처리기로 넘어간다.

2.3 질의 처리기

EZDB의 핵심인 질의 처리기는 질의 명령어를 수행하기 위하여 질의 동작을 조율한다. 즉, 질의 동작의 순서뿐만 아니라 메타데이타, 데이타베이스 데이타, 인덱스 데이타의 접근도 조정한다. 일반적으로 하나의 질의 명령어와 질의 변수들이 주어졌을 때 질의 처리기는 다음을 수행한다.

- 데이타베이스 테이블, 레코드, 속성에 대한 정보를 획득하기 위하여 메타데이타를 먼저 액세스한다.
- 생성, 삽입, 삭제와 같은 질의 명령어에 대응하는 처리기 내의 프로그램 모듈을 호출한다.
- 질의 변수의 조건을 만족시키는 레코드의 위치를 신속히 확인하기 위하여 탐색 엔진을 구동한다.
- 테이블내의 데이타베이스 레코드에 대해 순차 탐색을 수행한다.
- 데이타베이스 데이타를 검색하거나 수정하기 위하여 데이타베이스 데이타를 직접 액세스한다.
- 질의 명령어 종류에 따라 인덱스 데이타 구조에 적절한 노드를 삽입 또는 제거한다.

2.4 메모리 버퍼

일반적으로 메타데이타와 데이타베이스는 디스크에 저장된다. 그러나 EZDB에서는 이들이 메인메모리의 메모리 버퍼에 저장된다고 가정한다. 또한, 메모리 버퍼는 EZDB 프로그램 모듈에서 필요한 사유 또는 임시 변수를 위해 사용된다. 그리고 EZDB는 모든 데이타를 메모리에 저장하기 때문에 운영체제 기능이 시뮬레이션 중에 필요하지 않으며, 시뮬레이션 환경은 사용자 프로그램의 활동을 탐지할 수 있다.

메타데이타 자체는 각각의 데이타베이스를 묘사하는 데이타를 저장하고 있는 작은 데이타베이스이다. 그것은 개념 데이타베이스 스키마(schema), 내부 스키마, 외부 스키마, 여러 레벨의 스키마간의 매핑 등을 묘사한다. 메타데이타는 주로 질의 처리에서 액세스된다. 그러나 질의 컴파일러는 메타데이타를 검증 절차에서 사용하고, 무결성 집행기(Constrain and Integrity Enforcer)는 데이타베이스를 올바른 상태로 유지하기 위하여 메타데이타를 사용한다.

<그림 3>에 EZDB에서 정의한 메타데이타를 나타내었다. TableHead는 데이타베이스 내에 포함된 테이블

no_of_tables
3

(a) TableHead

table_num	table_name	num_recs	num_fields	num_primary_keys	primary_key_fields
0	table-1	10	3	1	pkey-1
1	table-2	50	4	1	pkey-2

(b) TableTable

table_number	no_of_fields	data type 1	field 1	data type 2	filed 2	data type 3	field 3	..
0	3	int	pkey-1-1	float	xxx	char	..	-
1	4	int	pkey-1-2	int	yyy	float	..	-

(c) RecordHead

그림 3 EZDB에서 메타데이타

의 수를 저장한다. 새로운 테이블이 생성될 때 마다 이 숫자는 하나씩 증가한다. TableTable은 테이블 이름의 리스트, 각 테이블의 레코드 수, 각 레코드의 필드수를 포함하며, 기본키(primary key)를 구성하는 필드의 수와 이러한 필드의 리스트를 포함한다. RecordHead는 테이블의 필드 이름과 데이타 종류를 포함하고, 각 테이블의 레코드 수를 저장한다.

delete bit	field 1 (bid)	field 2	field 3	field 4
0	1	105000	350-390	-
1	2	200100	350-370	-

그림 4 EZDB에서 데이타베이스 데이타

데이타베이스는 레코드를 포함하는 테이블의 집합이다. 각각의 레코드는 몇 개의 속성(attribute)으로 구성되는데, 테이블의 레코드는 어레이(array)에 저장되고 사용자의 요청에 의하여 검색되거나 수정된다. RecordTable은 레코드를 포함하는 데이타베이스 테이블인데, <그림 4>에 나타낸 바와 같이 삭제 비트를 포함한다. 따라서 하나의 레코드가 삭제되면 이 비트가 세트되지만, 실제로 레코드의 제거는 DBMS 종료시 발생한다.

