Sharing Pattern Analysis of an OLTP Application[+]

이 강 우[*], 김 희 철[**]
(Kangwoo Lee[*], Hiecheol Kim[**])

요 약 최근 다중 프로세서 시스템들이 상용 데이터베이스의 트랙잭션 처리에 널리 사용되고 있지만 데이터베이스의 데이터 공유 패턴에 대한 탐구는 거의 이루어지고 있지 않다. 본 논문에서는 캐쉬 일관성을 갖는 공유 메모리 구조의 다중 컴퓨터 상에서 대표적인 OLTP 응용 프로그램임 TPC-B 벤치마크에 대한 데이터 공유 패턴을 분석하였다. 또한 데이터 공유 패턴이 갖는 성능에 대한 영향을 분석하기 위하여 프로세서 개수, 캐쉬 크기, 그리고 캐쉬 블록 크기를 변화시켜 성능을 측정하였다. 이러한 성능 측정 결과로서 TPB-B는 과학기술 분야의 응용 프로그램과는 확연히 다른 데이터 공유 특성을 갖고 있음을 확인하였다.

Abstract Although multiprocessor systems are widely used in recent years to run commercial workloads, data sharing patterns are rarely explored due to several difficulties. In this paper, we made in-depth sharing pattern analysis for a representative OLTP application, the TPC-B benchmark, running on a cache-coherent shared-memory multiprocessor system. In addition, to illustrate their effects on the performance, the number of cache misses were measured for various numbers of processors, cache sizes and cache block sizes. From these measurements, we found out the shared data in TPC-B largely bear quite different sharing characteristics from those in scientific applications.

1. Introduction

Understanding the access patterns of shared data is the key to obtaining good performance on shared-memory systems. In this respect, scientific applications are fairly well understood. However, in these years, since parallel systems are mostly used for commercial applications, researchers in architecture are eager to explore the behavior of parallel commercial workloads. Unfortunately, DBMS programs are usually proprietary. Although the POSTGRES95 [11] was released, the large number of complex data structures, the complicated locking schemes and very large size of the code make it difficult to exploit for performance evaluations.

Due to these reasons, relatively few results have been published for commercial applications. Suggs and Reynolds [12] presented the instruction fetch statistics of TPC-B [7] workloads measured on the Motorola 88110. Torrellas et al [14] and Bhandarkar [1] evaluated commercial workload for the SGI multiprocessor and a DEC Alpha AXP system. In [9], the performance of a shared-nothing architecture is investigated using synthetic workload and actual traces. Shared-nothing architectures are preferred to shared-memory systems in [5]. The role of IO subsystems in shared-memory systems was investigated in [13]. In [15], the authors used POSTGRES95 to show the memory performance of some queries in TPC-D for various cache models on a shared-memory system.

To obtain more results more easily, we have developed our own DBMS called EZDB [8]. EZDB is a parallelized DBMS loosely inspired from POSTGRES95, which is able to execute parallel programs written in SQL. Like POSTGRES95, EZDB is an in-memory

+ This work was supported by 1999 Research Fund from Daegu University.
* Professor, School of Computer and Communication Engineering, Dongguk University.
** Professor, School of Computer and Communication Engineering, Daegu University.

relational database and makes no use of operating system facilities. EZDB runs on top of a software simulator and thus all events happening during the execution on a simulated architecture can be traced.

Under the simulation environment including EZDB, we ran our simulation to investigate the data sharing pattern of TPC-B. The cache size, cache block size and the number of processors are changed to see their effects on the number of cache misses for individual data structure separately.

This paper is organized as follows. EZDB and TPC-B are briefly described in Sections 2 and 3, respectively. Section 4 presents the simulation methodology and simulation results are presented in Section 5 along with detailed analysis. We close this paper with concluding remarks in Section 6,

2. EZDB

The architecture of the EZDB built based on the general DBMS is shown in Figure 1. The black boxes indicate the components implemented in EZDB; the components in white boxes are not implemented. Note that EZDB is not intended as a complete, usable database system, but rather its purpose is to provide researchers in architecture with detailed performance information including the memory behavior of each individual shared data structure.

