


## The Effects of Cache Memory on the System Bus Traffic

Yong Hoon Cho\*, Jung Sun Kim\*\* Regular Members

### 캐쉬 메모리가 버스 트래픽에 끼치는 영향

  
正會員 趙龍勳\*, 金正善\*\*

#### ABSTRACT

It is common sense for at least one or more levels of cache memory to be used in these day's computer systems. In this paper, the impact of the internal cache memory organization on the performance of the computer is investigated by using a simulator program, which is written by authors and run on SUN SPARC workstation, with several real execution trace files. 280 cache organizations have been simulated using n-way set associative mapping and LRU(Least Recently Used) replacement algorithm with write allocation policy. As a result, 16-way set associative cache is the best configuration, and when we select 256KB cache memory and 64 byte line size, the bus traffic ratio was decreased compared to that of the noncache system so that a single bus could support almost 7 processors without any delay and degradation of hit ratio(hit ratio was 99.21%). The smaller the line size we choose, the little lower hit ratio we can get, but the more processors can be supported by a single bus(maximum 18 processors). Therefore, using a proper cache memory organization can make a single bus structure be able to support multiple processors without any performance degradation.

#### 要 約

근래 사용되고 있는 컴퓨터들은 그 내부에 적어도 한 두 레벨의 캐쉬 메모리를 사용하는 것이 통례이다. 본 논문에서는 중앙처리장치 내에 존재하는 캐쉬 메모리의 구성이 그 컴퓨터의 성능(특히 버스 트래픽)에 미치는 영향을, 실존하는 컴퓨터가 특정 프로그램을 실행할 때 실제 접근하는 메모리 주소 트레이스를 발췌하여 시뮬레이션함으로써 분석한다. n-way set associative mapping과 LRU(Least Recently Used) 교환 알고리즘을 사용하여 전체 280가지의 캐쉬 구조를 분석하여 본 결과, 16-way set associative mapping 구조가 가장 좋았으며, 256K 캐쉬에서 그 라인크기(line size)를 64 바이트로 하였을 경우 적중률 99.21%를 유지하면서도, 버스는 캐쉬메모리를 사용하지 않는 시스템의 13.62%만 사용하므로 동시에 7개 정도의 프로세서를 단일 버스구조로 지원할 수 있다는 것을 알았으며, 그 라인

\* 경북실업전문대학 전자계산과

\*\* 한국항공대학교

論文番號 : 95067-0216

接受日字 : 1995年 2月 16日

크기가 작아질 수록 적중률은 약간 저하되었지만 버스 트래픽은 더욱 떨어져 최대 18개 까지의 프로세서를 지원할 수 있었다. 따라서 단일 버스 구조의 컴퓨터에서도 캐쉬 메모리를 적절히 이용한다면 시간지연 없이 다중프로세서 시스템의 지원이 가능하다.

## 1. Introduction

It is widely recognized that main memory and network latencies rapidly increase because CPU cycle time decreases. The feature that perhaps most distinguishes very-high-performance computers from more modest ones is extremely high processor-memory bandwidth. One alternative for the increased performance without proportionately increasing processor-memory bandwidth is to introduce local memory, which is called cache memory, on the same chip with the CPU. Moreover, most CPU chips are now being designed for integration into a massively parallel super computer, or a parallel workstation, and therefore optimizing memory hierarchy utilization including cache and virtual memory systems is critical. For all of these reasons, optimizing the cache behavior has become a major issue these days. In this paper, the effect of the cache memory on the system performance, especially on the system bus traffic, will be investigated, so that technical recommendations on the cache sizes and cache line sizes for the single bus multiprocessor system can be given.

An important problem for designers of new computer systems is the problem of estimating the performance of a new system. In fact, the designer does not assess the performance of a single new design, but rather tries to assess the impact of specific features of a design such as cache size, memory size, and I/O bandwidth. A further complication is

that features such as cache size may be evaluated for a variety of sizes. Hence, the number of alternatives to examine can be fairly large. The classic way to measure the performance of a system is to simulate the behavior of that system when executing a typical workload. The workload used in the simulation is produced by tracing the execution of that workload on an existing real computer, and the execution trace subsequently drives a simulation of the design variation of interest.

To do this, we developed a cache simulator program which could be used for simulating various cache behaviors. The simulator uses the real execution traces supplied by Morgan Kaufmann Publishers, Inc., San Mateo, California USA. The cache should not be so large that it represents an expense out of proportion to the added performance, nor should it occupy an unreasonable fraction of the physical space within the processor. In other papers [3,4], therefore, they used relatively small cache sizes (up to about 32K). In this paper, however, we extend the cache size up to 1M according to the present-day's tendencies to increase the on-chip cache size, which may be feasible in the near future. In addition, we extend the number of ways in the set associative mapping memory so that the effects of extending the number of ways on the system performance are revealed. 16-way set associative mapping was the best technique.

In chapter 2, a few basic cache design elements that serve to classify and differentiate

cache architecture are explained. This paper also shows how the simulator works and how the performance of the cache memory can be optimized in chapter 3. In chapter 4, the conclusion derived from the simulation analysis are given.

## 2. Cache Memory General

The sequence of memory addresses generated by a program typically exhibits the properties of temporal and spatial locality[4]. Temporal locality, or locality in time, means that memory addresses recently referenced by a program are likely to be referenced again in the near future. Spatial locality means that the address referenced by a program in a short period of time are likely to span a relatively small portion of the entire address space. For example, some programs frequently operate large data structures in which the consecutive elements of the structure are located in sequential memory locations. Thus, the memory addresses generated by a program to access such structures are likely to be clustered spatially into a small range of the address space. Private data caches, which are small, fast memories physically located near a processor exploit these memory-referencing properties to reduce the average time required to access the larger main memory. By temporarily storing in the cache a copy of a value from the main memory that is being actively referenced by a program, caches amortize the time required to copy the memory location from the slower main memory into the faster cache over several references to the same and nearby memory locations.

In order to locate an item in such a cache, it is necessary to have some function which maps the main memory address into a cache

location. For uniformity of reference, both main memory and cache are divided into equal-sized units, called lines. The placement policy determines the mapping function from the main memory address to the cache location. The line replacement policy proposes to select the line to be displaced when a miss occurs. In this chapter, various issues in the design of a cache are discussed.

### 2.1 Placement Policies

A key element of cache design is the mapping function, which maps blocks of memory into cache locations. Three techniques can be used: direct, associative, and set-associative. We will examine these three alternatives with an example. Consider a cache that can hold 64 Kbytes. And also, let us assume data is transferred between main memory and the cache in lines of 4 bytes each. This means that the cache is organized as 16K lines of 4 bytes each. And also, let us consider a main memory that hold 16 Mbytes, with each byte directly addressable by a 24-bit address ( $2^{24} = 16\text{M}$ ). For mapping purposes, we can consider the main memory to consists of 256 blocks ( $2^{24/16} = 256$ ) of 64 Kbytes each, which is same size as the cache size. Each block consists of 16K lines as the 64-Kbyte cache.

Since there are fewer cache lines than main memory lines, an algorithms is needed for mapping main memory lines into cache lines. Further, a means is needed for determining which main memory line currently occupies a cache line.

