# *Optimal Design of Megabyte Second-Level Caches for Minimizing Bus Traffic in Shared-Memory Shared-Bus Multiprocessors*

Yen-Jen Oyang and Le-Chun Wu
National Taiwan University

ABSTRACT: As the design of shared-memory shared-bus multiprocessors is heading toward employing megabyte second-level caches, how to optimize the design of the second-level caches in order to minimize the traffic on the shared memory bus and thus improve system scalability is of great interest. This paper presents a comprehensive study on this issue through extensive trace-driven simulation. The simulation results show that a good cache design could mean a reduction of bus traffic by more than 80 percent or, equivalently, an increase of system scalability by more than five times. Furthermore, they show that a few simple design guidelines can be derived because the optimal choice of cache configuration metrics exhibits a high degree of invariance over system-configuration variations.

This research was sponsored by the National Science Council of R.O.C. under grant NSC 79-0408-E-002-10.

## 1. Introduction

In recent years, the shared-memory shared-bus architecture has become a favorite scheme for building multiprocessors. One of the major issues in the design of such systems is how to minimize the traffic the CPUs would induce on the shared memory bus so that more CPUs can be incorporated. In this regard, a number of papers discussing how to achieve the goal with better cache designs and/or cache coherence protocols have been published [Archibald and Baer 1986; Karlin et al. 1988; Sites and Agarwal 1988; Eggers and Katz 1989]. Nevertheless, a comprehensive investigation on optimal design of megabyte second-level caches for minimizing bus traffic in shared-memory shared-bus multiprocessors is yet to be carried out. This issue is of significance because employing megabyte second-level caches will soon become a common practice in computer design due to two recent developments:

1. The introduction of commodity megabit memory chips makes the implementation of megabyte caches no longer unaffordable for most systems.
2. As computer architects turn to two-level cache design to overcome the widening gap between the CPU and main-memory speeds, the incorporation of megabyte caches at the second-level of the cache hierarchy is favored for maximizing system performance [Hennessy and Patterson 1990; Bugge et al. 1990].

Motivated by these observations, we carried out the study presented in this paper. In this study, we conducted extensive trace-driven simulation to determine the optimal organization of megabyte second-level caches under various system configurations. The system configuration metrics that we varied in the simulation runs include number of CPUs, width of the shared-memory bus, and main-memory access latency. The data collected in the simulation runs reveal several significant results. They show that a careful design could mean a reduction of bus traffic by more than 80 percent or, equivalently, an increase of system scalability by more than five times. Furthermore, they show that a few simple design guidelines can

be derived because the optimal choice of cache configuration metrics exhibits a high degree of invariance over system-configuration variations.

The rest of the paper is organized as follows. Section 2 discusses the methodology used in this study. Sections 3 and 4 present the simulation results and elaborate interesting observations, and Section 5 concludes the study.

## 2. Methodology

When discussing multiprocessor design optimization, we must take into account how the multiprocessor will be put into use, since multiprocessors installed for different purposes may run applications that have very different characteristics and behavior. In this paper, we concentrate on multiprocessors that are positioned to improve system throughput in a multiple-user/multiple-task environment. The reason is that multiprocessors serving this kind of purpose may account for the largest share of multiprocessor installations to this point. The study is carried out through a trace-driven simulation in which we used a collection of 15 traces, detailed in Appendix B, generated by SPARCSim [Sun 1989]. The effect of multitasking is simulated by having a fixed context switch interval of 16,000 memory references for all the traces. That is, each CPU is assumed to execute a section of code equivalent to 16,000 memory references from a task/process during each context switch interval. As far as process scheduling is concerned, we assume two different policies. The first policy assumes unrestricted process migration and that the system maintains one global ready queue for all the CPUs. Under this policy, processes are scheduled in the global ready queue based on a round-robin strategy. Unrestricted process migration means that processes are not stuck to any CPU or any group of CPUs. In other words, a process in the global ready queue will be scheduled to run at any available CPU regardless of the process's past trace of CPU visit. In contrast to the first policy, the second policy implements process affinity. With process affinity, the system maintains one ready queue for each CPU. A process is assigned to a CPU when it is created, and the process is stuck to the designated CPU during its life span.