Table 1 summarizes the implementation detail of EZDB. Note the operating system normally is in charge of file IO and memory management. But, since EZDB keeps all data in memory and our simulation environment can only monitor the activities of user programs, no operating system activity is included in our simulation. For more detail on EZDB, see [8]

3. OLTP Benchmark: TPC-B

The TPC benchmarks are standard benchmarks widely used in industry to compare the performance of database systems. The simplest among them is TPC-B whose logical database model is given in Figure 2. The banking transaction defined in TPC-B benchmark is given below.

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

4. Simulation Methodology

Figure 3 3illustrates our simulation method. The simulation environment is the CacheMire testbench [3] which executes parallelized applications to generate instructions and memory references. CacheMire executes the code of EZBD running the script file and generates memory references. Instruction fetches, private and shared data accesses and synchronization operations (test-and-set) are monitored on the fly as the execution progresses. The target architecture model is a cache-coherent single-bus shared-memory multiprocessors with up to 32 processors. The caches are 4-way set associative with the LRU (least-recently-used) replacement policy. Data coherence is enforced by the Illinois protocol.

In our simulation, all the database data are brought initially into the main memory along with the metadata and B-Trees so that we may track the accesses to shared data structures. Therefore, IO activities are not simulated. In addition, the instruction fetches and data accesses of operating system processes are not included.

In our methodology, we collect a global trace on a simulator of a four processor system which services 200 TPC-B transactions with four branches, 40 tellers and 400,000 accounts. The trace is then broken down into 200 small transaction traces, one for each transaction.

When the transaction traces are applied to a particular configuration, we first warm up the caches by running the 200 transactions. After this initial run, all metadata structures, B-Trees, and small tables are in the main memory and the caches are filled. The 200 trace files are then applied again and measurements are taken. We simulate configurations of up to 32 processors. Cache size varies from 32 KBytes to 1 MBytes. We have looked at cache block size between 8 and 1024 bytes. The main memory in our simulation is 1 Gbytes with 4 KByte pages.

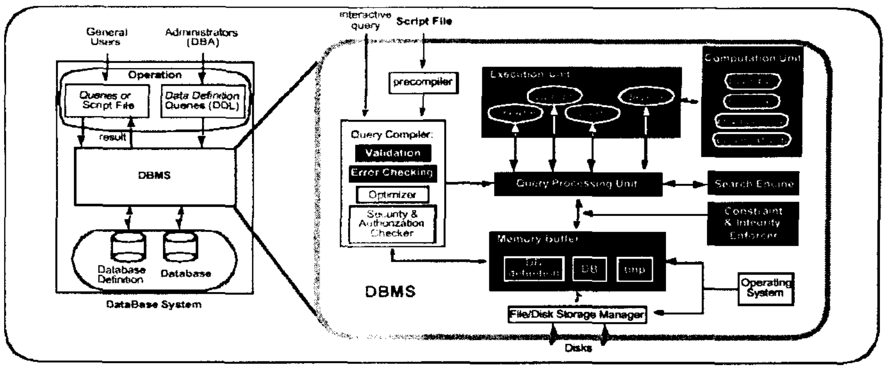

Figure 1. Architecture of EZDB.

Table 1. Architecture of EZDB.

Items		General DBMS	EZDB
Level of Users	Users	DBA & General Users	General users
	Query Language	DDL & General Query Language	Our own SQL
	Special Command	Privileged Commands for DBA	Not supported
Query Types		Interactive & Script File	Script File
Compilers	DDL Compiler	DDL Compiler	No
	Precompiler	Host machine-dependent	gcc
	Query Compiler	Validation	Implemented
		Error Checking	Implemented
		Query Optimizer	Not implemented
		Security & Authorization	Not implemented
Query Processing	Query Execution	create, insert, delete, select, update, group, order, *etc.*	Implemented
	Computation	numeric, boolean, relational, aggregation	Implemented
Data Structures	Data Definition	Metadata	Implemented
	Index Data	B-tree, B^+-tree or B^*-tree	B-tree
	Database Data	Relational or Object-oriented	Relational
Constraints & Integrity		Enforced	Enforced
Storages	Disk Storage	Hard Disk	Memory Buffer
	Disk Manager	Operating System	No
	Memory Storage	Memory	Memory Buffer
	Memory Manager	Operating System	No

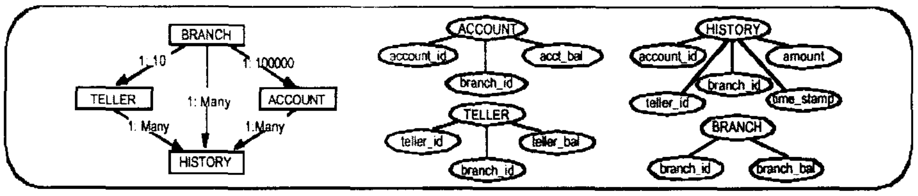

Figure 2. Basic database model in TPC-B.