The simplest technique, known as direct mapping, allows each line of main memory blocks to be allocated into only one possible cache line. As we mentioned above, direct mapping allows each same-numbered line of each block in main memory to be copied only

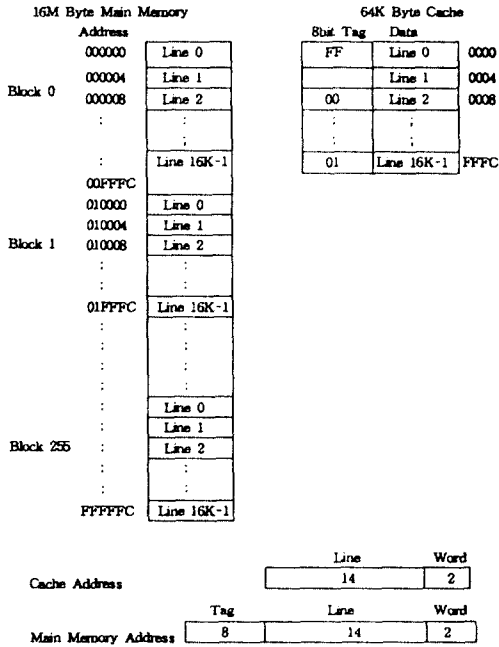


Figure 1. Direct Mapping

into the corresponding same-numbered line of the cache. This is illustrated in Figure 1. The mapping function is easily implemented using the 24-bit address. The least significant two bits serve to identify a unique byte within a line of main memory. The most significant 8 bits serve as the tag number, which is copied into the designated cache line along with the line contents. The remaining 14 bits specify one of the  $2^{14} = 16K$  lines. Thus, line 0 of each block of main memory can be copied only into cache line 0; line 1 of each block of main memory only into cache line 1; and so on, down to line 16K-1 which maps into cache line 16K-1. Note that no two or more same-numbered lines, each of which comes from those blocks of main memory, can map into a same cache line.

A read operation works as follows. The cache system is presented with a 24-bit address. The 14-bit line number is used as an

index of the cache to access a particular line. If the 8-bit tag number matches the tag number within that cache line, then the 2-bit word number is used to select one of the 4 bytes in that line. Otherwise, the 22-bit tag/line field is used to fetch a line from main memory. The actual address that is used for the fetch is the 22-bit field concatenated with two zero bits.

The direct mapping technique is simple, inexpensive to implement and has short cache access time. It's main disadvantage is that there is a fixed cache location for any given line of the main memory.

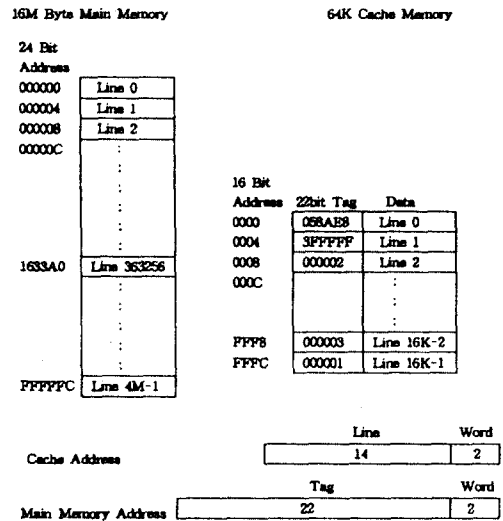


Figure 2. Associative Mapping

A technique that overcomes the disadvantage of the direct mapping approach is associative mapping, which permits a main memory line to be loaded into any line of the cache. In Figure 2, the main memory address consists of a 22-bit tag and a 2-bit byte number. A main memory line can be stored in any cache line, and its 22-bit tag is stored with it. To determine whether a block is in

the cache, logic is needed to simultaneously examine every line's tag for a match. Replacement algorithms, discussed in next section, are designed to maximize the hit ratio. The main disadvantage of the associative approach is the complex circuitry required to examine the tags of all cache lines in parallel.

Set-associative mapping is a compromise that captures the advantage of both the direct and associative approaches. Referring back to Figure 1, only one of the same-numbered lines of 256 main memory blocks could occupy the assigned cache line in the case of direct mapping organization. In the set-associative mapping, however, more than one line of those same-numbered main memory lines from each of 256 blocks can be copied into one cache line. As we see from Figure 3, two-way set-associative mapping makes 2 same-numbered main memory lines be located into one cache line. Thus, it decreases the number of lines in cache half way, and increases the number of tag bits by one. If we consider 4-way set associative mapping, it will decrease the number of cache lines by the factor of one fourth, and increase the number of tag bits by two.

The direct mapping and 2,4,8,16, and 32-way set associative mappings are used for simulation work in the next chapter. In general, direct mapping is simple and easy to implement. However, the performance is not better than that of set-associative mapping as we will see from our simulation results.

### 2.2 Replacement Policies

When the cache is full and a new line is brought into the cache, one of the existing lines must be replaced. For direct mapping, there is only one possible cache line for any

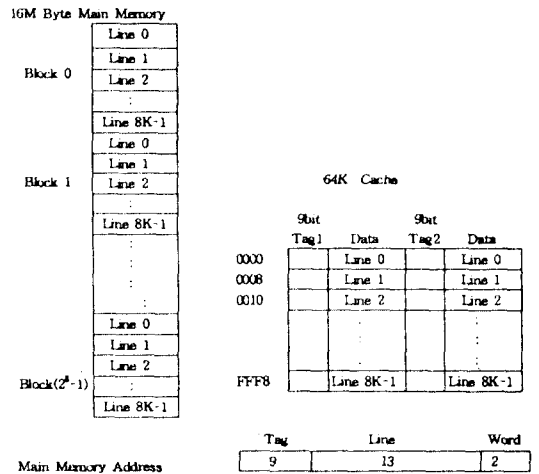


Figure 3. 2-way Set-Associative Mapping

particular main memory line, and no choice is necessary. For associative and set-associative mapping, a replacement algorithm is needed. A number of algorithms have been tried. Probably the most effective is LRU(Least-Recently Used). This policy replaces the block in the set which has been in the cache longest with no reference to it. Another possibility is FIFO(First-In First-Out) which replaces the block in the set which has been in the cache longest. FIFO is easily implemented as a round-robin of circular buffer technique. Still another possibility is LFU(Least-Frequently Used) which replaces that block in the set which has experienced the fewest references. Random algorithm replaces a line from among the candidate lines at random. In this paper, LRU algorithm is used for simulation work as we will see in the next chapter.

### 2.3 Write Policies

An important aspects of cache organization is concerned with memory write requests.

When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. However, if the operation is a write, the operation should be handled cautiously for memory consistency. There are three basic ways that the system can process : write-through, write-back, and write once.

The simplest procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the write-through method. This method has the advantage that main memory always contains the same data as the cache. This characteristic is important in systems with direct memory access transfer.

The second procedure is called the write-back method where only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory. Write-back method has more severe coherency problems than write-through, since even main memory does not always contain the current version of a particular memory location. However, the bus traffic is less than the one of the write-through method. This method is used for the simulation in this paper.

