論 文

# Performance Analysis of Futurebus+ based Multiprocessor Systems with MESI Cache Coherence Protocol

Seok Bum Ko*, In Kon Kang**, Sung Woo Park***,
Young Chon Kim** *Regular Members*

# MESI 캐쉬 코히어런스 프로토콜을 사용하는 Futurebus+ 기반 멀티프로세서 시스템의 성능평가

正會員 高 錫 范* 正會員 姜 寅 坤** 正會員 朴 聖 宇*** 正會員 金 永 川**

ABSTRACT

In this paper, we evaluate the performance of a Futurebus+ based multiprocessor system with MESI cache coherence protocol for four bus transaction types. Graphical symbols and compiler of SLAM II are used in modeling and simulation. A steady-state probability of each state for MESI protocol is computed by a Markov chain. The probability of each state is used as an input value for a correct simulation. Processor utilization, memory utilization, bus utilization, and the waiting time for bus arbitration are measured in terms of the number of processors, the hit ratio of cache memory, the probability of read operation, memory access time, the number of memory modules, the probability of internal operation, and bus bandwidth.

요 약

본 논문에서는 MESI 캐쉬 코히어런스 프로토콜을 사용하는 Futurebus+를 시스템 버스로 갖는 멀티프로세서 시스템에 대하여 4 종류의 버스 트랜잭션에 따라 시스템의 성능을 평가하였다. 성능 평가를 위한 모델링과 시뮬레이션은 SLAM II 그래픽 심볼과 컴파일러를 이용하였다. 정확한 시뮬레이션을 위하여 해석적 방법으로 MESI 프로토콜의 각 상태에 대한 확률을 구하였고, 구한 확률 값은 시뮬레이션의 입력으로 사용하였다. 시뮬레이션에서는 프로세서의 수, 캐쉬 메모리의 히트율, 읽기 명령을 수행할 확률, 메모리 액세스 시간, 메모리 모듈의 수, 프로세서가 내부 동작을 수행할 확률, 버스의 밴드 폭에 따른 프로세서의 이용률, 메모리의 이용률, 버스의 이용률, 버스 중재 대기 시간 등을 구하였다.

*韓國通信
**全北大學校 컴퓨터工學科
***한남大學校 情報通信工學科
論文番號 : 93-183

## I. INTRODUCTION

A multiprocessor system is an interconnection of two or more CPUs sharing common memory

and input-output equipments. The excellent benefit derived from a multiprocessor organization is an improved system performance. It is achieved through partitioning an overall function into a number of tasks that each processor can handle indivisually. And, multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system. For example, if a fault cause one processor to fail, a second processor can be assigned to perform the functions of the disabled processor. [1]

The simplest interconnection system for multiprocessor system is a common communication path connecting all of the functional units. This common path is often called a time-shared or common bus. The common bus multiprocessor system consists of a number of processors connected through a common path to a memory unit. Only one processor can communicatie with the memory at any given time. This organization is the least complex and the easiest to reconfigure. [2]

However, system expansion, by adding more processors or memory, increases the bus contention, which degrades system throughput and increases arbitration logic. The total overall transfer rate within the system is limited by the bandwidth and speed of this single path. For this reason, private cache memories and high performance bus are used to improve the performance of multiprocessor system. [3,4]

If multiple caches are allowed to have copies simultaneously of a given memory location, a mechanism must exist to ensure that all copies remain consistent when the contents of that memory location are modified. [5,6]

In this paper, we suggest a Futurebus$^+$ based multiprocessor system with MESI cache coherence protocol, examine that multiprocessor system, and evaluate their relative perfomance of four bus transaction types on the basis of a simulation model. Before a simulation, a steady-state probability of each state is obtained with a Markov chain. [7,8] This steady-state probability of each state is used as an input value for a correct simulation. Time and average utilizations are obtained with a set of parameters change.

## II. FUTUREBUS$^+$ AND MESI CACHE COHERENCE PROTOCOL

### 2.1 Futurebus$^+$

Futurebus$^+$ is a revised and substanially extended version of the original IEEE 896.1 1987 Futurebus standard, where the basic protocols and facilities were developed. From 1983 to 1987 the IEEE Futurebus Working Group, provided a forum for leading experts to develop innovative technologies and protocols for a scalable performance multiprocessor system bus. The most recent Futurebus$^+$ efforts, from 1988 onwards, represented a commercial consolidation of all the basic Futurebus philosophies into a realizable and practical standard as an industry consensus developed between the major organizations who became interested in developing products based on Futurebus$^+$. [9,10]

Futurebus$^+$ represents a major paradigm shift for the computer industry. It is the first comprehensive bus arhitecture designed a priori to be an OPEN shandard(An interface specification for which there are no restrictions for who may use it), and which was explicitly designed to support multiple generations of computer technology.

Futurebus$^+$ derives its name from its lack of built in obsolescence parameters, and its upwardly compatible architecture and protocol extensions. Futurebus$^+$ represents a significant departure from the philosophy of other standard or proprietary buses. The most important objectives of the Futurebus$^+$ project were [9]

1) to create a bus standard that would provide a significant step forward in the facilities and performance available to the designers of future multiprocessor systems, and,

2) to provide a stable platform upon which manufacturers can base several generations of computer systems.