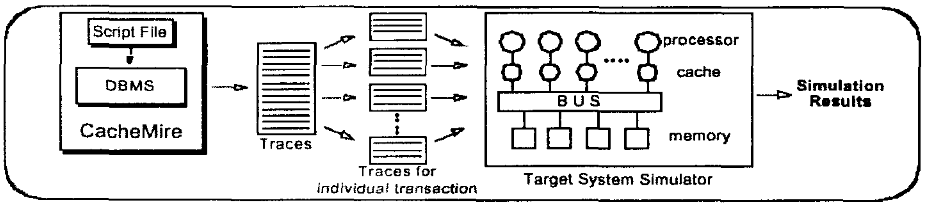

Figure 3. Overview of simulation methodology

5. Sharing Analysis

This section provides in-depth analysis of data sharing behavior of TPC-B benchmark.

5.1. Sizes of Data Structures

The records in all database tables are 196 byte long and each node in B-Trees holds 20 bytes. All shared data are stored in less than 100 MBytes of main memory. Table 2 shows the sizes of all shared data structures.

5.2. Memory Access Counts

Table 3 shows the measurements obtained for average transaction in the traces we collected. There are 34,258 instruction fetches per transaction. This number correlates well with previous evaluations of TPC-B. For example, Oracle on a Data General AViiON AV/8500 server generated about 35,000 user instruction fetches per TPC-B transaction [12]. Table 4 shows the number of data accesses to each metadata, database data and index data for the 200 transactions.

The accesses to metadata structures are mostly reads. In particular, TableHead and RecordHead are read-only in TPC-B. In TableTable one entry is updated once whenever a History record is inserted resulting in a total of 200 writes. One single field (id) of a record in each of Branch, Teller and Account is updated. By contrast, in History, all fields are written by insert operations and the number of writes dominates. Since B-Trees are updated only when a record is inserted or deleted in a table the B-trees in TPC-B are read-only except for the B-tree of History which is read-write.

5.3. Data Sharing (Infinite Caches)

Each graph in Figure 44 shows the number of misses for one data structure in a system with infinite caches as a function of block size. The five curves in each graph correspond to systems with different number of processors. In the following we comment on the effects of the block size and the number of processors. Accesses to TableHead and RecordHead experience no miss because they are read-only and the caches are warm at the beginning of the simulation. Thus, we ignore these data structures along with the Branch B-Tree and private data.

5.3.1. Block Size

Looking at the curves for the total misses in Figure 4, we observe some gains due to spatial locality for smaller blocks, but, for larger blocks, the number of misses increases, a sign that false sharing dominates. False sharing is due to accesses to TableTable, History and B-Tree of History. Now we discuss the accesses to data structures, one after another.

The only interesting metadata is TableTable. Given a query and a database table name, the records in TableTable are searched to find the number of records of the given table name and the number of fields in a record. During a delete or insert operation the number of the records of the table is updated in TableTable. In a TPC-B transaction, there are four queries and, thus, TableTable is searched four times per transaction. History is the only table where one record is inserted in a transaction and this causes cache misses on accesses to TableTable in consecutive queries. Because the size of TableTable is 104 bytes the number of false sharing misses grows dramatically for block sizes of 64 and 128 bytes, but remains constant for larger blocks.

Among the tables in database, Branch and Teller are the smallest ones. One integer variable, B_id or T_id, of one record in Branch or Teller, respectively, is modified once in a transaction. Since the size of each record in these tables is 196 bytes, a block size smaller than 196 bytes does not affect the number of misses and all misses are true sharing misses. Larger blocks will cause false sharing misses. We observe more (false) sharing misses in Branch than in Teller for larger blocks because there are only four records in Branch. Overall, the number of misses on accesses to Branch and Teller is small. Because there are 400,000 Account records and only one record is accessed (modified) in a transaction, the probability that an access to Account record hits in a cache is almost zero. Thus, most of times, one (cold) miss is counted in a transaction independently of the block size. In the case of History, a new record is crea-