The third alternative is the write-once method which uses write-back for locations written repeatedly to minimize bus traffic, but employs write-through on previously unmodified data in order to assure exclusivity and minimize the writing of unmodified data. This method is usually used for the multi-cache system because multiple caches present serious coherency problems. In this case, two bits defining one of the four states for the

associated data are associated with each line. The four states are Invalid, Valid, Reserved, and Dirty.

### 3. Simulation

The most accurate method of determining the performance of a specific computer design, or proving the validity of a new architectural approach, is to actually build it and verify it's performance. Actually building a complete computer system just for verification of a certain improvement, however, is very time consuming and expensive. It also requires the designer to select specific values for architectural parameters of the new system without knowing what reasonable values of the parameters may be for that new system. Therefore, before actually building it while consuming time and a great deal of labor, it is desirable to explore the limit of the design space using simulation technology. A large number of potential design option can be quickly examined by varying the various parameters.

#### 3.1 Simulator Program Used

##### 3.1.1 Traces Used

In order to evaluate the behavior of the cache memory working on a single bus structured computer, a simulator program had been developed by the author using C programming language which is run on the SUN SPARC work station. The program uses four different real traces which is extracted from VAX computer. These traces have been supplied by MORGAN KAUFMANN PUBLISHERS, INC. Each line of the trace file contains a 8-bit label and 32-bit virtual address. The label gives the access type of a reference.

0: read data

1: write data  
 2: instruction fetch  
 4: escape record(causes cache flush)

The address is a hexadecimal virtual address between 0 and ffffffff inclusively. As a matter of fact, VAX series has variable length instructions and it uses the Synchronous Backplane Interconnection (SBI) bus, out of which 32-bit lines are allocated for information transfer. Thus, CPU fetches 32-bit information from main memory whenever it needs memory access. An instruction consists of a 1- or 2-byte op code followed by from zero to six operand specifiers, depending on the op code. The minimum length of an instruction is 1 byte, and instructions of up to 37 bytes can be constructed[6]. Since the CPU does not know the length of the next instruction to be fetched, a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction. Thus, without cache memory, each access sometimes fetch useless portion of main memory locations, that is, multiple instructions, or it can spend more than one memory cycle for only one instruction fetch. In the case of the system which has cache memory in it, it can do useless portion, but it stores them into cache memory for later use. As we will see later in this chapter, therefore, the bus traffic of the cache structured system makes a big difference from the one which does not have the cache memory in it.

Table 1a  
 Direct Mapping

Name of outfile = dirtot.d	
Write Policy used = Write back	
Trace Program1 used = lisp.dat.	291390
Trace Program2 used = pasc.dat.	422090
Trace Program3 used = forf.dat.	368212
Trace Program4 used = macr.dat.	342828

	Bus Traffic Ratio							
	256	512	1K	2K	4K	8K	16K	32K
8	117.01	101.00	81.23	61.85	45.31	34.03	22.23	15.77
16	190.95	160.77	128.97	98.30	72.25	54.99	35.68	24.72
32	352.85	289.37	230.21	171.52	125.89	97.32	65.38	45.37
64	754.28	595.31	464.37	335.47	253.09	198.11	132.74	94.58
128	1806.10	1342.49	1027.60	714.00	533.23	420.90	279.48	202.63
256	4717.60	3420.15	2550.95	1797.08	1282.42	1009.78	630.86	469.86
512	9435.19	9312.19	6746.10	4792.54	3514.14	2635.36	1519.46	1120.90

	Miss Ratio							
	256	512	1K	2K	4K	8K	16K	32K
8	48.52	41.61	33.06	25.37	18.52	13.76	8.98	6.46
16	39.18	32.73	25.97	19.80	14.45	10.85	7.07	4.92
32	36.22	29.25	22.90	16.90	12.29	9.35	6.27	4.32
64	39.05	30.35	23.11	16.44	12.30	9.52	6.33	4.51
128	47.55	34.57	25.79	17.49	12.95	10.12	6.63	4.81
256	63.03	44.91	32.52	22.36	15.53	12.16	7.44	5.56
512	63.03	62.12	43.81	30.44	21.70	15.92	8.85	6.50

	Trace1	Trace2	Trace3	Trace4
Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

### 3.1.2 Simulator Program Study

The program simulates various kinds of unified cache memory structures using direct mapping or n-way set associative mapping functions. It adapts write-back replacement policy and write allocation.

Table 1b  
 Direct Mapping

Name of outfile = dirtot.d	
Write Policy used = Write back	
Trace Program1 used = lisp.dat.	291390
Trace Program2 used = pasc.dat.	422090
Trace Program3 used = forf.dat.	368212
Trace Program4 used = macr.dat.	342828

	Bus Traffic Ratio							
	4K	8K	16K	32K	64K	128K	256K	512K
8	45.31	34.03	22.23	15.77	11.93	9.27	7.53	6.97
16	72.25	54.99	35.68	24.72	17.94	13.59	10.82	9.68
32	125.89	97.32	65.38	45.37	29.81	21.81	17.02	14.66
64	253.09	198.11	132.74	94.58	55.10	39.24	30.08	25.32
128	533.23	420.90	279.48	202.63	117.47	83.78	63.62	53.34
256	1282.42	1009.78	630.86	469.86	263.56	192.08	145.54	121.93
512	3514.14	2635.36	1519.46	1120.90	649.55	484.90	335.18	279.15

	Miss Ratio							
	4K	8K	16K	32K	64K	128K	256K	512K
8	18.52	13.76	8.98	6.46	5.06	4.08	3.50	3.32
16	14.45	10.85	7.07	4.92	3.69	2.90	2.45	2.27
32	12.29	9.35	6.27	4.32	2.95	2.24	1.86	1.68
64	12.30	9.52	6.33	4.51	2.65	1.95	1.59	1.41
128	12.95	10.12	6.63	4.81	2.78	2.04	1.64	1.45
256	15.53	12.16	7.44	5.56	3.04	2.26	1.80	1.58
512	21.70	15.92	8.85	6.50	3.63	2.74	2.01	1.74

	Trace1	Trace2	Trace3	Trace4
Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

When the program is executed it accepts various simulation parameters including mapping procedure to be used(direct mapping or n-way set-associative mapping), the desired starting cache size, the number of trace files to be used, the number of main memory blocks which occupy a cache line, and the input file name(s) and output file name. When all the simulation parameters are given, it reads a 5-byte information from the given trace file, which consists of a 1-byte label and a 4-byte virtual address, and it calculates bus traffic ratio and miss ratio for each of 56 cache organizations starting from the desired cache size entered and 8-byte line size, while increasing the line size up to 512 bytes (starting line size is 8 bytes) and cache size. As a result, it gives us a bus traffic ratio and miss ratio table for 56 cache organizations at once(see Table 2, 3, 4, 5). When the number of trace files greater than one is entered, it calculates bus traffic and miss ratios for each of all the trace files entered and then takes an arithmetic averages for each of bus traffic and miss ratios, so that it gives much more reliable and realistic result than the one of a specific trace file.(Table 1,

6 - 10)

Table 2

4-way Set Associative Mapping  
 Name of outfile = set4lisp.d  
 Write Policy used = Write\_back  
 Trace Program1 used = lisp.dat, 291390

