A LOW-OVERHEAD COHERENCE SOLUTION FOR MULTIPROCESSORS WITH PRIVATE CACHE MEMORIES

Mark S. Papamarcos and Janak H. Patel
Coordinated Science Laboratory
University of Illinois
1101 W. Springfield
Urbana, IL 61801

ABSTRACT

This paper presents a cache coherence solution for multiprocessors organized around a single time-shared bus. The solution aims at reducing bus traffic and hence bus wait time. This in turn increases the overall processor utilization. Unlike most traditional high-performance coherence solutions, this solution does not use any global tables. Furthermore, this coherence scheme is modular and easily extensible, requiring no modification of cache modules to add more processors to a system. The performance of this scheme is evaluated by using an approximate analysis method. It is shown that the performance of this scheme is closely tied with the miss ratio and the amount of sharing between processors.

I. INTRODUCTION

The use of cache memory has long been recognized as a cost-effective means of increasing the performance of uniprocessor systems [Conti69, Meade70, Kaplan73, Strecker76, Rao78, Smith82]. In this paper, we will consider the application of cache memory in a tightly-coupled multiprocessor system organized around a timeshared bus. Many computer systems, particularly the ones which use microprocessors, are heavily bus-limited. Without some type of local memory, it is physically impossible to gain a significant performance advantage through multiple microprocessors on a single bus.

Generally, there are two different implementations of multiprocessor cache systems. One involves a single shared cache for all processors [Yeh83]. This organization has some distinct advantages, in particular, efficient cache utilization. However, this organization requires a crossbar between the processors and the shared cache. It is impractical to provide communication between each processor and the shared cache using a shared bus. The other alternative is private cache for each processor, as shown in Fig. 1. However, this organization suffers from the well known *data consistency* or *cache coherence* problem. Should the same writeable data block exist in more than one cache, it is possible for one processor to modify its local copy independently of the rest of the system.

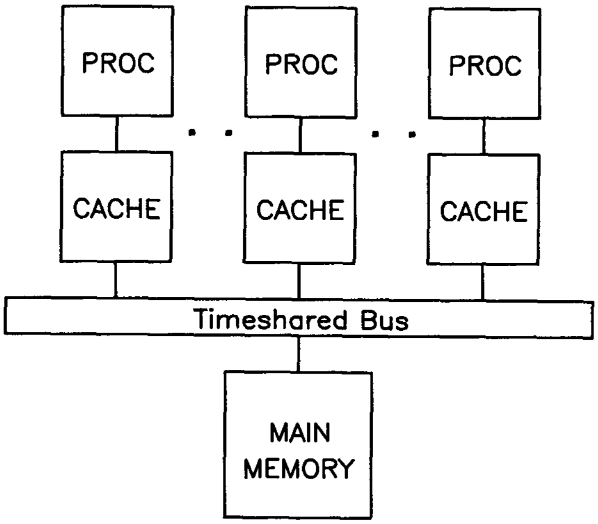

Fig. 1 System Organization

The simplest way to solve the coherence problem is to require that the address of the block being written in cache be transmitted throughout the system. Each cache must then check its own directory and purge the block if present. This scheme is most frequently referred to as *broadcast-invalidate*. Obviously, the invalidate traffic grows very quickly and, assuming that writes constitute 25% of the memory references, the system becomes saturated with less than four processors. In [Bean79], a *bias filter* is proposed to reduce the cache directory interference that results from this scheme. The filter consists of a small associative memory between the bus and each cache. The associative memory keeps a record of the most recently invalidated blocks, inhibiting some subsequent wasteful invalidations. However, this only serves to reduce the amount of cache directory interference without actually reducing the bus traffic.

Another class of coherence solutions are of the *global-directory* type. Status bits are associated with each block in main memory. Upon a cache miss or the first write to a block in cache, the block's global status is checked. An invalidate signal is sent only if another cache has a

ACKNOWLEDGEMENTS: This research was supported by the Naval Electronics Systems Command under VHSIC contract N00039-80-C-0556 and by the Joint Services Electronics Program under contract N00014-84-C-0149.

0194-7111/84/0000/0348$01.00©1984 IEEE

copy. Requests for transfers due to misses are also screened by the global table to eliminate unnecessary cache directory interference. The performance associated with these solutions is very high if one ignores the interference in the global directory. The hardware required to implement a global directory for low access interference is extensive, requiring a distributed directory with full crossbar. These schemes and their variations have been analyzed by several authors [Tang76,Censier78,Dubois82,Yen82,Archibald83].

A solution more appropriate for bus organized multiprocessors has been proposed by Goodman [Goodman83]. In this scheme, an invalidate request is broadcast only when a block is written in cache for the first time. The updated block is simultaneously written through to main memory. Only if a block in cache is written to more than once is it necessary to write it back before replacing it. This particular write strategy, a combination of write-through and write-back, is called write-once. A dual cache directory system is employed in order to reduce cache interference.