Table 2. Shared data structures and their sizes

Data Structures	Metadata			Database Data				Index Data			
	TableHead	TableTable	RecordHead	Branch	Teller	Account	History	Branch	Teller	Account	History
Size (bytes x items)	4 x 1	26 x 4	220 x 4	196 x 4	196 x 40	196 x 400K	196 x 1K	20 x 4	20 x 40	20x 400K	20 x 1K

Table 3. Global access counts in the average TPC-B transaction

Compile Time			Transaction Processing					
Instruction fetch	Private Data		Instruction fetch	Private Data		Shared Data		
	Read	Write		Read	Write	Read	Write	Lock
195571	18039	5966	34258	2063	291	261	41	51

Table 4. Access counts for individual data structures in 200 TPC-B transactions

Data	MetaData			Synch.	Record Table				Search Tree (B-Tree)			
	Table-Head	Table-Table	Record Head		Branch	Teller	Account	History	Branch	Teller	Account	History
Read	5795	4589	5594	-	432	542	834	545	1306	3136	10675	9603
Write	-	200	-	-	322	317	317	6459	-	-	-	1194
Lock	-	-	-	10176	-	-	-	-	-	-	-	-

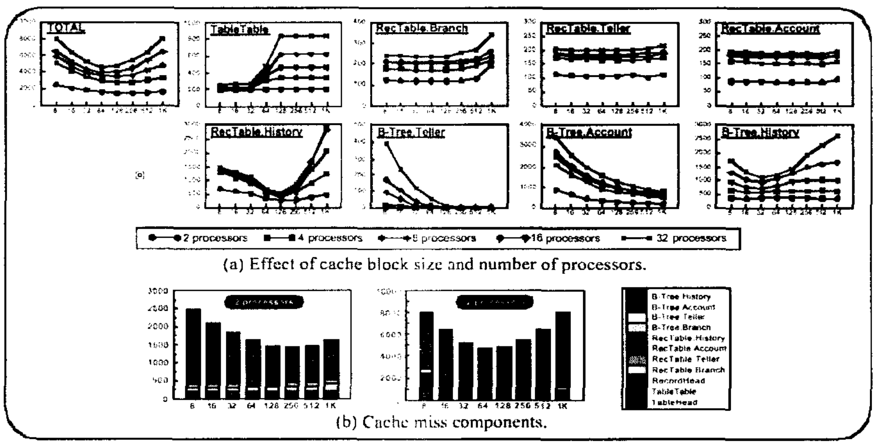

Figure 4. Effects of block size and number of processors in infinite cache.

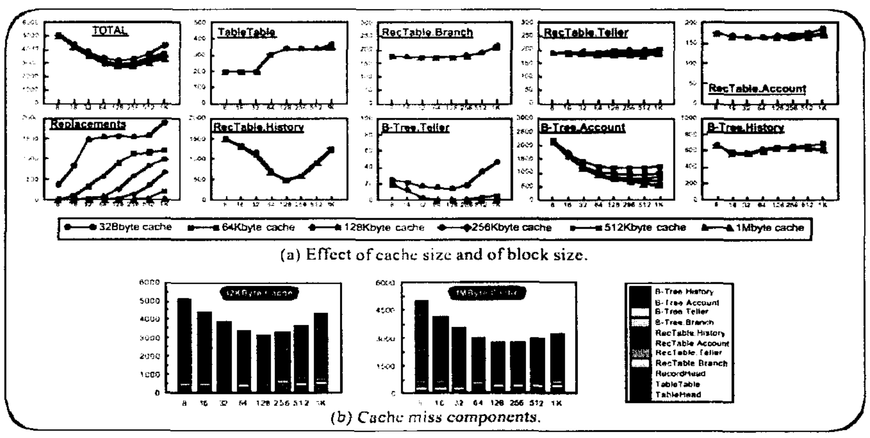

Figure 5. Effects of block size and cache size in finite cache.

ted and inserted into the table in each transaction and is never accessed again. On an insertion, a 196 byte record is filled, causing cold misses. Larger blocks reduce the number of cold misses. However, if the block is larger than 256 bytes, false sharing takes over.