	Bus Traffic Ratio							
	8K	16K	32K	64K	128K	256K	512K	1M
8	5.69	3.25	2.33	2.15	2.15	2.15	2.15	2.15
16	8.75	4.46	2.81	2.44	2.43	2.43	2.43	2.43
32	15.85	7.11	3.92	2.92	2.87	2.87	2.87	2.87
64	35.15	15.02	6.70	3.83	3.56	3.55	3.55	3.55
128	97.83	38.65	18.23	5.78	4.60	4.56	4.56	4.56
256	362.01	146.87	62.16	15.35	6.81	6.19	6.19	6.19
512	1119.84	554.10	331.96	130.42	12.74	9.09	9.05	8.83

	Miss Ratio							
	8K	16K	32K	64K	128K	256K	512K	1M
8	2.28	1.39	1.12	1.08	1.08	1.08	1.08	1.08
16	1.77	0.93	0.67	0.61	0.61	0.61	0.61	0.61
32	1.61	0.73	0.45	0.36	0.36	0.36	0.36	0.36
64	1.80	0.76	0.37	0.24	0.22	0.22	0.22	0.22
128	2.54	0.96	0.47	0.17	0.14	0.14	0.14	0.14
256	4.65	1.81	0.77	0.20	0.10	0.10	0.10	0.10
512	7.29	3.59	2.13	0.81	0.09	0.07	0.07	0.07

Trace1	
Fraction of Inst. fetches	0.58
Fraction of Data fetches	0.34
Fraction of Writes	0.08
Number of Cache Flushes	0

### 3.2 Simulation Result Analysis

Cache performance is greatly affected by cache parameters, particularly total size and line size. In addition, performance varies greatly depending upon the mapping procedure used and the trace program used, say, application executed. The simulator program produces a table to meet various simulation parameters entered. The table shows the bus traffic ratios and miss ratios for various cache memory organizations. The bus traffic ratio is the ratio of the total traffics of the cache organized system to the total traffics of



the noncache organized system. In this section, the simulation result is analyzed. Especially, the produced table shows the memory pollution phenomena for relatively small cache sizes. That is, there is a line size point where the miss ratio is not reduced further, rather increased, even though the line size is increased.

Table 3

4-way Set Associative Mapping

Name of outfile = set4pasc.d  
 Write Policy used = Write\_back  
 Trace Program1 used = pasc.dat. 422090

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	33.16	12.66	5.20	3.34	3.14	3.12	3.12	3.12
16	57.29	24.90	9.00	4.50	3.93	3.83	3.83	3.83
32	101.07	49.15	19.25	6.83	5.17	4.80	4.80	4.80
64	176.41	101.89	45.80	12.89	7.49	6.21	6.13	6.13
128	337.87	213.91	105.17	33.46	12.43	8.58	8.07	8.07
256	649.34	437.84	250.59	102.21	37.15	12.43	10.90	10.74
512	1315.21	958.58	597.04	297.19	112.17	37.27	15.41	15.04

Miss Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	13.81	5.31	2.26	1.61	1.56	1.56	1.56	1.56
16	11.90	5.23	1.90	1.06	0.97	0.96	0.96	0.96
32	10.40	5.15	1.99	0.77	0.63	0.60	0.60	0.60
64	8.98	5.24	2.35	0.70	0.44	0.39	0.38	0.38
128	8.49	5.38	2.61	0.88	0.35	0.26	0.25	0.25
256	7.92	5.34	3.05	1.30	0.49	0.18	0.17	0.17
512	7.92	5.74	3.54	1.83	0.69	0.26	0.12	0.12

Tracel

Fraction of Inst. fetches 0.46  
 Fraction of Data fetches 0.29  
 Fraction of Writes 0.25  
 Number of Cache Flushes 0

Table 4

4-way Set Associative Mapping

Name of outfile = set4forf.d  
 Write Policy used = Write\_back  
 Trace Program = forf.dat. 368212

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	23.63	19.00	14.44	11.12	9.36	8.78	8.69	8.68
16	33.74	27.11	20.74	15.38	12.44	11.18	10.96	10.94
32	52.36	41.42	31.98	22.84	17.75	14.97	14.30	14.22
64	95.51	66.83	52.35	37.47	27.05	21.13	19.12	18.83
128	202.69	120.78	89.92	65.00	44.51	31.69	26.21	25.21
256	453.56	247.67	163.00	117.62	79.75	51.92	37.53	34.19
512	1170.04	646.83	356.28	226.37	154.03	95.35	59.48	48.46

Miss Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	9.49	7.69	6.00	4.88	4.44	4.35	4.34	4.34
16	6.59	5.32	4.15	3.25	2.86	2.75	2.73	2.73
32	4.97	3.92	3.06	2.29	1.94	1.81	1.78	1.78
64	4.44	3.04	2.40	1.77	1.38	1.22	1.18	1.18
128	4.61	2.66	1.98	1.45	1.05	0.86	0.80	0.79
256	4.98	2.65	1.73	1.25	0.88	0.64	0.55	0.53
512	6.25	3.38	1.83	1.16	0.80	0.54	0.40	0.37

Tracel

Fraction of Inst. fetches 0.52  
 Fraction of Data fetches 0.29  
 Fraction of Writes 0.19  
 Number of Cache Flushes 0

3.2.1 Direct mapping versus n-way

Set- Associative mapping

The main argument for using a set associative cache rather than a direct mapped cache is its better hit ratio. As we briefly mentioned about mapping procedures in chapter 2, direct mapping is simple, fast, and easy to implement. However, it has worse hit ratio than set-associative mapping. We simulated

Table 5

4-way Set Associative Mapping

Name of outfile = set4macr.d  
 Write Policy used = Write\_back  
 Trace Program1 used = macr.dat. 342828

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	21.11	17.81	15.18	12.05	9.08	8.15	7.94	7.93
16	32.43	27.38	22.97	18.92	13.97	11.44	10.77	10.72
32	53.97	45.54	37.77	30.91	23.37	17.23	15.15	14.88
64	98.06	80.26	66.92	52.51	40.04	27.81	21.78	20.53
128	208.84	153.00	127.07	98.14	70.40	46.97	32.76	28.52
256	549.82	322.46	250.75	192.90	132.71	82.72	52.59	40.01
512	1586.73	868.52	541.57	390.88	262.59	146.43	87.78	56.68

Miss Ratio								
	8K	16K	32K	64K	128K	256K	512K	1M
8	7.91	6.71	5.80	4.92	4.18	4.00	3.97	3.96
16	5.78	4.90	4.18	3.59	3.01	2.74	2.68	2.68
32	4.57	3.85	3.24	2.74	2.29	1.96	1.87	1.86
64	3.97	3.21	2.69	2.19	1.80	1.46	1.32	1.28
128	4.20	2.94	2.43	1.91	1.46	1.13	0.94	0.89
256	5.47	3.05	2.32	1.78	1.28	0.90	0.70	0.62
512	7.94	4.14	2.47	1.75	1.19	0.73	0.53	0.43

Trace1	
Fraction of Inst. fetches	0.55
Fraction of Data fetches	0.28
Fraction of Writes	0.17
Number of Cache Flushes	0

various kinds of cache organization including direct mapping, and 2, 4, 8, 16, 32-way set associative mapping. As a result, we found that 16-way set associative mapping was the best configuration, say, it shows the lowest miss ratio and bus traffic ratio. Refer to Table 1b, 6, 7, 8, 9, 10, and Figure 4, 5, 6, 7.