Figure 1 shows the machine model used in the simulation. As one may note, the machine model has only one level of cache memory. This seems to contradict a previous argument claiming that the move toward employing two-level caches is a major reason behind the popularization of megabyte caches. However, as pointed out by Przybylski [1990], the existence of the lower-level caches can be ignored in the study of the behavior of higher-level caches as long as the higher-level caches are several times larger than the lower-level caches. We find this condition to be generally true because the typical size of first-level caches ranges from 4K Bytes

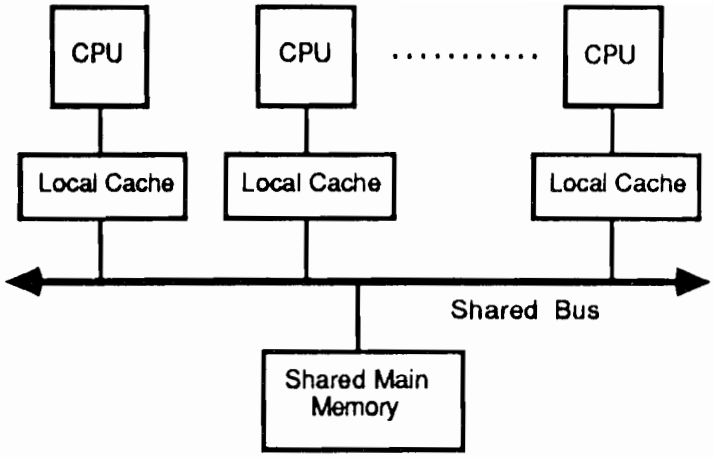



Figure 1. Machine model used in the simulation.

to 16K Bytes, whereas the size of second-level caches of interest in this paper is 1 megabyte or larger. Therefore, the machine model in Figure 1 is able to accurately reflect the behavior of the local second-level caches as observed from the shared-memory bus.

In the simulation, we assumed that the shared memory bus is a synchronous bus and executes the M Bus cache coherence protocol [Cypress 1991]. The M bus protocol is actually based on MOESI cache states proposed in [Sweazey and Smith 1986]. The basics of the M Bus protocol is summarized in Appendix A. In order to conduct a comprehensive study, we varied system configuration metrics in the simulation. These metrics include number of CPUs, width of the shared-memory bus, and main-memory access latency. We assumed the system had either 4 or 8 CPUs. The width of the shared memory bus refers to the number of data bits on the bus. We varied bus width from 32 bits, 64 bits, 128 bits, to 256 bits. We varied main-memory access latency to cover discrepancies in main-memory design, for example, differences in main-memory speed and degree of interleaving. We further assumed the following timing specifications:

- 2 clock-cycles latency for starting a new bus transaction
- a total of 6 clock cycles, including the 2 clock cycles for starting a new bus transaction, for executing an invalidation operation
- 1 clock cycle for transferring a piece of data over the shared-memory bus in the burst mode
- 4 clock cycles for accessing the second-level caches.

Table 1. Bus traffic in systems with 4 CPUs, a 64-bit bus, and a main memory with a latency of 30 clock cycles.

| Block size | Set association | Cache size 1M | 2M | 4M | 8M |
|---|---|---|---|---|---|
| 16 | 1 | 13633642 | 10678216 | 9371434 | 8652226 |
| | 2 | 9935848 | 8674758 | 7997158 | 7852498 |
| | 4 | 8754462 | 7963140 | 7819630 | 7789416 |
| | 8 | 8259730 | 7814290 | 7788290 | 7788052 |
| 32 | 1 | 9637928 | 7373534 | 6334436 | 5762714 |
| | 2 | 6865514 | 5807608 | 5241752 | 5107802 |
| | 4 | 5871494 | 5213292 | 5091194 | 5057784 |
| | 8 | 5493342 | 5091186 | 5057038 | 5056534 |
| 64 | 1 | 7862372 | 5948590 | 4999730 | 4488760 |
| | 2 | 5508930 | 4516668 | 4001318 | 3852792 |
| | 4 | 4568466 | 3955690 | 3837914 | 3803132 |
| | 8 | 4232144 | 3845882 | 3802880 | 3801842 |
| 128 | 1 | 7680542 | 5786994 | 4800154 | 4235424 |
| | 2 | 5402564 | 4299744 | 3741966 | 3547214 |
| | 4 | 4361608 | 3671172 | 3539966 | 3494420 |
| | 8 | 3970084 | 3559048 | 3495486 | 3493252 |
| 256 | 1 | 9167148 | 6885258 | 5685808 | 4909054 |
| | 2 | 6478272 | 5062362 | 4311282 | 4049386 |
| | 4 | 5221894 | 4219666 | 4038384 | 3976408 |
| | 8 | 4621316 | 4077286 | 3979634 | 3974472 |
| 512 | 1 | 12427602 | 9310830 | 7719284 | 6569004 |
| | 2 | 8825924 | 6767630 | 5785270 | 5387976 |
| | 4 | 7264188 | 5656548 | 5353184 | 5266136 |
| | 8 | 6339552 | 5432108 | 5273356 | 5262708 |
| 1024 | 1 | 22386056 | 17155884 | 12794600 | 9714350 |
| | 2 | 13239876 | 10270036 | 8519834 | 7858452 |
| | 4 | 11102462 | 8385648 | 7786710 | 7635352 |
| | 8 | 9733190 | 7957402 | 7651286 | 7627500 |