Table 2.2 Major Features of Five Buses.

| Features | NU BUS | VME BUS | MULTIBUS II | HiPi BUS | Futurebus+ |
|---|---|---|---|---|---|
| Addressing Capability (bits) | 32 | 16, 24, 32 | 32 | 32 | 32, 64 |
| Data Path Capability (bits) | 8, 16, 32 | 8, 16, 32 | 8, 16, 24, 32 | 64 | 32, 64, 128, 256 |
| Data Path Multiplexed | Yes | No | Yes | No | Yes |
| Bus Protocol | Sync. 10 MHz | Async. | Sync. 10 MHz | Sync. 12.5 MHz | Async. technology independent |
| Sequence Access Support | 2, 4, 8, 16 | Block transfer not to cross 256 byte boundary | Block transfer | Block transfer | 1, 2, 4, 8, 16, 32, 64, words or ED* controlled |
| Byte Ordering | Little Endian | Big Endian | Little Endian | Little Endian | No constraints |
| Data Transfer Rate | 37.5,block trans. | 57 MB/s | 40 MB/s | 100 MB/s | 3.2 GB/s |
| Standard | IEEE 1196 | IEEE 1104 | IEEE 1296 | | IEEE 896 |
| Message Passing | No | No | Yes | No | Yes |
| Cache Support | Some | No | Some | Some | Yes |
| Live Insertion | No | No | No | No | Yes |
| Bus Drivers | TTL | TTL | TTL | TTL | Any Technology |

### 2.2 MESI Cache Coherence Protocol

One specific application which the Futurebus+ is extraordinarily well suited to, is the shared memory multiprocessor system. The Futurebus+ protocols include all the transaction types necessary to drive the states of various cache modules to conform with almost any arbitrary cache coherence protocol. The Futurebus+ Working Group discovered, early in 1986, that all the existing cache coherence protocols are really variations on the same theme, but with an arbitrary nomenclature for the cache states. The result was the development of a unified cache model called the MOSEI Futurebus+, uses a slight restricition of this model (MESI : Modified, Exclusive, Shared, and Invalid), in which tradeoffs were made to increase system performance and the allow operation of the protocols across bridges.[9]

The coherence of data in multiple caches is based on the cache coherence attributes invalid, shared unmodified, exclusive unmodified, and exclusive modified. Only one of these cache attributes may be valid for a cache line in a particular module at a given time. These four attributes comprise the following cache line states.

The invalid attribute is true for a cache line if there is not an up to data copy in the module's cache. All cache lines in a module's cache are marked invalid when a system reset is received.

The shared unmodified attribute is true for a cache line if there is an up to date copy of the line in the module's cache and the module is to assume that a copy also exists in another module's cache. Modules are allowed to invalidate a shared unmodified copy of a cache line at any time.

The exclusive unmodified attribute is true for a cache line if there is an up to data copy of the line in the module's cache and the module has been guranteed that no other copies of the line are valid in any other cache in the system.

The exclusive modified attribute is true for a cache line if there is an up to date copy of the line in the module's cache and the module has the only valid copy in the system. The module has the responsibility to eventually update memory.

MESI cache coherence protocol guarantees that only one processor on the bus has a modified copy of any given cache line at the same time.

The IEEE 896.1 cached transactions are Read Invalid, Write Invalid, Read Shared, Copyback, Read Modified, Invalidate, Shared Response, and Modified Response.

When a processor issues a read, if the location requested is Shared Unmodified, Exclusive Unmodified, or Exclusive Modified, the operation is said to be a read hit. In the case of a read hit, no bus transaction is necessary and the processor is allowed immediate access to the data. If the location is invalid, the operation is said to be a read miss.

When a processor issues a write, if the location request is Exclusive Unmodified Exclusive Modified, the operation is said to be a write hit, and no bus transaction is needed. If the lcoation is not tagged exclusive, the operation is said to be a wirte miss, even if the location is valid.

## Ⅲ. MODELING AND SIMULATION

To show the performace of the multiprocessor system using the MESI protocol, we used SLAM II.[11] SLAM II is an advanced FORTRAN based simulation language which allows models to be built using its graphical models that are easily translated into input statements for direct computer processing. The simulation model was verified through tracing optional sequences with MONTR, TRACE control statement. For simulation results, a set of paramenters was successively changed under UNIX system V.

### 3.1 System Configurations for Simulation

Figure 3.1 shows the shared memory architecture considered in the simulation. The system consists of N statistically identical processors. Each processor is associatied with a private cache memory and is connected to a Futurebus+. The main memory is divided into M interleaved modules to increase the bandwidth.

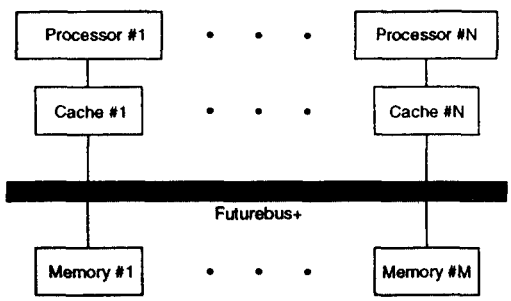



Fig 3.1 A Futurebus+ based Multiprocessor System.

### 3.2 A Markov State Diagram for the MESI Protocol

A cache block(line) in the MESI protocol can be one of the five states : NOTIN(not in cache), EXCLUSIVE UNMODIFIED(in cache, clean, potentially shared), EXCLUSIVE MODIFIED(in cache, modified, only copy), SHARED UNMODIFIED(in cache, clean, multiple copies), and INVALID(not valid). This protocol assumes that missed blocks always come from other caches if one or more copies are cached.