2.5 인덱스 탐색 엔진

데이타베이스 시스템의 기본 태스크는 탐색 동작이다. 이를 위하여 어떤 탐색 조건에 대한 레코드 검색을 신속히 수행하기 위하여 인덱스 자료 구조를 사용한다. EZDB에서는 동적으로 변하는 복수 레벨의 인덱스를 구현하기 위하여 B-트리 구조를 이용한다. 이때 B-트리의 수는 테이블의 수와 동일하며, B-트리 내의 노드의 수는 테이블내의 레코드 수와 동일하다.

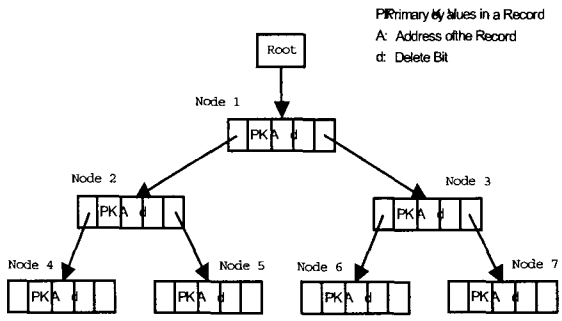

그림 5 B-트리

2.6 실행기

실행기(Execution Unit)는 질의 명령어를 수행하는데, 필요에 따라 산술, 부울(Boolean), 관계, 집합 동작을 위한 계산기(Computation Unit)을 사용한다. 메타데이타와 데이타베이스 데이타에 저장된 정보는 질의 처리기에서 제공한다. 질의 명령어의 결과는 임시 저장소에 저장된 후 질의 처리기로 전달된다. EZDB에서 실행기는 표준 SQL과 유사한 모든 명령, 즉, 삽입(insert), 삭제(delete), 갱신(update)뿐만 아니라 커서(cursor), 접근(fetch), 그룹(group-by), 순서(order-by) 등의 복잡한 명령어도 수행할 수 있다.

2.7 무결성 집행기

EZDB는 데이타베이스를 올바른 상태로 유지하기 위해 필요한 여러 가지 제약 및 무결성을 지원한다. 먼저 트랜잭션은 "원자성(atomic)"을 가져야 한다. 모든 트랜잭션은 부분 수행된 동작에 의하여 데이타가 갱신되지 않음을 보장해야 한다. 이를 위하여 각각의 트랜잭션은 다른 트랜잭션에 의하여 선점(preempt)됨이 없이 하나의 프로세서에 의해 완전히 수행되어야 한다. "일치성(consistency)"은 트랜잭션에 대한 임의의 수행이 데이타베이스의 상태를 어떤 일치된 상태에서 일치된 다른 상태로 변경하여야 한다. 이를 위하여 질의 컴파일 단계에서 먼저 검증 절차를 수행하고 데이타 무결성을 확인한다. "고립화(isolation)" 성질은 병렬 트랜잭션이 순차적으로 수행되었을 경우와 동일한 결과를 생성하는 것을 요구한다. 병렬 트랜잭션 수행은 다음 장에서 언급한다. "지속성(durability)"을 보장하기 위해서는 시험 시스템이 완료된 트랜잭션의 효과를 유지해야 되고, 어떠한 하드웨어 실패에 의한 복구 후에 데이타 일관성을 보장해야 된다. 이러한 모든 요구사항들은 EZDB에서 적절히 만족된다.

3. TPC-B

본 논문에서는 **On-Line Transaction Processing** (OLTP) 응용을 타겟으로 한 Transaction Processing Council(TPC) 벤치마크[10]를 상용 응용의 예로 가정한다. TPC 벤치마크는 데이타베이스 시스템의 성능을 비교하는데 널리 사용되는 표준 벤치마크 프로그램이다. 본 논문에서는 TPC-B를 수행시킨 성능을 분석한다. 사실 공식적으로는 TPC-C가 TPC-B를 대체하였지만, TPC-B는 아직도 다중프로세서서 시스템을 분석하는데 자주 이용된다[6,7,11]. 이는 TPC-B가 구현이 용이하고, 프로세서/메모리 관점에서 볼 때 그 동작 특성이 TPC-C와 유사하기 때문이다.