When we run our simulator program with same trace files and the same number of references, in the case of 8KB 16-way set-associative mapping, miss ratio is reduced more than 50 percent compared to the direct mapping, and 2-way mapping improves approximately 40%(see Table 1b, 7, 9, and Figure5, 7). In [2,7], the authors reported that using a 2-way set associative cache in place of a direct mapped cache removes about 30 percent of the misses.

In this paper, we can also observe dramatic improvement in bus traffic ratio. The bus traffic ratio declines much more dramatically than the miss ratio as the line size is increased. In the case of 8KB 16-way set-associative mapping, bus traffic ratio is reduced more than 50 percent compared to the direct mapping, and 512KB cache improves approximately 80%(see Table 1b, 9, and Figure4, 6).

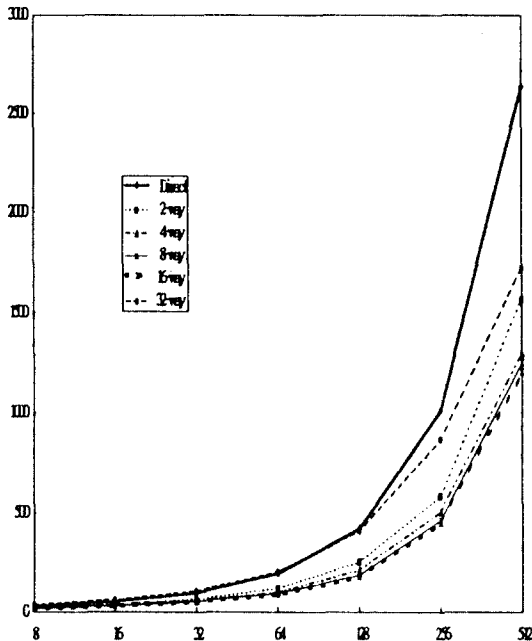


Figure 4. Bus Traffic Ratios for 8KB Cache

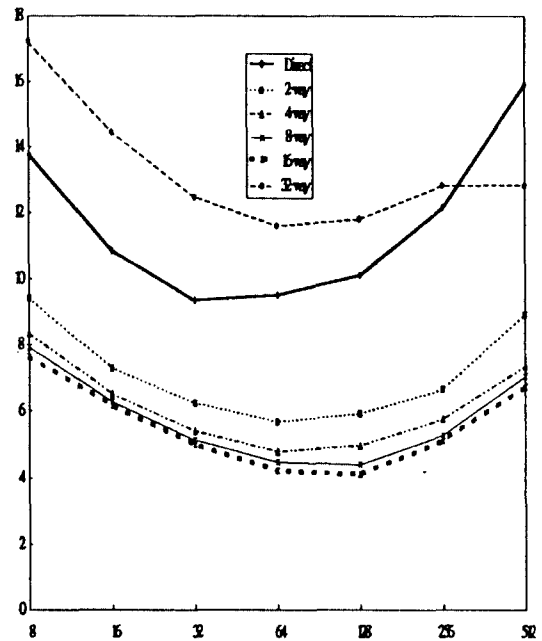


Figure 5. Miss Ratios for 8KB Cache

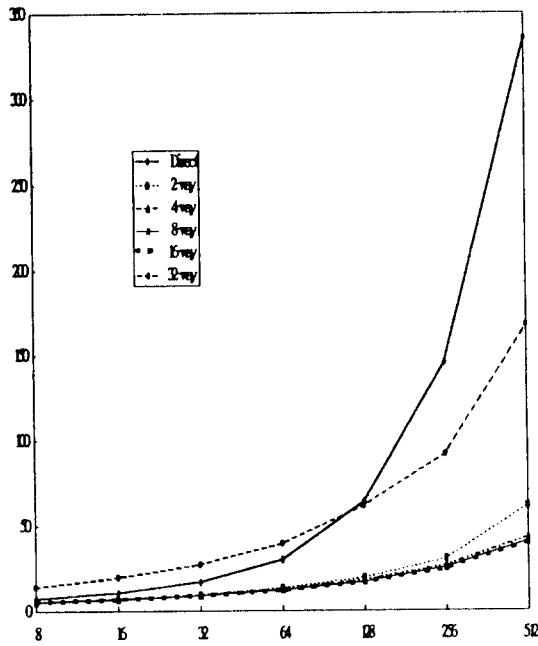


Figure 6. Traffic Ratios for 512KB Cache

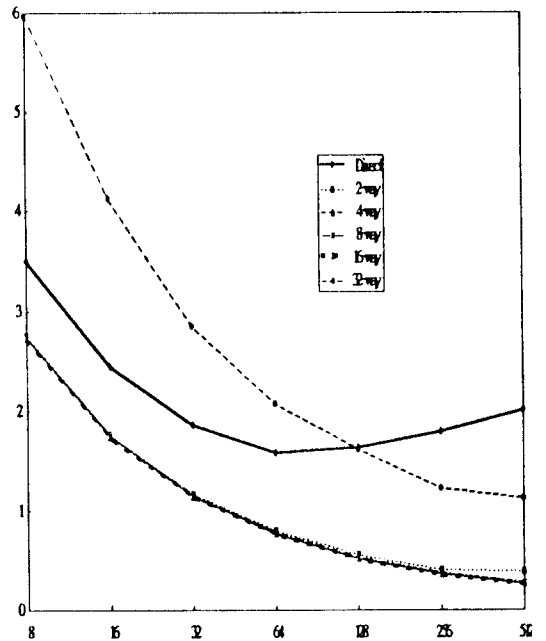


Figure 7. Miss Ratios for 512KB Cache

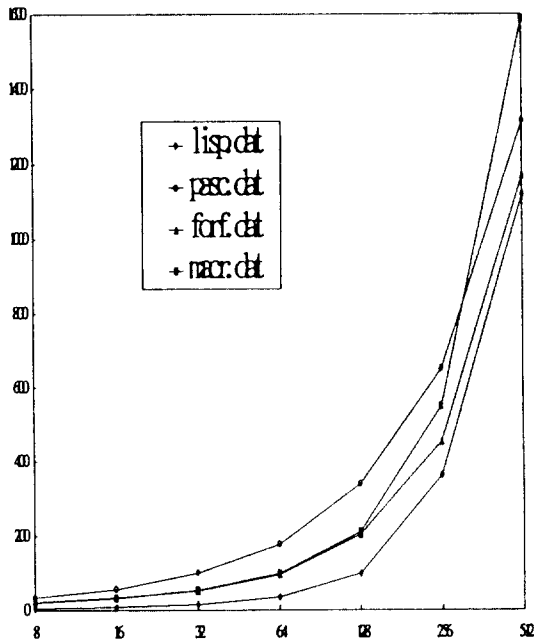


Figure 8. Bus Traffic Ratios for 8KB Cache

### 3.2.2 Characteristics of Applications (Trace Files)

The cache performance greatly varies depending upon the trace files, cache sizes, and line sizes used. The results are shown in Table 2 - 5. In tables, the bus traffic and miss ratios for each separate trace file are shown for a variety of cache sizes, with the line sizes of 8, 16, 32, 64, 128, 256, and 512 bytes. The characteristics of each trace file for 8KB and 512KB cache memories are depicted in Figure 8, 9, 10, and 11.