The state transition diagram of a block is shown in Figure 3.2. In order to compute the Markov state diagram, we can get the steady-state probability of each state :

$$P_N = m/(w+r+m)$$
$$P_{EU} = (r(1-s)(P_N+P_I))/(m+(N-1)s(r+w)+w)$$
$$P_{SU} = ((N-1)sr(P_{EU}+P_{EM})+rs(P_N+P_I))/(m+w+(N-1)sw)$$
$$P_{EM} = w/(m+(N-1)s(r+w)+w)$$
$$P_I = ((N-1)sw(1-P_N))/(r+w+m+(N-1)sw)$$
$$P_N+P_{EU}+P_{SU}+P_{EM}+P_I = 1$$

The simulation parameters and their ranges are summarized in Table 3.1

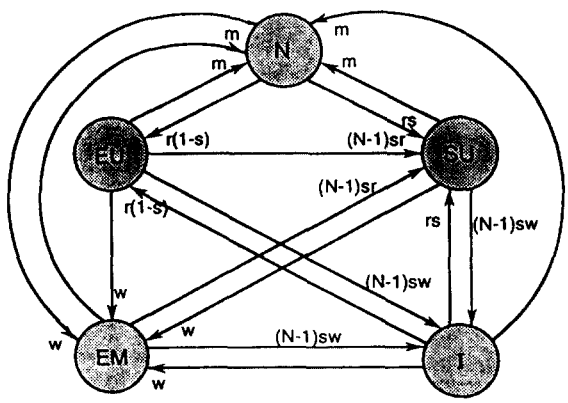



Fig 3.2 A Markov State Diagram for the MESI protocol.

Table 3.1 Summary of parameters and their ranges

| Parameter | Range |
|---|---|
| s | 2.5 % |
| r | 70 - 90 % |
| h | 90 - 99 % |
| Memory Access Time | 320 - 1600 ns |
| Block Size | 64 bytes |
| Number of Processors | 4 - 40 |
| Number of Memory Modules | 2 - 8 |
| Bus Bandwidth | 32 - 256 bit |
| Internal Operation | 50 - 90 % |
| Processor Clock | 20 Mhz |

### 3.3 Assumptions

A few number of assumptions are needed for our simulation.

1) In simulation, the operation of processor module depends on the state of cache block. In order to adopt this operation to the simulation, the steady-state probability of each state is used as an input value of the simulation for the correct results of the simulation.

2) The default probability of the internal operation of processors is fixed as 0.7.

3) The processor, a central arbiter, and the main meory are assumed to have fixed service times 50ns, 45ns, and 320ns, respectively.

4) The default number of processors is 15.

5) The default hit ratio is 0.99.

6) The default read ratio is 0.7

7) The default number of memory modules is 16.

8) The ratio of shared blocks, which the probability of a memory request to a shared block, is 0.0255.

9) The default bandwidth of the Futurebus+ is 64bit.

10) Simulation time is 1,000,000ns.

### 3.4 SLAM II network modeling

A connected transaction is modeled such that a processor occupies the bus during the entire duration of a memory access. A split transaction is modeled such that a processor releases the bus as soon as it has sent a request which contains the desired operation and memory address to the memory system.

A processor is modeled to conduct bus transactions during its bus tenure. Each bus transaction encompasses the following :

1) A connection phase : the processor selects and establishes a connection to a desired slave.

2) An optional data transfer phase : data are transferred between the processor and the connected slave.

3) A disconnection phase : the processor terminates the transaction and disconnects from the slave.

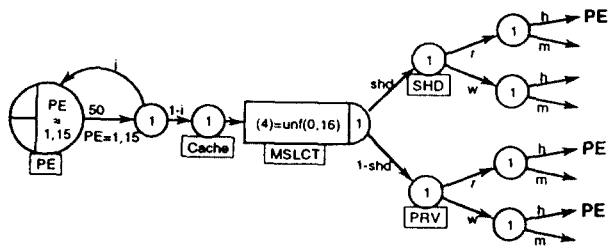
During the data transfer phase, the processor conducts data transfers uning either the compelled data transfer mode or the packet data transer mode.

#### 3.4.1 Processor modeling

SLAM II simulation language provides 'create' statement for entity creation. The 'create' statement is inappropriate in modeling the closed network, however. Instead, we used a 'queue' statement for modeling the closed network. It should be noted that entities in SLAM II are neither created nor terminated within the graphical model. Therefore, we must insert entity statement to initiate the request of processors. The request of processors reaches the bus due to the fixed probability.

The reference stream of each processor is viewed as the merging of two reference streams-one being references to shared blocks and the other references to private blocks. Whenever a memory reference is called for, the processor generates a reference to a shared block with probabilty $s$ and a reference to a private block with probability $1-s$. In a similar fashion, the probability of the reference being a read is $\gamma$ and the probability of its being is a write is $1-\gamma$. The request is a hit with probability $h$ and a miss with probability $1-h$.

The processor modeling remains unchanged, albeit transaction types change. Figure 3.3 shows a graphical model for the processors.




MSLCT : Memory SeLeCT
SHD : SHareD
PRV : PRiVate
PE : Processing Element
i : internal operation

Fig 3.3 Graphical Model for Processors.

### 3.4.2 Arbiter modeling

In this paper, central arbiter is selected for our simulation because it is simple to model and be cause it is been for most profiles. Any number of the bus allocation schemes may be implemented. A simple, single prioity, first-come-first-served arbiter is modeled out of many possible schemes. Its arbitration time takes 45ns (from sending a request to receiving a grant).