3.1 TPC-B 모델

TPC-B는 뱅킹 시스템을 모델링 한 것으로, 각각의 트랜잭션은 임의로 선택된 계정의 잔고를 수정한다. 즉, Tid, Bid, Aid, Delta가 주어졌을 때, <그림 6> (a)에 나타낸 트랜잭션이 수행된다. <그림 6> (b)는 병렬화된 스크립트 파일을 나타낸다. TPC-B에서 Teller, Branch, Account, History 테이블이 사용되는데, Account, Teller, Branch 각각이 포함하는 레코드 수의 비율은 100,000:10:1 이다. Teller 레코드는 적어도 100Byte의 크기를 갖으며 Tid, Bid, Teller_Balance 필드를 갖는다. Branch 레코드도 적어도 100Byte의 크기를 갖으며 Bid, Branch_Balance 필드를 갖으며, Account 레코드는 100Byte 이하의 크기를 갖으며 Aid, Bid, Account_Balance 필드를 갖는다. 마지막으로, History 레코드는 50Byte 이하의 크기를 갖으며 Aid, Tid, Bid, Delta, Time_Stamp 필드를 갖는다.

이제 기본 동작이 메타데이타와 인덱스데이타를 이용하여 어떻게 수행되는가를 살펴보자. Bid와 Delta가 주어졌을 때, Branch 레코드의 Branch_Balance 필드는 Delta 만큼 증가된다. TableHead로 부터 데이타베이스 시스템의 테이블 수를 확인하고, 이를 이용하여 TableTable의 탐색공간을 정의한다. TableTable 레코드의 table_name 필드가 Branch와 비교되어, 테이블 수, 레코드 수, 필드 수, 기본키에 대한 정보를 입수한다. 그리고 RecordHead에 의하여 Bid와 Delta의 데이타 형태가 점검된다. Bid가 기본키이기 때문에, Branch에 대한 B-트리를 이용하여 인덱스 탐색이 이루어진다. 그리고 B-트리로부터 주어진 Bid에 대한 Branch 레코드의 메모리 위치가 찾아진다. Branch_Balance와 Delta의 합은 계산기에 의해 계산되고 Branch_Balance 필드에 쓰여진다.

3.2 병렬화

병렬 트랜잭션 처리 시스템을 시뮬레이트하기 위하여, EZDB와 스크립트 파일의 병렬화가 필요하다. 본 논문에서는 이러한 병렬성을 제공해 주는 런타임 시스템(runtime system)으로 **ANL 매크로**[12]를 가정하였다. 일반적으로 공유메모리 시스템에서 병렬 프로그램을 동작시키기 위해서는 세가지 메커니즘, 즉 프로세스 생성, 공유 변수 선언, 공유 데이타 액세스의 동기 등이 필요하다. 이러한 메커니즘을 이용하여 어떻게 병렬화를 수행하는지를 살펴보기 위해 <그림 6>에 스크립트 파일과 병렬 스크립트 파일을 나타내었다. 여기서 Bid, Tid,

```
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
```

(a) A transaction TPC-B

```
Main() {
  MAIN_INITENV();
  Initialize_by_master();
  For (i=0; i<P; i++) CREATE(Slave);
  BARRIER;
  Parallel_Section();
  BARRIER;
}
Slave() {
  BARRIER;
  Parallel_Section();
  BARRIER;
}
Parallel_Section() {
  WHO_AM_I(&Pid);
  Start = Num_Tr/P*Pid;
  End = start + Num_Tr/P;
  For (i=start; i<end; i++) {
    Get_Input(&Bid,&Tid,&Aid,&Delta);
    /* Update Account */
    Pqexec(comm);
        .
        .
        .
  }
}
```

(b) A parallelized script file

그림 6 트랜잭션과 병렬화된 스크립트 파일의 예

Aid, Delta는 임의로 생성된 입력 변수이다.

ANL 매크로 모델을 따라, 먼저 마스터 프로세스가 초기화를 수행한다. 그리고 여러 개의 슬레이브 프로세스를 생성하고, 자신을 포함한 모든 프로세스가 병렬 구역을 수행한다. 각각의 프로세스는 병렬 트랜잭션 처리를 위하여 동일한 코드 이미지를 수행한다. 이때 사유 변수 start와 end는 각 프로세스에서 수행할 트랜잭션 번호를 지정한다. 또한, 초기화 중에 메타데이타, 인덱스 데이타, 데이타베이스 데이타는 공유 데이타로 선언되어, 추후에 다른 프로세스에서 이들을 액세스할 때 모든 갱신이 정확히 반영되어 있어야 한다. 이러한 목적에서 사용한 ANL 매크로는 G_MALLOC이다. 그외의 모든 변수는 사유 데이타(private data)로 간주한다. 또한, 시스템이 올바른 상태를 유지하기 위하여 공유 데이타 접근은 동기화 되어야 한다. EZDB에서는 모든 공유 데이타에 대하여 이진록(binary lock)을 설정한다.