In this study, the amount of traffic on the shared-memory bus is measured by the number of clock cycles taken to complete all bus operations in a simulation run. This measurement can more accurately reflect the real situation than

just counting total number of bytes transferred over the bus because some coherence operations such as invalidation do not involve data transfer. What is missing in this measurement is the time taken to carry out bus arbitration when multiple bus masters issue requests at the same time. Table 1 shows a sample of the data collected in the simulation. Each entry in the table represents the amount of bus traffic measured under a specific cache configuration and a specific system configuration. For example, the data in Table 1 were collected assuming the system had 4 CPUs, a 64-bit bus, and a main memory with a latency of 30 clock cycles. As mentioned earlier, we conducted simulation runs under a wide range of system configuration variations and two process scheduling policies, process migration and process affinity. Therefore, we actually created 48 such tables and derived the observations discussed in the following sections based on these tables.

## 3. *Optimization of Cache Design*

This section and the next elaborate optimal design of megabyte second-level caches for minimizing bus traffic in shared-memory shared-bus multiprocessors. This section presents simulation results assuming unrestricted process migration, and the next addresses the effect of process affinity.

### 3.1. *Significance of Optimal Design*

The first observation on the collected data is that good cache design is crucial for minimizing bus traffic in shared-memory shared-bus multiprocessors with megabyte second-level caches. Table 2 shows the bus-traffic ratio between two cache designs under various system configurations. The first design, the denominator, is a 1 megabyte cache with direct mapping and 16-byte blocks. The second design, the numerator, is a 4 megabyte cache with 2-way set associativity and 128-byte blocks. Table 2 reveals that a reduction of bus traffic up to more than 80 percent would result due to a good cache design.

Table 2. Bus traffic ratio between two cache designs under various system configurations. The first design, the denominator, is a 1 megabyte cache with direct mapping and 16-byte blocks. The second design, the numerator, is a 4 megabyte cache with 2-way set associativity and 128-byte blocks.

| Bus width | CPU #=4 | CPU #=8 |
|---|---|---|
| 32 bits | 47.3% | 53.2% |
| 64 bits | 33.6% | 37.8% |
| 128 bits | 25.7% | 28.8% |
| 256 bits | 22.0% | 24.7% |

(a) Main-memory access latency = 20 bus cycles.

| Bus width | CPU #=4 | CPU #=8 |
|---|---|---|
| 32 bits | 38.2% | 43.2% |
| 64 bits | 27.5% | 31.1% |
| 128 bits | 21.5% | 24.4% |
| 256 bits | 18.7% | 21.2% |

(b) Main-memory access latency = 30 bus cycles.

| Bus width | CPU #=4 | CPU #=8 |
|---|---|---|
| 32 bits | 32.7% | 37.2% |
| 64 bits | 23.9% | 27.2% |
| 128 bits | 19.1% | 21.8% |
| 256 bits | 16.8% | 19.2% |

(c) Main-memory access latency = 40 bus cycles.