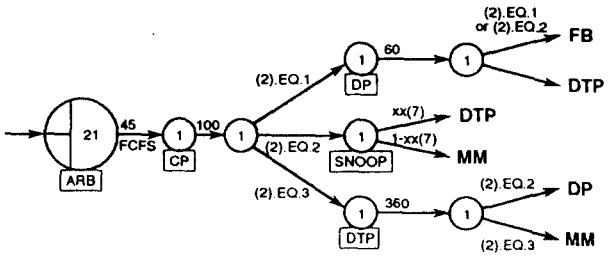
### 3.4.3 Bus modeling

Entities have two attributes, which determines the use of the bus. The requests wait for the bus in a central until they reach the bus. According to the value of the attributes, request of the winner processor branches to one of the four possible transactions. To solve the collision of the bus, we define the bus as a resource.

Figure 3.4 shows a graphical model for the Futurebus$^+$.

Each transaction take the same amount of time in the connection phase and in the disconnection phase. Timing values of the connection phase and the disconnection phase are shown at Table 3.2.

However, it must be noted that data transfer time in each varies with data transfer methods.




DTP : Data Transfer Phase
MM : Memory Module
DP : Disconnection Phase
CP : Connection Phase
FB : Free Bus
ARB : ARBiter
FCFS : First Come First Served

Fig 3.4 Graphical Model for Futurebus$^+$.

Table 3.2 Timing specifications in connection and disconnection phases

| Connection phase | Disconnection phase |
|---|---|
| Maser(17) | Master(8) |
| Futurebus+(5) | Futurebus+(5) |
| Slave(47) | Slave(16) |
| Futurebus+(5) | Futurebus+(5) |
| Master(26) | Master(26) |
| Total = 100 ns | Total = 60 ns |

How the transfer time varies is detailed in the next two sections.

#### 3.4.3.1 Transactions using compelled mode

Data transfer time for write or read operation takes about 45ns. Therfore, we calculate the sustained transfer rate in the compelled mode.

For a single word transfer, 20MB is transferred per 1 sec for a 32-bit bus, 40MB for a 64-bit bus, 80MB for a 128-bit bus, and 160MB for a 256-bit bus. Meanwhile, for a single cache line(64 bytes) transfer, total data transfer time is calculated as follows :

For 32-bit bus : $100+16*45+60=880$ nsec/64 bytes,
for 64-bit bus : $100+8*45+60=520$ nsec/64 bytes,
for 128-bit bus : $100+4*45+60=340$ nsec/64 bytes,
for 256-bit bus : $100+2*45+60=250$ nsec/64 bytes.

#### 3.4.3.2 Transactions using packet mode

Futurebus+ provides two kinds of packet mode. One is 75MHz(13.33 ns/transfer), and the other is 100MHz(10 ns/transfer). In this paper, only the latter was considered. As in the case of the compelled mode, we calculate the sustained transfer rates with the packet mode. In a packet mode, one start bit, one stop bit, and one empty cycle are needed in addition to several data transfers. For a single cache line transfer, toal data transfer time is calculated as follows :

For 32-bit bus : $100+19*10+10=350$ ns/64 bytes,
for 64-bit bus : $100+11*10+60=270$ ns/64 bytes,
for 128-bit bus : $100+7*10+60=230$ ns/64 bytes,
for 256-bit bus : $100+5*10+60=210$ ns/64 bytes.

#### 3.4.4 Transaction modeling

Connected transactions using compelled mode, connected transactions using packet mode, split transactions using compelled mode, and split transactions using packet mode are modeled according to following Futurebus+ transactions.

Four transactions have several phases :

1) ReadModified transactions and ReadShared transactions have arbitration phase, connection phase, data transfer phase, and disconnection phase.

2) CopyBack transactions have arbitration phase, connection phase, data transfer phase, and memory access phase prior to the same phases of ReadModified transactions being performed.

3) Invalidate transactions have arbitration phase, connection phase, and disconnection phase.

Connected transaction is used to conduct a module's request and reponse in a single bus transaction. In a connected transaction, a processor occupies the bus during the entire duration of a memory access.

Split transaction is used to split a modul's request and reponse phases into seperate bus tenures and transactions. The module serviving the request responds by becoming a master, addressing the requester, and transmitting a responses. In the split transaction, the bus and memory modules are seperate severs. A processor release the bus as soon as it has sent a request that contains the desired operation and memory address to the memory system. The released bus can then be

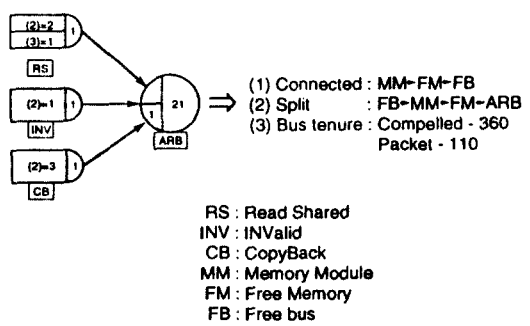



Fig 3.5 Graphical Model for Each Transaction.

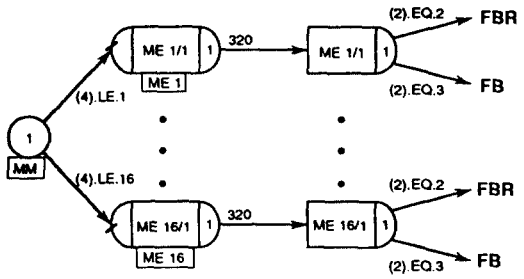
used for other purposes.