For a relatively small cache (8KB), trace file lisp.dat shows the best bus traffic and miss ratios and pasc.dat does the worst for relatively small line sizes up to 256B. As the line size gets larger than 256B, however, lisp.dat gives worse results due to memory pollution which will be mentioned in next section, and forf.dat gives the best results.

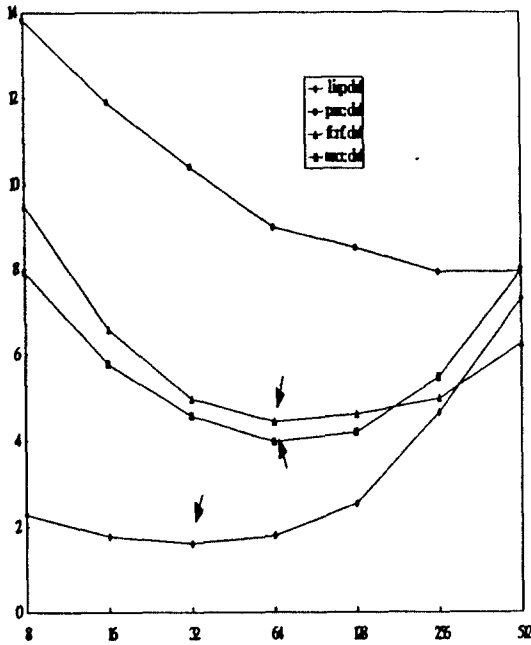


Figure 9. Miss Ratios for 8KB Cache

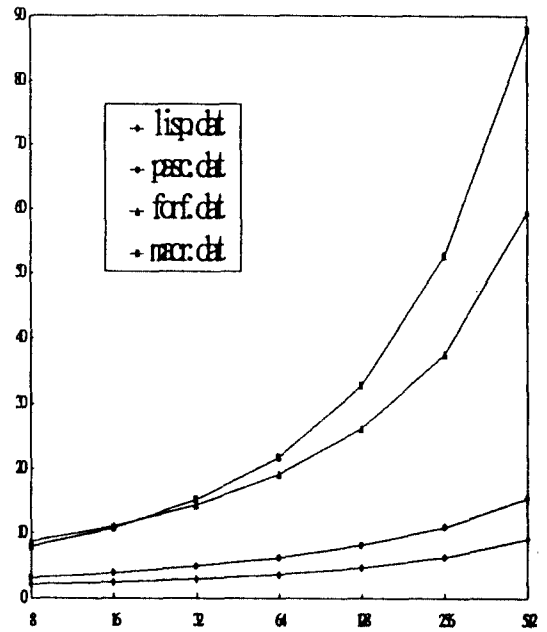


Figure 10. Bus Traffic Ratios for 512KB Cache

As we see from Table 2, 3, 4, and 5, lisp.dat has a relatively small write operation fraction (just 8% of the whole references) and pasc.dat has the largest fraction of writes (25%).

From this, we understand that lisp.dat has poor spacial locality, but it shows nice results due to the relatively small fraction of writes. The trace file pasc.dat has nice spacial locality, however, so that it could give favorable results when the cache size gets larger.

For a relatively large cache memory (see Figure 10 and 11), as we could imagine, lisp.dat and pasc.dat give improved performance due to the small write fraction and the beautiful spacial locality, respectively.

### 3.2.3 Line Sizes and Memory pollution

In general, as the line size is increased, the bus traffic is increased and the miss ratio is

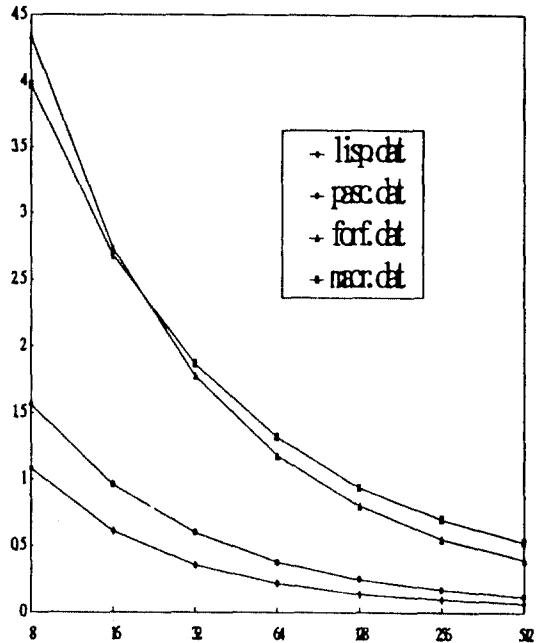


Figure 11. Miss Ratios for 512KB Cache

decreased. For the small cache sizes, keeping the total size constant (for example, 8K cache), as the line size is increased, the miss ratio generally declines up to a point - 64 bytes for 8K cache, 128 bytes for 16K cache - then increases again (see Table 6 or Figure 5). This increase, known as memory pollution [8], results from the fact that there should be fewer lines due to large line size, so some unneeded data occupies cache instead of useful data that happens not to be contiguous (see Table 6). The miss ratio declines by 45 percent as the line size increased from 8 to 64 bytes. However, as the cache size increases (for example, 512K cache), memory pollution does not occur because there are enough numbers of lines in cache (see Figure 7). Same thing will be true for the larger cache organizations with larger line size. Thus, in order to for us to avoid memory pollution, we'd better increase cache size if possible.

### 3.2.4 Optimal Line Size for a Uniprocessor System

As we see from the tables, as the line size is increased, the miss ratio generally declines favorably. In contrast to the miss ratio, however, the bus traffic always increases as the line size is increased. The bus traffic grows more rapidly than the miss ratio declines for all the cache sizes. Thus, for a uniprocessor system, the line size will be able to be increased in order to decrease the miss ratio as long as the bus traffic falls down within the range of 100% and memory pollution is not occurred. With 8K cache, for example, we can use 64B line with little bus latency, and for 64K cache we can further increase the line size up to 256B without any performance degradation. For 256K cache, 512B line will

be good enough for the proper operation (see Table 6).