### *3.2. Figuring Out Optimal Cache Design*

Having learned the significance of good cache design, the next issue is to figure out the optimal choice of cache metrics, that is, cache size, degree of set associativity, and block size, for various system configurations. Based on the general perception, we may anticipate that the amount of bus traffic would decrease as cache size and degree of set associativity increase. This perception is confirmed by the data collected in the simulation runs. However, the data also show that the decrease of bus traffic achieved by increasing cache size and degree of set associativity gets saturated beyond certain points. If we vary cache size and degree of

| Set Associativity | Cache Size in Bytes | | | |
|---|---|---|---|---|
| | 1M | 2M | 4M | 8M |
| 1-way | | | | |
| 2-way | | | | |
| 4-way | | | | |
| 8-way | | | | |

Figure 2. The region where the reducing of bus traffic due to larger cache size and higher degree of set associativity gets saturated.

set associativity and use 10 percent as the criterion of saturation, then, the shaded region in Figure 2 marks the combinations of cache size and degree of set associativity that consistently yield less than 10 percent more traffic than the selection of 8 megabytes with 8-way set associativity, which is the largest cache size with the highest degree of set associativity in our simulation runs. It is interesting to observe that the same saturation region occurs no matter what the combination of block size, number of CPUs, bus width, and main-memory latency is. This result is significant since it implies that caches with the following two combinations of cache size and degree of set associativity are in general the most cost-effective choices with respect to minimizing bus traffic:

- 2 megabyte with 4-way set associativity
- 4 megabyte with 2-way set associativity.

Once the cache size and degree of set associativity are fixed, the remaining cache metric yet to be determined is block size. An important observation from the simulation output data is that the optimal choice of block size is virtually independent of CPU number, cache size, degree of set associativity, and main-memory latency as long as the 2-tuple of cache size and degree of set associativity is in the saturation region as shown in Figure 2. The optimal block size is, however, a function of bus width. Figure 3 plots the amount of bus traffic versus block size in systems that have 8 CPUs, 2-megabyte 4-way set-associative caches, and a main memory with a latency of 20 clock cycles. For systems with a wider bus, the optimal choice of block size tends to be larger. This effect is expected because the wider the bus is the less the time needed to transfer a block of data over the bus. Table 2 summarizes the optimal choice of block size for systems with a

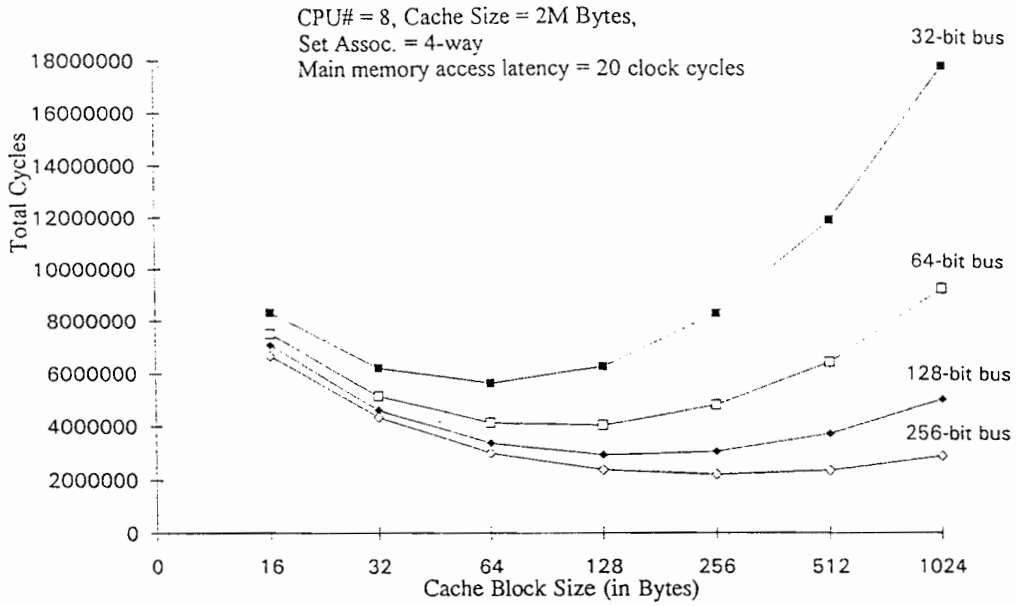



Figure 3. Bus traffic versus block size in systems that have 8 CPUs, 2-megabyte 4-way set-associative caches, and a main memory with a latency of 20 clock cycles.

combination of cache size and degree of set associativity that falls in the shaded area of Figure 2.