Figure 3.5 shows a graphical model for each transaction.

### 3.4.5 Memory Modeling

Memory is defined as a resource, and consists of sixteen memory modules. Since memory is defined as a resource, requests wait for the memory in each AWAIT node. Sixteen memory modules are selected randomly with UNIFORM DISTRIBUTION.

Figure 3.6 shows a graphical model for memory.




FBR : Free Bus for Read shared

Fig 3.6 Graphical Model for Memory.

### 3.4.6 System Modeling

System is described with SLAM II in a closed network. System consists of several models mentioned above. Figure .7 shows a block diagram for system.

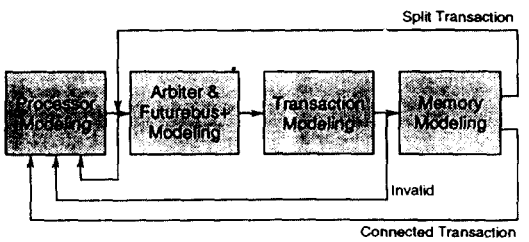



Fig 3.7 Block Diagram for System.

## Ⅳ. RESULT ANALYSIS

Timings and average utilizations are used to evaluate the performance of a Futurebus+ based multiprocessor system using MESI protocol. The experiment was repeated for 1,000,000ns for each of the four transaction types-connected transaction using compelled mode, connected transaction using packet mode, split transaction using compelled mode, and split transaction using packet mode-in order to compare their performances and to obtain a general tendency with simulation parameters change.

### 4.1 Performance Versus Number of processors

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction processing types were calculated with the number of processors changed from 4 to 40 with an interval of 4. The results are given in Table 4.1.

From Table 4.1, we can conclude that connected transactions using the packet mode performed best. Because the bus tenure takes more times than that of memory, connected transaction showed better performance than split trasaction.

When the number of processors increased, the bus utilization got saturated rapidly, thus causing

| # of P | CCP.R | | | | SCP.R | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 4 | 106 | 92.8 | 25 | 0.9 | 114 | 89.3 | 30 | 1 |
| 8 | 267 | 91.7 | 45 | 1.3 | 335 | 87.0 | 54 | 1.1 |
| 12 | 481 | 90.3 | 61 | 1.8 | 585 | 85.0 | 71 | 2 |
| 16 | 743 | 88.8 | 74 | 2 | 1102 | 79.8 | 87 | 2 |
| 20 | 1388 | 83.8 | 87 | 2.4 | 1837 | 74.0 | 95 | 2 |
| 24 | 2176 | 79.5 | 93 | 2.6 | 3010 | 65.8 | 99 | 2.1 |
| 28 | 3496 | 71.9 | 98 | 2.8 | 4304 | 59.0 | 100 | 2 |
| 32 | 4873 | 65.8 | 100 | 2.9 | 5752 | 52.6 | 100 | 2 |
| 36 | 5799 | 63.7 | 100 | 2.7 | 7093 | 48.4 | 100 | 2 |
| 40 | 7200 | 59.7 | 100 | 2.8 | 8611 | 44.2 | 100 | 2 |
| # of P | CPP.R | | | | SPP.R | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 4 | 67 | 94.2 | 20 | 0.9 | 55 | 92.2 | 21 | 1 |
| 8 | 153 | 93.8 | 37 | 1.3 | 139 | 91.6 | 39 | 1.3 |
| 12 | 260 | 93.1 | 53 | 2 | 226 | 91.0 | 52 | 2 |
| 16 | 392 | 92.3 | 65 | 2.1 | 361 | 89.5 | 66 | 2.1 |
| 20 | 744 | 89.4 | 78 | 2.7 | 533 | 87.8 | 77 | 2.8 |
| 24 | 1106 | 86.9 | 87 | 2.8 | 825 | 84.7 | 87 | 3 |
| 28 | 1756 | 82.5 | 93 | 3.1 | 1206 | 81.2 | 94 | 3 |
| 32 | 2482 | 78.6 | 98 | 3.3 | 1798 | 76.3 | 98 | 3.2 |
| 36 | 3193 | 75.6 | 99 | 3.2 | 2580 | 70.6 | 99 | 3 |
| 40 | 4772 | 68.0 | 100 | 3.2 | 3463 | 65.1 | 100 | 3.1 |

the request to wait longer before being granted bus service. Processor utilization decreased slowly from the point when sixteen or more processors were installed.

### 4.2 Performance Versus Hit ratio

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction processing types were calculated with the hit ratio changed from 0.9 to 0.99 with an interval of 0.01. The results are given in Table 4.2.

From Table 4.2, we can conclude that connected transaction using packet mode performed best. Because the bus tenure took more times than memory access, connected transaction showed better performance than split transaction.

In most cases, performance of the multiprocessor system is known to be influenced very much with hit ratio change. However, since Futurebus+ fixed a cache line size as 64 byte, performance was not as good as usual from 0.90 to 0.98. Therefore, for a good performance, hit ratio should maintained about 0.99 or above in a Futurebus+ based multiprocessor system.