Table 6  
4-way Set Associative Mapping

Name of outfile = set4tot.d								
Write Policy used = Write back								
Trace Program1 used = lisp.dat, 291390								
Trace Program2 used = pasc.dat, 422090								
Trace Program3 used = forf.dat, 368212								
Trace Program4 used = macr.dat, 342828								
Bus Traffic Ratio								
	8K	16K	32K	64K	128K	256K	512K	1M
8	20.90	13.18	9.29	7.17	5.93	5.55	5.47	5.47
16	33.06	20.96	13.88	10.31	8.19	7.22	7.00	6.98
32	55.81	35.80	23.23	15.88	12.29	9.97	9.28	9.19
64	101.28	66.00	42.94	26.67	19.53	14.67	12.65	12.26
128	211.81	131.58	85.10	50.59	32.98	22.95	17.90	16.59
256	503.68	288.71	181.63	107.02	64.10	38.32	26.80	22.78
512	1297.95	757.01	456.71	261.21	135.38	72.04	42.93	32.25
Miss Ratio								
	8K	16K	32K	64K	128K	256K	512K	1M
8	8.37	5.27	3.79	3.12	2.82	2.75	2.74	2.73
16	6.51	4.09	2.73	2.13	1.86	1.76	1.75	1.74
32	5.39	3.41	2.19	1.54	1.31	1.18	1.15	1.15
64	4.80	3.06	1.95	1.22	0.96	0.82	0.78	0.77
128	4.96	2.98	1.88	1.10	0.75	0.60	0.53	0.52
256	5.75	3.21	1.97	1.13	0.69	0.46	0.38	0.35
512	7.35	4.21	2.49	1.38	0.69	0.40	0.28	0.25
Trace1 Trace2 Trace3 Trace4								
Fraction of Inst. fetches				0.58	0.46	0.52	0.55	
Fraction of Data fetches				0.34	0.29	0.29	0.28	
Fraction of Writes				0.08	0.25	0.19	0.17	
Number of Cache Flushes				0	0	0	0	

### 3.2.5 The Optimal Number of Processors

which can be supported by a Single Bus

Since the gap between main memory access time and processor cycle time is continuously increasing, processor performance dramatically depends on the behavior of caches, and particularly on the behavior of small on-chip caches. Most of the recently introduced microprocessors have relatively small on-chip caches. High clock frequency and/or parallel

instruction issuing are used, yet main memory access time remained approximately constant for several years(around 100ns) and relative miss penalties become very high even when second-level caches are used. As cache misses induce long pipeline stall, reducing the miss ratio has become an important challenge for microprocessor designers. If the miss ratio is low, the cost of cache misses doesn't dominate on execution time and if the cache access time is short, the cache does not slow

down the clock frequency. Therefore, CPU will feel likely as if it should have a large main memory which has the short access time of cache memory. However, in the case of the multiprocessor system which has a single shared bus, it was well known that a low miss ratio will increase the bus traffic ratio which is the greatest bottleneck in the single bus multiprocessor systems. With high hit ratio the single shared bus can't support the given number of processors without bus latencies due to the burst of traffic. Thus, by compromising these two aspects, we can reduce the effective memory access time and bus traffic ratio for the single shared bus multiprocessor system.

Table 7

2-way Set Associative Mapping

Name of outfile = set2tot.d	
Write Policy used = Write_back	
Trace Program1 used = lisp.dat.	291390
Trace Program2 used = pasc.dat.	422090
Trace Program3 used = forf.dat.	368212
Trace Program4 used = macr.dat.	342828

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	23.44	15.73	10.57	7.93	6.53	5.78	5.54	5.49
16	36.92	24.75	16.25	11.78	9.25	7.67	7.14	7.01
32	64.29	42.89	27.34	19.06	14.04	10.83	9.60	9.26
64	120.11	82.38	51.59	34.26	22.89	16.11	13.37	12.42
128	252.52	168.60	102.93	65.22	39.84	25.49	19.40	16.99
256	581.77	399.81	232.99	146.53	78.27	44.66	30.47	24.19
512	1563.46	1020.67	589.52	348.38	175.77	93.29	61.14	45.76

Miss Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	9.42	6.32	4.33	3.40	3.01	2.80	2.75	2.74
16	7.28	4.85	3.21	2.41	2.05	1.82	1.76	1.75
32	6.23	4.07	2.60	1.84	1.47	1.24	1.17	1.15
64	5.67	3.78	2.35	1.57	1.12	0.87	0.80	0.77
128	5.91	3.82	2.29	1.43	0.91	0.65	0.56	0.52
256	6.64	4.49	2.59	1.57	0.84	0.52	0.41	0.36
512	8.93	5.79	3.36	1.91	0.93	0.53	0.39	0.33

Trace1 Trace2 Trace3 Trace4

Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

Uptill now, the system which has a single bus has been simulated. The bus system which has 32 information lines is assumed. An four-byte transfer, therefore, is counted as one cycle. The traces are for a 32-bit machine, and so a minimum of 32-bit bus is a more realistic assumption. The bus traffic

Table 8

8-way Set Associative Mapping

Name of outfile = set8tot.d	
Write Policy used = Write_back	
Trace Program1 used = lisp.dat.	291390
Trace Program2 used = pasc.dat.	422090
Trace Program3 used = forf.dat.	368212
Trace Program4 used = macr.dat.	342828

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	19.84	12.26	8.73	7.00	5.72	5.51	5.47	5.47
16	31.96	19.25	12.80	10.01	7.86	7.11	6.98	6.98
32	53.38	33.98	20.91	15.15	11.90	9.64	9.21	9.19
64	95.46	62.99	37.94	25.16	18.75	14.01	12.40	12.25
128	189.70	122.33	76.02	47.05	30.60	21.99	17.24	16.52
256	462.82	264.55	162.16	98.96	55.82	36.66	25.51	22.62
512	1249.60	684.22	366.67	217.32	118.20	63.75	40.59	31.65

	Miss Ratio							
	8K	16K	32K	64K	128K	256K	512K	1M
8	7.92	4.88	3.58	3.06	2.77	2.74	2.73	2.73
16	6.27	3.73	2.51	2.07	1.82	1.75	1.75	1.74
32	5.13	3.24	1.95	1.48	1.28	1.16	1.15	1.15
64	4.48	2.94	1.70	1.15	0.93	0.80	0.77	0.77
128	4.41	2.78	1.66	1.01	0.70	0.58	0.52	0.52
256	5.27	2.94	1.74	1.04	0.59	0.44	0.37	0.35
512	7.04	3.78	1.94	1.13	0.60	0.35	0.27	0.25

	Trace1	Trace2	Trace3	Trace4
Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

ratio shown in each table is given as a percent of the number of accesses that would be required if no cache were present. If we fix the cache size 256K bytes, 64-byte line size shows 99.21% hit ratio and 13.62% bus traffic ratio(see Table 9). This tells us when we use 256K byte cache and 64-byte line size for a uniprocessor system, more than 85% of the total executing time keeps the bus idle! For the line sizes which are smaller than 64 bytes the effect is even more dramatic! for 8-byte line size the bus traffic ratio is 5.5% so that only 5.5% out of the total execution time makes the bus busy with 95.26% hit ratio which is not that bad. Thus, if bus traffic is not a potential bottleneck, the greater the line size, the better the system performance. With 512-byte line size, for example, we can get 99.66 % hit ratio without any bus delay.