병렬화된 스크립트의 for 루프에서 트랜잭션의 네개 질의 중 하나를 나타내었다. DBMS는 PQexec()를 통하여 P개의 프로세스로부터 여러 개의 질의를 접수할 수 있으며, 이러한 질의는 2장에서와 언급한대로 컴파일되고 수행된다. EZDB는 질의내 병렬성은 지원하지 않는다. 병렬 질의 처리를 위하여, ANL 매크로 LOCK/UNLOCK이 공유 데이타 접근 앞 뒤로 삽입된다.

4. 시뮬레이션 분석 환경 및 분석 대상 시스템

<그림 7>에 본 논문에서 사용한 시뮬레이션 환경을 나타내었다. 아키텍처 시뮬레이터는 인스트럭션과 메모리 접근을 생성하기 위해 병렬 응용을 수행할 수 있는 **CacheMire**[13]를 사용하였다. CacheMire는 스크립트 파일을 동작시키는 EZDB의 코드를 수행하고 메모리 접근을 생성한다. 그리고 수행이 진행되면서 인스트럭션 접근, 사유 데이타 및 공유 데이타 액세스, 동기화 동작 등이 추적된다.

또한, 본 논문에서 분석 대상으로 한 아키텍처 모델은 캐쉬 일관성 유지 다중프로세서 시스템이다. 캐쉬는 Least Recently Used(LRU) 교체 방침의 4-way, set-

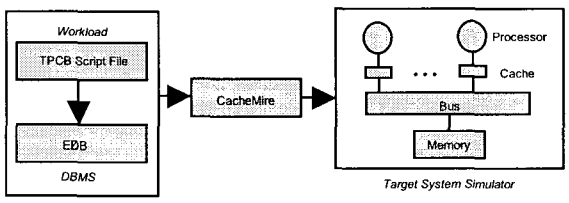

그림 7 시뮬레이션 환경

표 1 시뮬레이션 변수 값

시뮬레이션 변수	값
프로세서 수	2, 4, 8, 16, 32
캐쉬 크기	32KByte, 64KByte, 128KByte, 256KByte, 512KByte, 1MByte
캐쉬 블록 크기	8Byte, 16Byte, 32Byte, 64Byte, 128Byte, 256Byte, 512Byte, 1024Byte
메모리 크기	1GByte
페이지 크기	4KByte

associative 방식으로 구성되며, 데이타 일관성은 일리노이 프로토콜을 따른다. 또한, 본 논문에서 수행한 시뮬레이션에서는 모든 데이타베이스 데이타가 메타데이타, B-트리 등과 함께 초기에 메인메모리에 저장된 것으로 가정하여 공유 데이타 구조에 대한 액세스를 추적한다. 따라서 입출력 활동이 포함되지 않으며, 인스트럭션 접근과 운영체제 프로세스의 데이타 접근도 배제하였다. 마지막으로 200개의 트랜잭션을 수행시켜 캐쉬를 워밍업하면, 워밍업 후에는 모든 메타데이타, B-트리 등이 캐쉬에 존재한다. 본 논문에서 설정한 시뮬레이션 변수의 값을 <표 1>에 요약하였다.

5. 시뮬레이션 결과

앞 장에서 언급한 시뮬레이션 환경을 이용하여 다중프로세서 시스템의 캐쉬 성능을 비교 분석하였다. 먼저 본 연구에서 개발한 EZDB의 타당성을 검증하기 위하여, EZDB를 이용한 TPC-B 수행시 메모리 접근 횟수를 기존에 발표된 결과와 비교하였다. 그리고 무한 크기의 캐쉬를 갖는 시스템에서 블록 크기와 프로세서 수의 변화에 따른 캐쉬 성능을 분석하였다. 마지막으로 프로세서 수를 고정시킨 상태에서 캐쉬 크기와 블록 크기를 변화시키면서 캐쉬 성능을 분석하였다.

표 2 공유 데이타 구조의 특성

공유 데이타		크기	엑세스 종류	엑세스 빈도
메타 데이타	TableHead	4Byte×1	읽기	11.5%
	TableTable	26Byte×4	읽기/쓰기	9.2%
	RecordTable	22Byte×4	읽기	10.8%
데이타 베이스	Branch	196Byte×4	읽기/쓰기	1.4%
	Teller	196Byte×40	읽기/쓰기	1.6%
	Account	196Byte×400,000	읽기/쓰기	2.2%
	History	196Byte×1,000	읽기/쓰기	13.5%
인덱스 데이타	Branch	20Byte×4	읽기	2.5%
	Teller	20Byte×40	읽기	6.0%
	Account	20Byte×400,000	읽기	20.5%
	History	20Byte×1,000	읽기/쓰기	20.8%

본 연구에서 수행한 시뮬레이션에서, 모든 데이타베이스 테이블의 레코드 크기는 196Byte, B-트리 내의 각 노드는 20Byte의 크기를 가지며, 모든 공유 데이타는 100MByte 이하의 메인메모리에 저장된다. 이러한 공유 데이타 구조의 특성을 <표 2>에 나타내었다. 또한, History의 엔트리 수는 1,000을 넘지 않도록 설정하였으며, 가장 큰 데이타 구조는 400,000개의 엔트리를 갖는 Account이다.