In the preceding discussion, we did not take into account the cost and system performance impacts of employing a larger cache or a higher degree of set associativity. One of the reasons is that it is impossible to derive a general cost/effectiveness tradeoff since costs of hardware components depreciate all the time. What we can say here is that whenever we can use a less complicated design, for example, smaller caches or lower degree of set associativity, and still achieve the same effect, we should adopt it. This is what the saturation region shown in Figure 2 is about. As far as system performance is concerned, this issue is of less significance since our discussion is about the second-level caches rather than the first-level caches. In contemporary computer systems, the first-level caches must match the CPU speeds to deliver a good system performance. The speed of the second-level caches does not play an important role in determining the overall system performance as long as the second-level caches are not unreasonably slow.

Table 3. Optimal choice of block size for systems with optimal choice of cache size and degree of set associativity.

| Bus width | 32-bit | 64-bit | 128-bit | 256-bit |
|---|---|---|---|---|
| Optimal block size | 64 bytes | 128 bytes | 128 bytes | 256 bytes |

## 4. *The Effect of Implementing Process Affinity*

In Section 3, we discussed optimal cache design for minimizing bus traffic under an unrestricted process-migration policy. Under this policy, a process in the global ready queue will be scheduled to run at any available CPU regardless of the process's past trace of CPU visit. Sometimes, due to other considerations, one may want to implement process affinity. With the process-affinity policy, the system maintains one ready queue for each CPU. A process is assigned to a CPU when it is created, and the process will only run on the assigned CPU during its life span. In this section, we will discuss optimization of second-level megabyte caches under the process-affinity policy.

In the simulation, the 15 traces were evenly distributed to the CPUs. The first phenomenon one would observe from the simulation output data is that the amount of bus traffic is significantly reduced with process affinity. The percentage of reduction ranges from 40 percent to 70 percent depending on system configuration. This effect is expected due to the following reasons:

1. Process migration induces more compulsory cache misses [Hennessy and Patterson 1990] as a process migrates to a CPU that the process has never visited.
2. Process migration means that $N$ processes contend for space in each CPU's cache, instead of $N/K$ processes, where $K$ is number of CPUs. Thus, process migration increases the interprocess interference [Sites and Agarwal 1988].
3. Process migration also means that a large number of read-only blocks end up with multiple copies in multiple caches, effectively reducing the total system cache size [Sites and Agarwal 1988].

As far as optimization of cache design is concerned, an interesting and important observation is that the same saturation region as plotted in Figure 2 also applies here. Thus, optimal choice of cache size and degree of set associativity regarding minimization of bus traffic is not affected by the implementation of process affinity. Hence, the remaining cache metric that needs to be determined is

cache block size. With process affinity, the optimal block size tends to be larger. The increase of optimal block size when process affinity is implemented is due to a reduction of data sharing among caches. Because process affinity causes no data sharing induced by process migration, the effect of false sharing is reduced. Since false sharing is a negative factor for employing large cache blocks, the reduction of false sharing means that the trade-off is inclined to having larger blocks.

## 5. *Conclusion*

This paper presents a comprehensive study on optimal design of megabyte second-level caches for minimizing bus traffic in shared-memory shared-bus multiprocessors. It focuses on the common multiple-user/multiple-task workload, since the majority of multiprocessors are installed for improving system throughput under this kind of environment. The results from the simulation runs are significant. They show that a careful design could mean a reduction of bus traffic by more than 80 percent or, equivalently, an increase of system scalability by more than five times. Furthermore, according to the simulation results, optimal choice of cache configuration metrics are invariant under several system configurations. This leads to the development of a few simple design guidelines, the most important of which are the following:

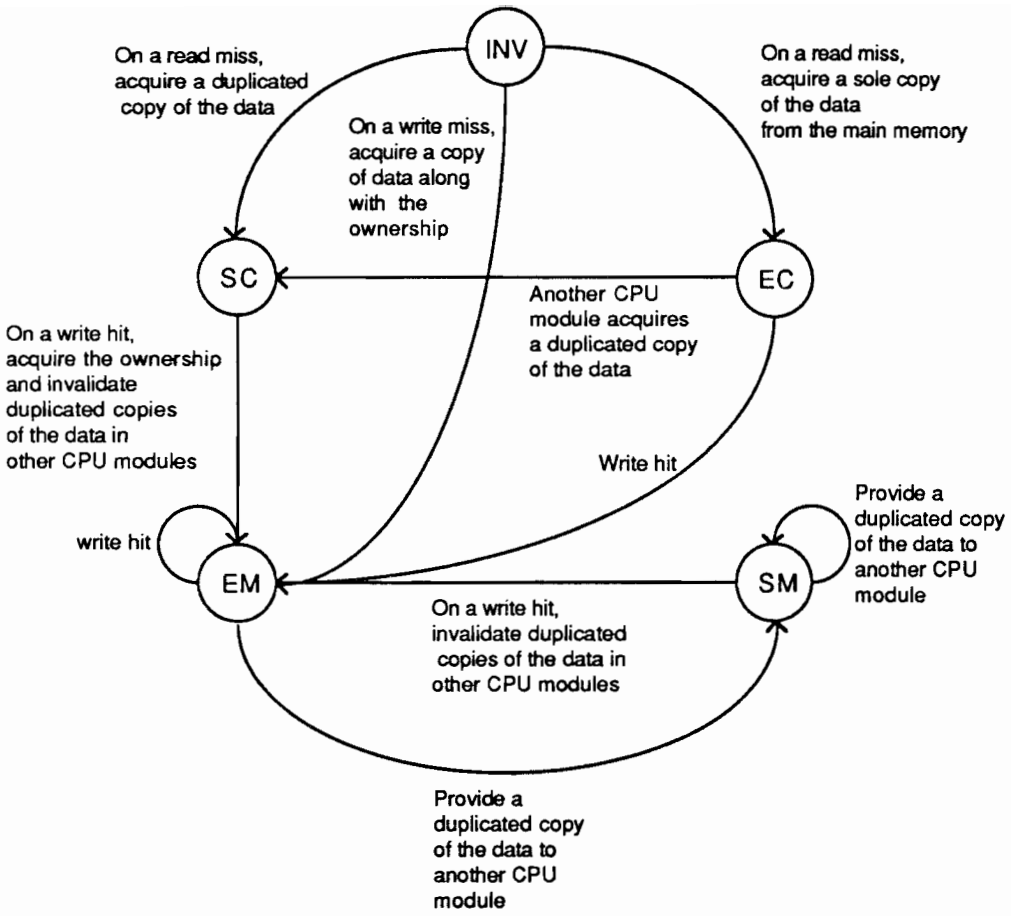
1. A good design of megabyte second-level caches could mean up to 80 percent reduction in bus traffic.
2. The following two combinations of cache size and degree of set associativity are in general the most cost-effective choices for minimizing bus traffic: 2 megabyte with 4-way set associativity or 4 megabyte with 2-way set associativity.
3. The optimal choice of block size is virtually independent of CPU number, cache size, degree of set associativity, and main-memory latency as long as a combination of cache size and degree of set associativity that minimizes bus traffic, that is, in the saturation region as shown in Figure 2, is selected. The optimal choice of block size, however, is a function of bus width and also depends on whether process affinity is implemented. With process affinity implemented, the optimal choice of block size tends to be larger due to less false-sharing effect.

## *Appendix A*

The M Bus cache coherence protocol [Cypress 1991] is based on MOESI cache states proposed in [Sweazey and Smith 1986]. The M Bus protocol employs the write-back and write-invalidate policies. The key of the M Bus protocol is the ownership mechanism [Katz et al. 1985]. The possession of the ownership of a block of data by a CPU module means that the CPU module has the exclusive right to update the block of data and the responsibility to provide a duplicated copy of the data to another CPU module if requested. Multiple copies of a block of data can co-exist in more than one CPU module but only one CPU module can be the owner at one time since the right to update is exclusive. The owner of a block of data is also responsible for writing back the data to the main memory when the data is to be expelled from its cache memory. CPU modules that cache duplicated copies of data of which they are not the owner do not need to perform the write-back operation. The M Bus protocol has five cache states, namely,

1. Invalid: The cache block contains no valid data.
2. Exclusive Clean: The cache block contains the only copy of the data among all CPU modules. The copy is clean but the CPU module is not the owner of the data.
3. Shared Clean: The cache block contains one of the multiple copies of the data among all CPU modules. The copy is clean but the CPU module is not the owner of the data.
4. Exclusive Modified: The cache block contains the only copy of the data among all CPU modules. The copy is dirty (modified), and the CPU module is the owner of the data.
5. Shared Modified: The cache block contains one of the multiple copies of the data. The copy is dirty (modified), and the CPU module is the owner of the data.