Table 4.2 Performances Versus Hit ratio

| Hit ratio | CCHR | | | | SCHR | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.90 | 6850 | 13.3 | 100 | 3.1 | 5567 | 10.2 | 100 | 2.5 |
| 0.91 | 6791 | 14.3 | 100 | 3 | 5535 | 11.6 | 100 | 2.6 |
| 0.92 | 6405 | 18.0 | 100 | 3.1 | 5400 | 12.8 | 100 | 2.6 |
| 0.93 | 6381 | 20.4 | 100 | 3.1 | 5284 | 15.4 | 100 | 2.5 |
| 0.94 | 6289 | 22.3 | 100 | 3 | 5029 | 18.1 | 100 | 2.6 |
| 0.95 | 5729 | 25.9 | 100 | 3.1 | 5014 | 18.6 | 100 | 2.4 |
| 0.96 | 5124 | 33.2 | 100 | 2.9 | 4557 | 25.2 | 100 | 2.4 |
| 0.97 | 4043 | 42.3 | 100 | 3.1 | 3870 | 35.6 | 100 | 2.4 |
| 0.98 | 2488 | 64.1 | 97 | 3.1 | 2632 | 53.1 | 100 | 2.4 |
| 0.99 | 784 | 87.9 | 73 | 2.1 | 999 | 80.5 | 84 | 2.1 |
| Hit ratio | CPHR | | | | SPHR | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.90 | 5517 | 16.1 | 100 | 3.7 | 3467 | 15.5 | 100 | 3.9 |
| 0.91 | 5389 | 17.5 | 100 | 3.6 | 3408 | 16.8 | 100 | 4 |
| 0.92 | 5149 | 20.9 | 100 | 3.8 | 3241 | 20.2 | 100 | 3.8 |
| 0.93 | 4981 | 23.1 | 100 | 3.7 | 3117 | 22.9 | 100 | 3.8 |
| 0.94 | 4791 | 25.9 | 100 | 3.7 | 3058 | 24.9 | 100 | 3.9 |
| 0.95 | 4270 | 32.9 | 100 | 3.7 | 2718 | 31.7 | 100 | 3.8 |
| 0.96 | 3661 | 41.2 | 100 | 3.6 | 2393 | 39.3 | 100 | 3.7 |
| 0.97 | 2880 | 52.2 | 100 | 3.7 | 1872 | 50.5 | 100 | 3.8 |
| 0.98 | 1429 | 74.0 | 95 | 3.3 | 1047 | 68.9 | 94 | 3.6 |
| 0.99 | 461 | 91.2 | 65 | 2.3 | 329 | 89.8 | 63 | 2.3 |

### 4.3 Performance Versus Memory Access Time

We changed the momory access time from 1 cycle(320 ns) to 5 cycles(1600 ns) with an interval of 1 cycle. Arbitration waiting time, processor utilization, bus utilization, and memory utilization are given in Table 4.3. As memory access times are increased, the processor utilization decreased rapidly because of memory access delay increase.

From Table 4.3, we can conclude that the difference between the connected transactions and split transactions is mostly influenced with change of the memory access time. We also can see that split transaction using packet mode performed best because split transaction released the bus during memory access.

We arrange the order of each transaction type on the basis of its processor utilization : split transaction using packet mode, split transaction using compelled mode, connected transaction using packet mode, and connected transaction using compelled mode, in the same order.

Table 4.3 Performances Versus Memory Access Time

| MAT(ns) | CCAR | | | | SCAR | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 320 | 729 | 88.6 | 72 | 2 | 959 | 81.2 | 83 | 2 |
| 640 | 2117 | 77.3 | 91 | 3.6 | 1311 | 76.6 | 87 | 3.8 |
| 960 | 4757 | 62.6 | 98 | 4.3 | 1924 | 69.7 | 91 | 5 |
| 1280 | 7682 | 52.8 | 100 | 4.8 | 2427 | 65.6 | 92 | 6 |
| 1600 | 11310 | 43.2 | 100 | 5 | 3200 | 59.6 | 94 | 7 |
| MAT(ns) | CPAR | | | | SPAR | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 320 | 388 | 92.2 | 62 | 2.1 | 317 | 90.1 | 62 | 2.2 |
| 640 | 1225 | 84.3 | 84 | 3.6 | 518 | 86.1 | 69 | 4.1 |
| 960 | 3585 | 68.5 | 96 | 4.7 | 891 | 80.0 | 77 | 5.6 |
| 1280 | 6914 | 54.1 | 100 | 5.3 | 1483 | 72.9 | 81 | 6.9 |
| 1600 | 9878 | 46.0 | 100 | 5.3 | 2105 | 66.0 | 87 | 8.3 |

### 4.4 Performance Versus Bus Bandwidth

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction processing types can be obtained with the bus bandwidth changed from 32-bit to 256-bit with an interval of 32-bit. The results are given in Table 4.4.

From Table 4.4, performance improvement of split transactions is larger than that of connected transactions because split transaction is sensitive to the data transfer time. And, performance improvement of the packet mode is larger than that of compelled mode.

We can arrange the order of each transaction type : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

Table 4.4 Performances Versus Bus Bandwidth

| BW(bit) | CCB.R | | | | SCB.R | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 32 | 1500 | 81.9 | 83 | 2 | 482 | 91.1 | 66 | 2 |
| 64 | 696 | 88.9 | 72 | 2 | 387 | 92.2 | 62 | 2 |
| 128 | 470 | 91.3 | 65 | 2 | 347 | 92.7 | 60 | 2 |
| 256 | 373 | 92.4 | 61 | 2 | 345 | 92.7 | 60 | 2 |
| BW(bit) | CPB.R | | | | SPB.R | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 32 | 2830 | 64.5 | 97 | 2 | 472 | 87.6 | 70 | 2 |
| 64 | 887 | 82.2 | 83 | 2 | 317 | 90.0 | 63 | 2 |
| 128 | 453 | 87.8 | 70 | 2 | 261 | 91.0 | 59 | 2 |
| 256 | 290 | 90.5 | 61 | 2 | 234 | 91.5 | 57 | 2 |

### 4.5 Performance Versus Internal Operation

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction processing types can be obtained with the internal operation changed from 0.5 to 0.9 with an interval of 0.1. The results are given in Table 4.5.