5.1 메모리 접근 횟수

<표 3>에 EZDB를 이용한 트레이스에서 추출한 TPC-B 트랜잭션의 평균 액세스 횟수를 나타내었다. 즉, 트랜잭션당 34,258개의 인스트럭션 접근이 발생하는데, 이는 기존에 발표된 TPC-B의 통계수치들과 일치한다. 예를 들어, Data General에서 Oracle DBMS로 TPC-B를 수행시키면 트랜잭션당 약 35,000개의 사용자 인스트럭션 접근이 발생하는 것으로 발표되었다[14].

표 3 TPC-B 트랜잭션의 평균 액세스 횟수

인스트럭션 접근	사유 데이타		공유 데이타		
	읽기	쓰기	읽기	쓰기	록
34,258	2,063	291	261	41	51

5.2 무한 크기 캐쉬

무한 크기의 캐쉬를 갖는 시스템에서 각각의 데이타 구조에 대한 블록 크기별 캐쉬미스 수를 <그림 8>에 나타내었다. 이때 각 그래프내의 다섯개 곡선은 상이한 프로세서 수에 대한 미스 수이다. 먼저, 블록 크기 및 프로세서 수에 따른 영향을 살펴보자. TableHead나 RecordHead에 대한 접근에서는 미스가 발생하지 않는데, 이는 이러한 데이타 구조가 읽기 전용이고 시뮬레이션 초기에 캐쉬가 이미 워밍업 되었기 때문이다. 따라서 본 논문에서는 이러한 데이타 구조를 Branch B-트리와 더불어 더 이상 고려하지 않는다.

먼저 <그림 8>의 전체 미스율을 살펴보면, 작은 블록 크기에서는 공간적 국부성(spatial locality)에 따른 이득을 확인할 수 있다. 그러나 큰 블록 크기에서는 미스율이 증가하고, 이는 거짓 공유(false sharing)가 두드러짐을 알 수 있다. 이러한 거짓 공유는 TableTable, History 및 그것의 B-트리 접근에 의해 발생된다. 다음에 각각의 데이타 구조에 대하여 살펴보자.

가장 특이한 메타데이타는 TableTable이다. 즉, 질의와 데이타베이스 테이블명이 주어졌을 때, 레코드 수와 레코드내의 필드의 수를 확인하기 위하여 TableTable에 있는 레코드를 탐색한다. TableTable의 레코드 수는 삭제 또는 삽입 동작에서 갱신된다. TPC-B 트랜잭션에서는 네개의 질의에 의해 TableTable이 네번 탐색된다. History는 트랜잭션에서 레코드가 삽입되는 유일한 테이블이고, 이것이 연속된 질의에서 TableTable 접근을 할 때 캐쉬 미스를 야기한다. 그리고 TableTable의 크기가 104Byte이기 때문에 거짓 공유에 의한 미스의 수가 64Byte와 128Byte 크기의 블록 사이에서 가장 급격히 증가하지만, 그 이상에서는 상수로 남아있다.

데이타베이스 테이블에서는 Branch와 Teller가 작은 크기를 갖는데, 한 트랜잭션에서 b_id 또는 t_id가 갱신된다. 이러한 테이블내의 레코드 크기가 196Byte이기 때문에, 196Byte 보다 작은 블록은 미스의 수에 영향을 주지 못하며 모든 미스는 참 공유(true sharing) 미스이다. 또한, Branch에는 네개의 레코드만이 있기 때문에 큰 블록의 경우에 Teller 보다 Branch에서 더 많은 거짓 공유가 발생한다. 그러나, 전체적으로는 Branch나 Teller 접근에 의한 미스는 많지 않다. 또한, 하나의 트랜잭션은 400,000개의 Account 중에서 하나의 레코드만을 액세스하므로, Account 접근이 캐쉬에서 히트 될 확률은 거의 영에 가깝다. 따라서, 대부분의 경우에는 블록 크기에 상관없이 하나의 트랜잭션에서 하나의 초기 미스(cold miss) 만이 발생한다. History의 경우에는 하나의 레코드가 생성되어 테이블에 삽입되지만, 다시 액세스 되지는 않는다. 삽입의 경우, 196Byte가 채워질 때 초기 미스를 유발하며, 큰 블록은 이러한 초기 미스 수를 감소시킨다. 그러나, 블록의 크기가 256Byte를 초과하면 거짓 공유가 주요 원인이 된다.