We seek to integrate the high performance of global directory solutions associated with the inhibition of all ineffective invalidations and the modularity and easy adaptability to microprocessors of Goodman's scheme. In a bus-organized system with dual directories for interrogation, it is possible to determine at miss time if a block is resident in another cache. Therefore a status may be kept for each block in cache indicating whether it is Exclusive or Shared. All unnecessary invalidate requests can be cut off at the point of origin. Bus traffic is therefore reduced to cache misses, actual invalidations and writes to main memory. Of these, the traffic generated by cache misses and actual invalidations represents the minimum unavoidable traffic. The number of writes to main memory is determined by the particular policy of write-through or write-back. Therefore, for a multiprocessor on a timeshared bus, performance should then approach the maximum possible for a cache coherent system under the given write policy.

The cache coherence solution to be presented is applicable to both write-through and write-back policies. However, it has been shown that write-back generates less bus traffic than write-through [Norton82]. This has been verified by our performance studies. Therefore, we have chosen a write-back policy in the rest of this paper. Under a write-back policy, coherence is not maintained between a cache and a main memory as can be done with a write-through policy. This in turn implies that I/O processors must follow the same protocol as a cache for data transfer to and from memory.

II. PROPOSED COHERENCE SOLUTION

In this section we present a low-overhead cache coherence algorithm. To implement this algorithm, it is necessary to associate two status bits with each block in cache. No status bits are associated with the main memory. The first bit indicates either Shared or Exclusive ownership of a block, while the second bit is set if the block has been locally modified. Because the state Shared-Modified is not allowed in our scheme, this status is used instead to denote a block containing invalid data. A write-back policy is assumed. The four possible statuses of a block in cache at any given time are then:

1. Invalid: Block does not contain valid data.
2. Exclusive-Unmodified (Excl-Unmod): No other cache has this block. Data in block is consistent with main memory.
3. Shared-Unmodified (Shared-Unmod): Some other caches may have this block. Data in block is consistent with main memory.
4. Exclusive-Modified (Excl-Mod): No other cache has this block. Data in block has been locally modified and is therefore inconsistent with main memory.

A block is written back to main memory when evicted only if its status is Excl-Mod. If a write-through cache was desired then one would not need to differentiate between Excl-Mod and Excl-Unmod. Writes to an Exclusive block result only in modification of the cached block and the setting of the Modified status. The status of Shared-Unmod says that some other caches may have this block. Initially, when a block is declared Shared-Unmod, at least two caches must have this block. However, at a later time when all but one cache evicts this block, it is no longer truly Shared. But the status is not altered in favor of simplicity of implementation.

Detailed flow charts of the proposed coherence algorithm are given in Figs. 2 and 3. Fig. 2 gives the required operations during a read cycle and Fig. 3 describes the write cycle. The following is a summary of the algorithm and some implementation details which are not present in the flow charts.

Upon a cache miss, a read request is broadcast to all caches and the main memory. If the miss was caused by a write operation, an invalidate signal accompanies the request. If a cache directory matches the requested address then it inhibits the main memory from putting data on the bus. Assuming cache operations are asynchronous with each other and the bus, possible multiple cache responses can be resolved with a simple priority network, such as a daisy chain. The highest priority cache among the responding caches will then put the data on the bus. If no cache has the block then the memory provides the block. A unique response is thus guaranteed. On a read operation, all caches which match the requested address set the status of the corresponding block to Shared-Unmod. In addition, the block is written back to main memory concurrently with the transfer if its status was Excl-Mod. On a write, matching caches set the block status to Invalid. The requesting cache sets the status of the block to Shared-Unmod if the block came from another cache and to Excl-Unmod if the block came from