As the ratio of internal operation increased, performance is improved because of total time delay decrease.

We can arrange the order of each transaction type : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

Table 4.5 Performances Versus Internal Operation

| LO | CCLR | | | | SCLR | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.5 | 2051 | 68.6 | 97 | 2.7 | 2262 | 58.0 | 99 | 2.3 |
| 0.6 | 1359 | 79.0 | 89 | 2.5 | 1771 | 66.9 | 95 | 2.4 |
| 0.7 | 632 | 89.4 | 72 | 1.9 | 822 | 83.2 | 82 | 2 |
| 0.8 | 386 | 93.8 | 54 | 1.6 | 454 | 90.5 | 63 | 1.6 |
| 0.9 | 150 | 97.9 | 25 | 0.9 | 166 | 96.5 | 32 | 0.9 |
| LO | CPLR | | | | SPLR | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.5 | 1233 | 76.7 | 92 | 3.1 | 901 | 74.0 | 91 | 3.1 |
| 0.6 | 616 | 87.5 | 78 | 2.6 | 489 | 84.2 | 79 | 2.7 |
| 0.7 | 388 | 92.0 | 64 | 2.1 | 328 | 89.7 | 64 | 2.2 |
| 0.8 | 207 | 95.7 | 44 | 1.4 | 173 | 94.2 | 45 | 1.8 |
| 0.9 | 97 | 98.2 | 22 | 0.9 | 80 | 97.6 | 23 | 0.9 |

### 4.6 Performance Versus Read Ratio

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction processing types can be obtained with the read ratio changed from 0.7 to 0.9 with an interval of 0.05. The results are given in Table 4.6.

As the ratio of the read operation increases, copyback of the content in cache memory is decreased when the replacement is occured. Therefore, performance is improved.

Table 4.6 Performances Versus Read Ratio

| R.R | CCR.R | | | | SCR.R | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.70 | 729 | 88.6 | 72 | 2 | 959 | 81.2 | 83 | 2 |
| 0.75 | 626 | 89.8 | 69 | 2 | 811 | 83.2 | 81 | 2 |
| 0.80 | 569 | 90.7 | 66 | 2 | 766 | 83.8 | 79 | 2 |
| 0.85 | 475 | 91.9 | 63 | 2 | 648 | 85.6 | 76 | 2 |
| 0.90 | 386 | 93.2 | 57 | 2 | 556 | 86.8 | 73 | 2 |
| R.R | CPR.R | | | | SPR.R | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 0.70 | 392 | 92.2 | 62 | 2 | 321 | 90.0 | 63 | 2 |
| 0.75 | 357 | 92.7 | 60 | 2 | 285 | 90.6 | 61 | 2 |
| 0.80 | 333 | 93.2 | 58 | 2 | 260 | 91.6 | 58 | 2 |
| 0.85 | 301 | 93.5 | 56 | 2 | 222 | 91.7 | 56 | 2 |
| 0.90 | 256 | 94.5 | 52 | 2 | 181 | 92.6 | 51 | 2 |

### 4.7 Performance Versus Number of Memory Modules

The average arbitration waiting time, processor utilization, bus utilization, and memory utilization in the four transaction types can be obtained with the number of memory modules changed from 2 to 8 with an interval of 2. The results are given in Table 4.7.

In view of total memory access time, two memory modules take four cycles, four memory modules two cycles, six memory modules two cycles, and eight memory modules only one cycle. Therefore, performance is improved according to increase of the number of memory modules except for the case of four memory modules and six memory modules.

We arrange the order of each transaction type on the basis of its system power : split transaction using packet mode, split transaction using compelled mode, connected transaction using packet mode, and connected transaction using compelled mode, in that order.

Table 4.7 Performance Versus Number of Memory Modules

| # of MM | CCMR | | | | SCMR | | | |
|---|---|---|---|---|---|---|---|---|
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 2 | 11489 | 31.5 | 100 | 39 | 9833 | 34.6 | 100 | 41 |
| 4 | 5621 | 45.0 | 100 | 15.5 | 4548 | 49.7 | 100 | 18 |
| 6 | 5325 | 47.8 | 100 | 10.7 | 3649 | 57.7 | 100 | 11.8 |
| 8 | 1727 | 73.0 | 95 | 5.4 | 1049 | 80.2 | 88 | 6.1 |
| # of MM | CPMR | | | | SPMR | | | |
| | awt(ns) | pu(%) | bu(%) | mu(%) | awt(ns) | pu(%) | bu(%) | mu(%) |
| 2 | 2648 | 63.4 | 92 | 46.5 | 1756 | 69.9 | 82 | 51.5 |
| 4 | 1409 | 75.1 | 88 | 14.8 | 602 | 84.5 | 71 | 16 |
| 6 | 1245 | 77.3 | 88 | 9.5 | 556 | 85.4 | 70 | 10.5 |
| 8 | 976 | 80.6 | 85 | 4 | 339 | 89.7 | 63 | 4.3 |

## V. SUMMARY AND CONCLUSIONS

In the shared bus multiprocessor system, bottlenecks of the bus and memory access delays are known to be the two major problems which negatively affect system performance. The former problem is usually solved by employing the fast bus which is capable of transferring a large amount of data at a time. The latter latter problem is solved with the use of private caches attached to each processor.