Figure 4 shows the state transitions in the M Bus protocol. Not included in Figure 4 are edges from each of the valid states to the invalid state in response to an invalidation request.




INV: Invalid
SC: Shared Clean
EC: Exclusive Clean
SM: Shared Modified
EM: Exclusive Modified

Figure 4. State transitions in the M Bus protocol.

## *Appendix B*

The characteristics of the 15 SPARCSim traces used in the simulation are presented in Table 4, where

- CS is a cache simulator.
- SC is a superscalar compiler.
- PSC is a preprocessor of a superscalar compiler.
- Othello is a chess game.
- Zip is a file-compression program.
- Compress is a file-compression program.
- Diff is a UNIX utility that compares two files and lists their difference.
- Find is a UNIX utility that searches files in a directory.
- GO is a Chinese chess program.
- Less is an enhanced UNIX utility of More.
- Indent is a formatter.
- CPP is a preprocessor of a C compiler.
- Bison is a Yacc-like package.
- GAS is the GNU SPARC assembler.
- Ispell is a UNIX utility that checks spelling errors.

Table 4. Characteristics of the traces used in the simulation.

| SPARCSim traces | Total instructions executed | Supervisor instructions executed | User instructions executed |
|---|---|---|---|
| CS | 4770171 | 691 | 4769480 |
| SC | 6728246 | 3320 | 6724926 |
| SPC | 5208786 | 124498 | 5084288 |
| Othello | 5117904 | 275503 | 4842401 |
| Zip | 2667238 | 359 | 2666879 |
| Compress | 4414012 | 729 | 4413283 |
| Diff | 3385299 | 1139 | 3384160 |
| Find | 4610356 | 9867 | 4600489 |
| Go | 4820332 | 2186 | 4818146 |
| Less | 3496604 | 3244 | 3493360 |
| Indent | 3812538 | 1909 | 3810629 |
| CPP | 5714003 | 7091 | 5166912 |
| Bison | 4700577 | 28107 | 4672470 |
| Gas | 4845604 | 1059 | 4844545 |
| Ispell | 5278586 | 5025 | 5273561 |

## *References*


1. J. Archibald and J.-L. Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. ACM Trans. on Computer Systems 4(4):273–298, November 1986.
2. H. O. Bugge, E. H. Kristiansen, and B. O. Bakka. Trace-Driven Simulations for a Two-Level Cache Design in Open Bus Systems. Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 250–259, 1990.
3. Cypress Semiconductor. *Sparc MBus Interface Specification.* Cypress Semiconductor, January, 1991.
4. S. J. Eggers and R. H. Katz. Evaluating the Performance of Four Snooping Cache Coherency Protocols. Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 2–15, 1989.
5. J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* San Mateo, Calif.: Morgan Kaufmann Publishers, 1990.
6. A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive Snoopy Caching. *Algorithmica* vol. 3 pp. 79–119, 1988.
7. R. Katz, S. Eggers, D. Wood, C. L. Perkins, and R. Sheldon. Implementing a Cache Consistency Protocol. Proceedings of the 12th Annual International Symposium on Computer Architecture, pp. 276–283, 1985.
8. S. Przybylski. *Cache and Memory Hierarchy Design.* San Mateo, Calif.: Morgan Kaufmann, 1990.
9. R. Sites and A. Agarwal. Multiprocessor Cache Analysis Using ATUM. Proceedings of the 15th Annual International Symposium on Computer Architecture, pp. 186–195, 1988.
10. Sun Microsystems. *SPARCSim Manual Set.* Sun Microsystems, 1989.
11. P. Sweazey and A. J. Smith. A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus. Proceedings of the 13th Annual International Symposium, pp. 414–423, 1986.