하나의 트랜잭션 중에 질의의 레코드 위치를 찾아내기 위해 B-트리 내의 여러 노드를 액세스하는데, Branch, Teller, Account의 B-트리는 읽기 전용이다. TPC-B의 각 질의는 History 레코드의 기본키 필드를 갱신하고 그것의 B-트리를 수정한다. B-트리의 메모리 동작을 이해하기 위하여, <그림 8>에 나타낸 것과 같은 완벽하게 균형잡힌 B-트리를 가정하자. 비록 트리의 상위 부분에 있는 노드의 수는 적지만, 그들의 접근 빈도는 하위 부분에 있는 것보다 훨씬 높다. 더구나, 상위 부분에 있는 노드들은 실제 메모리상에서 근접하게 위치하여, 큰 블록은 탐색 동작에서 한번에 많은 수의 보다 유용한 노드들을 캐쉬로 읽어 들인다. 이러한 이유로 Branch, Teller, Account의 B-트리는 큰 블록에서 적은 미스를 나타낸다. 특히 Teller의 B-트리 크기가 800Byte 밖에 되지 않기 때문에, 1024Byte 캐쉬 블록에 포함된다. History의 B-트리 접근은 거짓 공유를 발생시키고, 큰 블록에 대해서 미스가 증가한다.

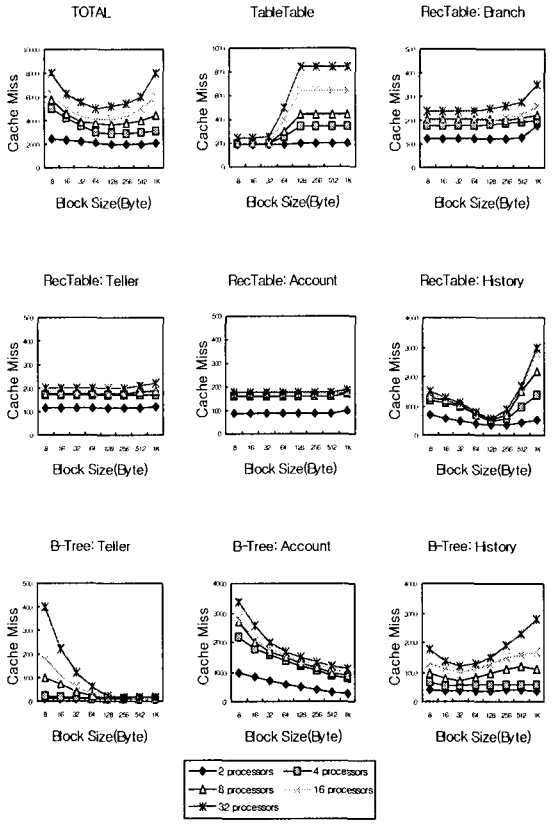

그림 8 무한 크기를 갖는 캐쉬에서 블록 크기 및 프로세서 수의 영향

과학계산 응용의 경우, 프로그램 초기에 공유 데이타가 분할되어 프로세서에 할당된다. 따라서 적절한 데이타 분할과 할당으로 데이타 통신을 최소화 할 수 있다. 그러나 TPC-B에서는 임의의 프로세서가 공유 데이타의 임의의 부분을 액세스할 수 있다. 따라서 대부분의 쓰기 공유 블록에 대한 소유권(ownership)은 프로세서 간에 빈번히 이동한다. 그리고 <그림 8>에 나타낸 모든 데이타 구조의 미스 수는 프로세서 수에 크게 영향을 받는다.

이러한 프로세서 수의 영향은 TableTable, History, B-트리 보다는 Branch, Teller, Account에서 적게 나타난다. 그 이유는 Branch, Teller, Account에서 하나의 변수가 갱신되고 교환되는 반면, TableTable과 History의 B-트리에서는 하나의 변수가 갱신되지만 한 트랜잭션 중에서 그 변수가 여러 번 교환되기 때문이다. 그리고 History에서는 전체 196Byte가 갱신되어 보다 많은 미스를 유발한다.