During a transaction, multiple nodes in B-Trees are accessed to locate the record in a query. The B-Trees of Branch, Teller and Account are read-only. Each query in TPC-B updates the primary key field of a record of History, and thus its B-Tree is updated. To understand the memory behavior of B-Trees, consider a perfectly balanced B-Tree4. Although the number of nodes in the upper portion of the tree (near the root) is small, they are accessed more frequently than the nodes in the lower levels. Furthermore, the nodes in higher levels reside closely to each other in physical memory. Therefore, in a search operation, larger blocks bring more useful nodes in the upper portion of the tree into a cache. For this reason the B-trees of Branch, Teller and Account have fewer misses for larger blocks. In particular, since the size of the B-Tree of Teller is only 800 Bytes, it fits into a 1024 byte cache block. Accesses to the B-Tree for History cause false sharing misses. In consequence, the number of misses ends up increasing for large block sizes.

5.3.2. Number of Processors

In scientific applications, shared data structures are partitioned and allocated among processors at the initial stage of a program. Careful data partition and allocation can minimize data communication. By contrast, in TPC-B, any processor can access any portion of the shared data. Therefore, most write-shared blocks are migratory, i.e. the ownership of these cache blocks is transferred among processors very often. Hence the number of misses on every data structure in Figure 4 4 is drastically affected by the number of processors.

The effect of the number of processors is less pronounced in Branch, Teller and Account than in TableTable, History and B-Trees. The reason is that a single integer variable is modified and communicated in the first three data structures. On the other hand, even though one single integer variable is modified in TableTable and the B-Tree of History, the variables are communicated multiple times during a transaction. Finally, in History, a whole 196 byte record is modified causing more misses.

5.4. Finite Cache Effects

We now consider finite cache sizes between 32 KBytes and 1 MBytes. The number of processors is four and the block size is variable. The graphs in Figure 45 show that the block size minimizing the total misses of TPC-B is around 128 bytes, for all cache sizes. Moreover, the number of total misses is not very sensitive to the cache size, implying that a small data cache of 32Kbytes is sufficient for TPC-B.

To explain the cache size effects observed in Figure 54 on each individual data structure, we group data structures into three classes. The first class includes the metadata plus two database tables, i.e. TableTable, Branch and Teller. These data structures are all small and accessed frequently. Therefore, they are rarely victimized for replacements and the cache size has little effect on their miss rate. The second class of data includes Account and History and have little or no temporal locality. Hence the number of misses is not affected by the cache size, even though they are the cause of most replacements. The only data structures whose number of misses is sensitive to the cache size are the B-Trees, especially the B-Tree of Account. The temporal locality on accesses to a large B-Tree varies across the tree. At the top of a B-Tree, the locality is high and the hit rate is high independently of the cache size. The locality decreases for lower levels of the tree. The spatial locality is good and, for a given cache size, we observe a constant decrease of the number of misses with the block size. However, for a given block size, the chances that a block is displaced in between two consecutive accesses is higher when the number of blocks decreases.

Table 5. Sharing characteristics of shared data.

		Size (bytes x items)	Accesses		Locality		Sensitivity to		
			Type	Number (%)	Spatial	Temporal	Cache Size (32K to 1M)	Num.Procs (2 to 32)	Block Size (8 to 1K)
Metadata	TableHead	4 x 1	RO	11.52	1 field	4	-	-	-
	TableTable	26 x 4	R/W	9.23	4 records	4	low	high	high
	RecordHead	220 x 4	RO	10.78	4 records	1	-	-	-
Database Data	Branch	196 x 4	R/W	1.45	1 field	1	low	low	low
	Teller	196 x 40	R/W	1.65	1 field	1	low	low	low
	Account	196 x 400K	R/W	2.21	-	1/4K	low	low	low
	History	196 x 1K	R/W	13.50	-	-	low	very high	very high
Index Data	Branch	20 x 4	RO	2.51	few nodes	1/4 to 1	-	-	-
	Teller	20 x 40	RO	6.04	few nodes	1/40 to 1	high	high	high
	Account	20 x 400K	RO	20.58	few nodes	1/400K to 1	high	high	high
	History	20 x 1K	R/W	20.81	few nodes	0 to 1	high	very high	very high

5.5. Summary: Locality

Table 5 summarizes the memory behavior of all shared data structures in TPC-B. The spatial locality is defined as the amount of data in contiguous address range accessed at a time. The temporal locality is defined as the average number of the sequences of accesses to a particular memory location per transaction. The working sets of TableTable, Branch and Teller tables are so small and they presented little sensitivity to the changes of the cache size (Figure 5). Since the Account and History tables bear no locality, they presented negligible differences as the cache size varies. Finally, a few nodes in the upper portion of B-Trees form a working set and their sensitivities to the cache size is significant as shown in Figure 5.