In order to solve the problems mentioned above, we selected the Futurebus+ based multiprocessor system with MESI cache coherence protocol. Then, we used simulation technique with SLAM II to evaluate the performance of that multiprocessor system. Before simulations, the steady-state probability of each state was calculated using Markov chain. This probability was used as an input paramenter for correct simulation.

Conclusions based on the simulation results are :

1) In case where the number of processors varies, we can arrange an order of four transaction types according to its system power. The order of the four types is connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

2) In case of the change in hit ratio, we can arrange the order of each transaction type on the basis of its system power : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

3) In case where read ratio varies, we can arrange the order of transaction type on the basis of its system power : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

4) In case where internal operation varies, we can arrange the order of each transaction type on the basis of its system power : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

5) In case where memory access time changes, we can arrange the order of each transaction type on the basis of its system power : split transaction using packet mode, split transaction using compelled mode, connected transaction using packet mode, and connected transaction using compelled mode, in that order.

6) In case where number of memory modules changes, we can arrange the order of each transaction type on the basis of its system power : split transaction using packet mode, split transaction using compelled mode, connected transaction using packet mode, and connected transaction using compelled mode, in that order.

7) In case where bus bandwidth changes, we can arrange the order of each transaction type on the basis of its system power : connected transaction using packet mode, split transaction using packet mode, connected transaction using compelled mode, and split transaction using compelled mode, in that order.

For all of the cases, the transaction using the packet mode performed better than that using compelled mode because of its fast data transfer time. Connected transaction showed better performance than split transaction except when memory access time changed and when the number of memory module was changed. When the memory

access time or the number of memory modules changed, split transaction showed better results than connected transactions because split transaction released the bus during memory access.

Finally, Futurebus+ is well suited to implement the shared bus multiprocessor system in view of the cache coherence protocol and connected transaction using packet mode performed best among four transaction types.

## REFERENCE


1. D. D. Gajski and J. K. Pier, "Essential Issues in Multiprocessor System," IEEE Transactions on Computers, Vol. 18 June 1985, pp. 9-27.
2. K. Hwang and F. A. Briggs, Computer Architecture and parallel Processing, McGraw-Hill, 1985.
3. J. H. Patel, "Analysis of Multiprocessors with Private Cache Memories," IEEE Transactions on Computers, Vol. C-31, April 1982, pp. 296-304.
4. M. K. Vernon, E. D. Lazowska, "An Accurate and Efficient Performane Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols," The 15th Annual International Symposium on Computer Architecture, May 1988, pp. 308-315.
5. L. M. Censier and Paul Feautrier, "A Bew Solution to Coherence Problems in Multicache Systems," IEEE Transactions on Computers, Vol. C-27, December 1978, pp. 1112-1118.
6. S. V. Adve, V. S. Adve, M. D. Hill, and M. K. Vernon, "Comparison of Hardware and Software Cache Coherence Schemes," The 18th Annual International Symposium on Computer Architecture, May 1991, pp. 298-308.
7. I. H. Onyuksel and K. B. Irani, "Markovian Queueing Models for Performance Analysis of a Single-Bus Multiprocessor System," IEEE Transactions on Computers, Vol. 39, July 1990, pp. 975-980.
8. 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, August 1989, pp. 1143-1153.
9. IEEE Standard 896.1/896.2/896.3/896.4-1991, IEEE Standard for Futurebus+
10. Peter Robinson, "IEEE Futurebus Cache Coherence Protocol as a Logic Program," Micorprocessors and Microsystems, Vol. 14, October 1990, pp. 491-496.
11. A. Pritsker, Introduction to Simulation and SLAM II, Systems Publishing Corporation, 1986.
12. IEEE Std 1194.1-1991, IEEE Standardfor Electrical Characteristics of Backplane Transciver Logic(BTL) Interface Circuits.(ANSI)

**高 錫 范**(Seok Bum Ko) 정회원
1969년 5월 16일생
1991년 2월 : 전북대학교 컴퓨터공학과 졸업(공학사)
1993년 2월 : 전북대학교 컴퓨터공학과 졸업(공학석사)
1993년 2월 ~ 현재 : 한국통신 진임 연구원

**朴 聖 宇**(Sung Woo Park) 정회원
1962년 9월 13일생
1981년 3월 ~ 1985년 2월 : 연세대학교 전자공학과 졸업(공학사)
1985년 1월 ~ 1986년 3월 : 한국 데이타 통신(주) 연구원
1986년 6월 ~ 1989년 8월 : Texas A&M Univ. Dept. of Electrical Engineering(MS)
1989년 9월 ~ 1991년 12월 : Univ. of Califonia, Irvine Dept. of Electrical and Computer Engineering(Ph.D)
1992년 3월 ~ 현재 : 한남대학교 정보통신공학과 조교수
※주관심분야 : Computer Networks

**姜 實 坤**(In Kon Kang) 정회원
1966년 4월 22일생
현재 : 전북대학교 컴퓨터공학과 박사과정
제18권 제1호 참조

**金 永 川**(Young Chon Kim) 정회원
1956년 12월 10일생
현재 : 전북대학교 컴퓨터공학과 부교수
제18권 제1호 참조