5.3 유한 크기 캐쉬

이제 캐쉬의 크기가 32KByte에서 1MByte 사이인 경우에 대하여 살펴보자. 이때 프로세서의 수는 4개이며, 블록 크기는 변수이다. 예를 들어, <그림 9>에서 TPC-B의 전체 미스율을 최소화하는 블록의 크기는 128Byte 정도임을 알 수 있다. 또한, 전체 미스율은 캐쉬 크기에 별로 영향을 받지 않는데, 이는 32KByte 정도의 작은 캐쉬도 TPC-B에 충분함을 의미한다.

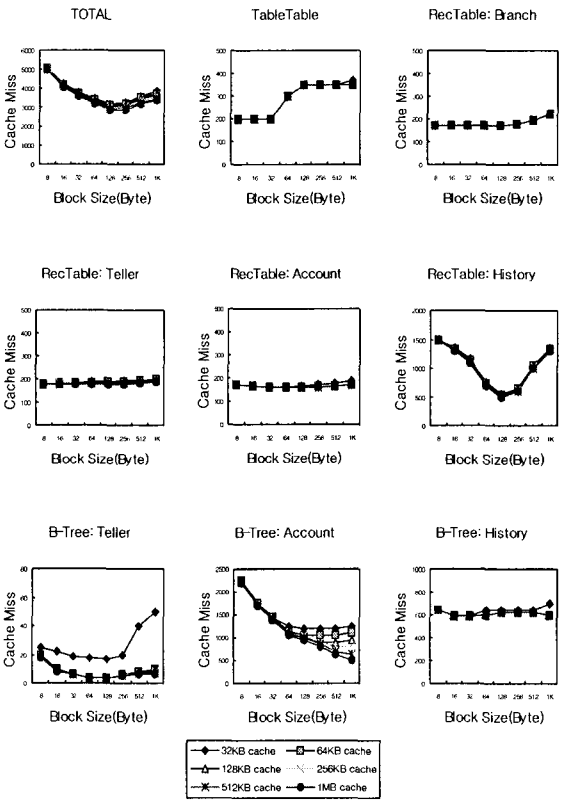

그림 9 유한 크기를 갖는 캐쉬에서 블록 크기 및 캐쉬 크기의 영향

<그림 9>에 나타낸 각 데이타 구조별 캐쉬 크기의 영향을 설명하기 위해 데이타 구조를 세개의 그룹으로 구분한다. 먼저 메타데이타와 두개의 데이타베이스 테이블, 즉, TableTable, Branch, Teller는 크기가 작고 빈번히 액세스된다. 따라서 이러한 데이타 구조는 간혹 무효화 될 뿐이고 캐쉬의 크기는 전체 미스율에 커다란 영향을 주지 못한다. 두번째 데이타 구조는 Account와 History로 시간적 국부성(temporal locality)이 거의 없으며, 대부분의 교체에 대한 원인을 제공하지만 캐쉬 크

기에 거의 영향을 받지 않는다. 마지막으로 캐쉬 미스율이 캐쉬 크기에 민감한 유일한 데이타 구조는 B-트리, 특히 Account의 B-트리이다. 대규모 B-트리 접근에 대한 시간적 국부성은 전체 트리를 통하여 변하는데, 트리의 정점에서는 캐쉬 크기에 상관없이 국부성도 높고 히트율도 높다. 그러나 트리의 하부로 내려갈수록 국부성은 낮아진다. 반면 공간적 국부성은 좋은데, 주어진 캐쉬 크기에서 블록 크기에 따라 미스율이 지속적으로 감소함을 알 수 있다. 그러나, 주어진 블록 크기에 대해서 블록의 수가 감소하면 그 블록이 두개의 연속된 접근 사이에서 교체될 확률이 높다.

6. 결 론

기존의 DBMS에서 상용 응용의 동작 특성을 추출하는 것은 간단한 작업이 아니다. 따라서 대부분의 다중프로세서 시스템에 대한 기존 연구가 과학계산 응용을 대상으로 하고 있으나, 현실적으로 많은 다중프로세서 시스템이 상용 응용을 대상으로 하기 때문에 이에 대한 연구가 필요하다.