6. Conclusions

Obtaining reliable measurements for commercial application behavior on existing DBMS is, for many reasons, a very difficult task, which explains why practically all evaluations of computer architectures done today are based on scientific applications. This situation is deplorable since most machines today are designed to run commercial applications. To address this problem and make progress in understanding commercial applications, we have developed our own parallelized DBMS, which we call EZDB. EZDB provides an easy-to-use platform to run and evaluate database programs on multiprocessor architectures.

We have applied the tool to the TPC-B benchmark and have presented the unique characteristics of each individual shared data structure including the localities, working sets and cache misses over various cache sizes, block sizes and numbers of processors. Overall, the cache size does not play a critical role for the database and metadata. On the other hand, some parts of the B-Trees show good utilization of caches. Some spatial locality also exists in the data base because of the large record size. In consequence, the cache block size under TPC-B is suggested to be larger than that used for scientific applications.

References

[1] Cvetanovic, Z and Bhandarkar, D., Characterization of Alpha AXP Performance Using TP and SPEC Workloads, Proc. of the 21st Annual International Symposium on Computer Architecture, Apr. 1994.

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

[3] Brorsson, M., Dahlgren, F., Nilsson, H., and Stenstrom, P., The CacheMire Test Bench—A Flexible and Effective Approach for Simulation of

Multiprocessors, Proc. of 26th Ann. IEEE International Simulation Symposium, Apr. 1993.

[4] Cormen, T. H., Leiserson, C. E. and Rivest, R. L., Introduction to Algorithms, McGrow Hill Book Co., 1989.

[5] DeWitt, D. and Gray, J., Parallel Database Systems: The Future of High Performance Database Systems, Communications of the ACM, 35(6):85-98, Jun. 1992.

[6] Elmasri, R. and Navathe, S. B., Fundamentals of Database Systems, 2nd Edition, The Benjamin/Cummings Publishing Co. Inc., 1994.

[7] Gray, Jim, The Benchmark Handbook for Database and Transaction Processing Systems, Second Edition, Morgan Kaufmann, 1993.

[8] Lee, K., Kim, Y., Chung, Y. and Park, J., Simulation Environment for Performance Evaluation of Commercial Applications, Proc. of KISS, Sept. 1998.

[9] Marek, Robert, and Rahm, Erhard, Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems, Proc. of PARLE, May 1992.

[10] Singh, J. P., Weber, W-D, and Gupta, A., SPLASH: Stanford Parallel Applications for Shared-Memory, Computer Architecture News, 20(1): 5-44 March 1992.

[11] Stonebraker, M. and Kemnitz, G., The POSTGRES Next generation Database Management System, Communications of the ACM, June, 1986.

[12] Suggs, D. and Reynolds, R., Constructing Multiprocessor Workload Characterizations, Proc. 33rd ACM Annual Southeast Conference, Clemson, March 1995.

[13] Thakkar, Shreekant and Sweiger, M., Performance of an OLTP Application on Symmetry Multiprocessor System, Proc. of the 1990 Int. Computer Architecture Symposium.

[14] Torrelas, J., Gupta, A. and Henessey, J., Characterizing the Caching and SynchronizationPerformance of a Multiprocessor Operating System, Proc. of the 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Oct. 1992.

[15] Trancoso, P., Larriba-Pey, J., Zhang, Z. and Torrelas, J., The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors, Proc. of 23rd Ann. Int. Symposium on Computer Architecture, 1996.

이 강 우 (Kangwoo Lee)

1985: Yonsei University (Electronic Eng. B.S.)
1991: University of Southern California (Computer Eng. M.S.)
1997: University of Southern California (Computer Eng. Ph.D.)
1998 ~ : Dongguk University, Assistant Professor Interest : GRID, Computer Architecture, Parallel Processing, Compiler

김 희 철 (Hicheol Kim)

1983: Yonsei University (Electronic Eng. B.S.)
1991: University of Southern California (Computer Eng. M.S.)
1996: University of Southern California (Computer Eng. Ph.D.)
1997 ~ : Daegu University, Assistant Professor
Interest : GRID, Computer Architecture, Parallel Processing, Compiler