Increasing the line size from one bus cycle(four bytes) to two(eight bytes), however, decreases the miss ratio by 30 to 40 percent increasing the bus traffic by 20 percent. As we increase the line size further, however, the bus traffic is increased much more rapidly than the miss ratio declines. What if we have multiple processors to be supported

by this single bus at the same time? If ten processors were to be supported, 32 byte line size is optimal with 98.85% hit ratio. Eight byte line size would be optimal for twenty processor system. Thus, the optimal line size

Table 9

## 16-way Set Associative Mapping

```
Name of outfile = set16tot.d
Write Policy used = Write_back
Trace Program1 used = lisp.dat,      291390
Trace Program2 used = pasc.dat,     422090
Trace Program3 used = forf.dat,     368212
Trace Program4 used = macr.dat,     342828
```

## Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	19.21	11.72	8.49	6.99	5.62	5.50	5.47	5.47
16	31.35	18.30	12.36	9.97	7.64	7.08	6.98	6.98
32	52.02	32.75	20.26	14.86	11.79	9.52	9.20	9.19
64	90.23	61.72	36.29	24.23	18.64	13.62	12.31	12.25
128	178.60	120.16	71.61	45.23	30.01	21.62	16.93	16.52
256	448.84	250.05	160.14	89.86	54.17	36.03	24.82	22.53
512	1203.06	640.94	352.45	205.41	110.36	60.45	39.99	31.34

## Miss Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	7.64	4.66	3.48	3.06	2.75	2.74	2.73	2.73
16	6.15	3.53	2.43	2.06	1.80	1.75	1.74	1.74
32	5.00	3.09	1.89	1.45	1.27	1.16	1.15	1.15
64	4.23	2.86	1.61	1.11	0.92	0.79	0.77	0.77
128	4.11	2.72	1.55	0.97	0.69	0.57	0.52	0.52
256	5.11	2.76	1.72	0.92	0.57	0.43	0.36	0.35
512	6.73	3.51	1.86	1.05	0.55	0.34	0.27	0.24

## Trace1 Trace2 Trace3 Trace4

Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

varies depending upon the number of processors to be supported by a single bus. However, we have another problem related with this small line size. Small lines are very much costly in that they greatly increase the overhead of the cache: an address tag, n(say, 16 here) counters for replacement policy, and

one status bit are normally stored in the cache for each line. The smaller the line size is, the more lines, and so the more tag bits should be stored in the cache. This is one of the problems which needs further study.

Table 10

32-way Set Associative Mapping

Name of outfile = set321tot.d  
 Write Policy used = Write\_back  
 Trace Program1 used = lisp.dat. 291390  
 Trace Program2 used = pasc.dat. 422090  
 Trace Program3 used = forf.dat. 368212  
 Trace Program4 used = macr.dat. 342828

Bus Traffic Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	37.64	25.44	19.07	15.25	13.59	13.61	14.36	15.72
16	63.47	43.32	29.85	23.32	19.91	18.90	19.81	21.41
32	109.29	74.35	51.11	36.90	30.40	26.95	27.41	30.30
64	202.86	134.82	92.03	62.76	47.34	41.40	39.38	42.64
128	405.05	269.61	181.38	113.49	83.76	64.28	61.23	60.74
256	865.29	598.53	380.37	237.61	154.17	112.72	91.59	89.30
512	none	1320.94	865.12	527.03	325.64	213.04	167.10	136.83

Miss Ratio

	8K	16K	32K	64K	128K	256K	512K	1M
8	17.23	11.72	8.73	6.90	6.06	5.93	5.96	5.98
16	14.44	9.90	6.82	5.23	4.41	4.10	4.12	4.10
32	12.47	8.52	5.82	4.12	3.34	2.91	2.85	2.91
64	11.60	7.76	5.24	3.49	2.61	2.25	2.07	2.07
128	11.80	7.77	5.18	3.21	2.33	1.76	1.62	1.49
256	12.83	8.70	5.48	3.36	2.15	1.53	1.23	1.11
512	none	9.75	6.33	3.74	2.25	1.48	1.13	0.87

Trace1 Trace2 Trace3 Trace4

Fraction of Inst. fetches	0.58	0.46	0.52	0.55
Fraction of Data fetches	0.34	0.29	0.29	0.28
Fraction of Writes	0.08	0.25	0.19	0.17
Number of Cache Flushes	0	0	0	0

### 4. Conclusion

As the integration density increases, caches may be integrated on the microprocessor chip. Yet, on-chip cache sizes remain limited, therefore the available space must be cautiously used. The objective of this work is to

investigate the effect of cache memory on the system performance. Thus, we have developed a cache memory simulator program, which simulates various kinds of unified cache organizations and produce useful information. In our program, 32-bit bus, direct or n-way set associative mapping functions, LRU replacement policy, write back policy, and write allocation are introduced. We simulated various kinds of cache organization including direct mapping, and 2, 4, 8, 16, 32-way set associative mapping. As a result, we found that 16-way set associative mapping was the best configuration, say, it shows the lowest miss ratio and bus traffic ratio. 32-way mapping does not improve the performance, rather degrades it.

As we mentioned in chapter 3, the system that has a small cache memory with relatively large line size creates a serious problem, called memory pollution. For us in order to avoid this problem, relatively large cache memory is suggested. For a uniprocessor system, we can select large line size for high hit ratio as long as the bus traffic ratio does not go over 100 percent and memory pollution can be avoided.

When we consider multiple processors in a single bus organized system, the optimal line size varies depending upon the number of processors to be supported by a single bus and favorable hit ratio if needed.

### References

1. J.L. Hennessy, D.A. Patterson, "Computer Architecture a Quantitative Approach", Morgan Kaufmann Publishers, Inc., 1990.
2. M.D. Hill, A.J. Smith, Evaluating Associativity in CPU Caches, IEEE Trans. on Comp., Dec. 1989.



3. A.J. Smith, Line(Block) size choice for CPU Cache Memories, IEEE Trans. on Comp. C-36, 9(Sept.), pp.348-354, 1987.
4. A.J. Smith, Cache Memories, ACM Computing Surveys, Vol.15, No.3, pp.473-530, 1982.
5. David J. Lilja, Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and

Comparisons, ACM Computing Surveys, Vol.25, No.3, pp.303-312, 1993.

6. William Stalling, "Computer Organization and Architecture", 2nd Edition, pp.317-357, 1990.
7. M.D. Hill, "Aspects of Cache Memory and Instruction Buffer Performance", Ph.D Thesis, Univ. of Berkeley, 1987.



趙龍勳(Yong Hoon Cho) 정회원

1953년 6월 18일생

1979년 2월 : 한국항공대학교 항공통신공학과(공학사)

1978년 12월~1982년 2월 : (주)대한항공 전자보기공장 전자과 ATEC 선임항공기술사

1985년 8월 : 미국 Purdue Univ. Dept. of Electrical Engineering(공학석사, 병렬처리 전공)

1992년 12월~1994년 2월 : 미국 Ball State Univ. Dept. of Computer Science 교육부 교환교수

1992년 3월~현재 : 한국항공대학교 대학원 전자공학과 박사과정

1986년 3월~현재 : 경북실업전문대학 전자계산과 부교수

\*주관심 분야 : 컴퓨터 구조학, 병렬처리



金正善(Jeong Sun Kim) 정회원

1941년 5월 5일생

1965년 2월 : 한국항공대학교 항공전자공학과(공학사)

1972년 2월 : 한양대학교 전자공학과(공학석사)

1983년 2월 : 경희대학교 전자공학과(공학박사)

1976년~현재 : 한국항공대학교 교수

1996년 1월~현재 : 한국통신학회 재무이사

1994년 6월~현재 : 전국대학전자계산소장협의회 회장

\*주관심 분야 : 컴퓨터구조학