이러한 문제점을 해결하기 위하여 본 연구에서는 컴퓨터 아키텍처 분석용의 DBMS를 자체 개발하였다. 즉, EZDB는 다중프로세서 시스템에서 데이타베이스 응용을 동작시키고 평가할 수 있는 플랫폼을 제공한다. 또한, 본 논문에서는 이를 기반으로 TPC-B 작업부하를 동작시키고, 여러 캐쉬 크기/블록 크기/프로세서 수에 따른 국부성/작업 세트/캐쉬 미스 등의 특성을 분석하였다. 전체적으로 프로세서 수가 증가함에 따라 캐쉬 성능이 블록 크기에 영향을 받고, 캐쉬 크기는 데이타베이스와 메타데이타에 큰 영향을 주지않는다. 반면, 데이타베이스에는 큰 레코드 크기에 따른 공간적 국부성이 존재하여, 과학계산 응용에 비하여 더 큰 블록 크기를 필요로 한다.

EZDB의 현재 버전이 정확한 동작을 하지만, 질의 및 코드 최적화를 위하여 EZDB에 컴파일러 모듈을 추가하고, B-트리를 B+-트리나 B*-트리로 대체하여 인덱스 데이타 구조를 개선할 예정이다. 끝으로 본 논문에서는 TPC-B 작업부하를 부과한 성능 분석 결과를 기술하였는데, 개발된 DBMS를 활용하여 TPC-C나 TPC-D 등 다른 작업부하나 Cache Coherent Non-Uniform Memory Access(CC-NUMA) 등 다른 다중프로세서 시스템 구조에 대한 캐쉬 성능 분석도 컴퓨터 아키텍처 설계자에게 흥미로운 연구분야가 될 것으로 기대된다.

참 고 문 헌

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

[2] J. Singh, W. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared Memory," *Computer Architecture News*, Vol. 20, No. 1, pp. 5-44, 1992.

[3] C. Wang, C. Wang, and K. Hwang, "STAP Benchmark Evaluation of Three Massively Parallel Processors," *Proc. of Int'l Conf. on Parallel and Distributed Computing Systems*, 1997.

[4] M. Stonebraker, L. Rowe, and M. Hirohama, "The Implementation of POSTGRES," *IEEE Tr. on Knowledge and Data Engineering*, Vol. 2, pp. 125-142, 1990.

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

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

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

[8] P. Trancoso, et al., "The Memory Performance of DSS Commercial Workloads," *Proc. of Int'l Symp. On High-Performance Computer Architecture*, 1997.

[9] H. Garcia-Molina and K. Salem, "Main Memory Database Systems: An Overview," *IEEE Tr. on Knowledge and Data Engineering*, Vol. 4, No. 6, pp. 185-202, 1992.

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

[11] J. Nilsson and F. Dahlgren, "Improving Performance of Load-Store Sequences for Transaction Processing Workloads on Multiprocessors," *Proc. of Int'l Conf. On Parallel Processing*, pp. 246-255, 1999.

[12] J. Boyle, et al., *Portable Programs for Parallel Processors*, Holt, Rinehart, and Winston, Inc., 1987.

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

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

정 용 화

1984년 한양대학교 전자통신공학과 졸업(공학사). 1986년 한양대학교 전자통신공학과 졸업(공학석사). 1997년 University of Southern California 컴퓨터공학과 졸업(Ph.D). 1986년 ~ 현재 한국전자통신연구원 책임연구원 관심분야는 컴퓨터구조, 병렬알고리즘, 성능분석

한 우 종

1981년 고려대학교 전자공학과 졸업(공학사). 1984년 고려대학교 전자공학과 졸업(공학석사). 1995년 고려대학교 전자공학과 졸업 (공학박사). 1985년 ~ 현재 한국전자통신연구원 책임연구원 관심분야는 컴퓨터구조, 마이크로프로세서 구조, 멀티미디어 서버

윤 석 한

1977년 고려대학교 전자공학과 졸업(공학사). 1986년 한국과학기술원 전자계산학과 졸업 (공학석사). 1995년 고려대학교 전자공학과 졸업 (공학박사). 1977년 ~ 현재 한국전자통신연구원 책임연구원 관심분야는 컴퓨터구조, 마이크로프로세서 구조, 멀티미디어 서버.

박 진 원

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

이 강 우

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. 관심분야는 컴퓨터구조, 시뮬레이션, 성능분석

김 양 우

1984년 연세대학교 전자공학과 학사. 1986년 Syracuse Univ. 컴퓨터공학과, 석사. 1992년 Syracuse Univ. 컴퓨터공학과, 박사. 1992년 6월 ~ 1996년 8월 한국전자통신연구원 선임연구원. 1996년 9월 ~ 현재 동국대학교 컴퓨터 정보통신학부 교수. 관심분야는 병렬처리 컴퓨터 구조, 멀티미디어 컴퓨팅 서버시스템, 정보검색 및 데이터베이스 전용 시스템