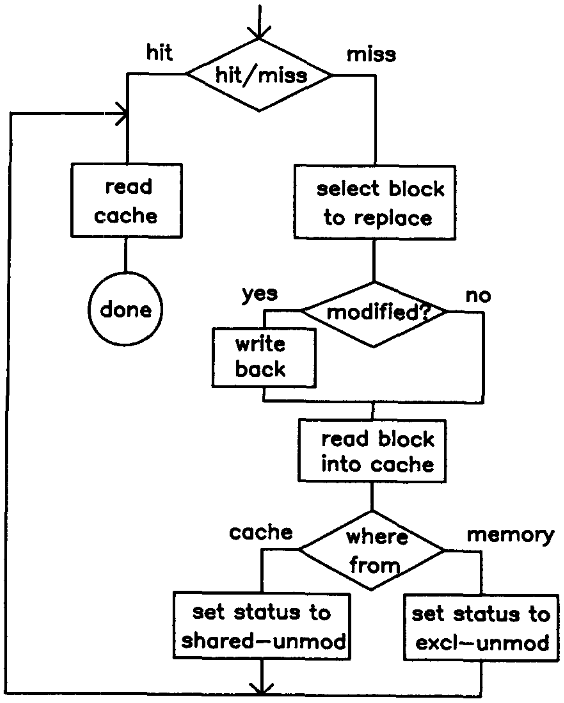

Fig. 2 Cache Read Operation

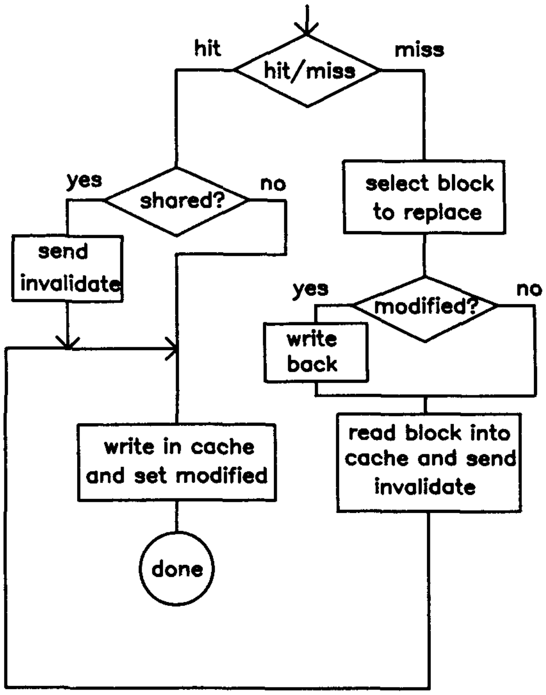

Fig. 3 Cache Write Operation

main memory. Upon a subsequent cache write, an invalidation signal is broadcast with the block address only if the status is Shared-Unmod, thus minimizing unnecessary invalidation traffic.

As will be seen in the following sections, the performance of the proposed coherence algorithm is directly dependent on the miss ratio and the degree of sharing, while in algorithms not utilizing global tables the performance is tied closely with the write frequency. Since the number of cache misses are far fewer than the number of writes, intuitively it is clear that the proposed algorithm should perform better than other modular algorithms.

Most multiprocessing systems require the use of synchronization and mutual exclusion primitives. These primitives can be implemented with indivisible read-modify-write operations (e.g., test-and-set) to memory. Indivisible read-modify-write operations are a challenge to most cache coherence solutions. However, in our system, the bus provides a convenient "lock" operation with which to solve the read-modify-write problem. In our scheme if the block is either Excl-Unmod or Excl-Mod no special action is required to perform an indivisible read-modify-write operation on that block. However, if the block is declared Shared-Unmod, we must account for the contingency in which two processors are simultaneously accessing a Shared block. If the operation being performed is designated as indivisible, then the cache controllers must first capture the bus before proceeding to execute the instruction. Through the normal bus arbitration mechanism, only one cache controller will get the bus. This controller can then complete the indivisible operation. In the process, of course, the other block is invalidated and the other processor treats the access as a cache miss and proceeds on that basis. An implicit assumption in this scheme is that the controller must know <u>before</u> it starts executing the instruction that it is an indivisible operation. Some current microprocessors are capable of locking the bus for the duration of an instruction. Unfortunately, with some others it is not possible to recognize a read-modify-write before the read is complete; it is then too late to backtrack. For specific processors we have devised elaborate methods using interrupts and system calls to handle such situations. We will not present the specifics here, but it suffices to say that the schemes involve either the aborting and retrying of instructions or decoding instructions in the cache controller.

III. PERFORMANCE ANALYSIS

The analysis of this coherence solution stems from an approximate method proposed by Patel [Patel82]. In this method, a request for a block transfer is broken up into several unit requests for service. The waiting time is also treated as a series of unit requests. Furthermore, these unit requests are treated as independent and random requests to the bus. It was shown in that paper that this rather non-obvious transformation of the problem results in a much simpler but fairly accurate analysis. The errors introduced by the approximation are less than 5% for a low miss ratio. First, let us define the system parameters:

N	number of processors
a	processor memory reference rate
m	miss ratio

w fraction of memory references that are writes

d probability that a block in cache has been locally modified before eviction, i.e., the block is "dirty"

u fraction of write requests that reference Unmodified blocks in cache

s fraction of write requests that reference Shared blocks, equivalent to the fraction of Shared blocks in cache if references are assumed to be equally distributed throughout cache

A number of cycles required for bus arbitration logic

T number of cycles required for a block transfer

I number of cycles required for a block invalidate

To analyze our cache system, consider an interval of time comprising k units of useful processor activity. In that time, kb bus requests will be issued, where

$$b = ma + (1-m)awsu$$

The term ma in the above expression represents the bus accesses due to cache misses and the term (1-m)awsu accounts for the invalidate requests resulting from writes to Shared-Unmod blocks.

The actual execution time for 1 useful unit of work, disregarding cache interference, will be

$$1 + bA + maT + madT + (1-m)awsuI + bW$$

where W is the average waiting time per bus request. The cpu idle times per useful cpu cycle are the factors bA for bus arbitration, maT for fetching blocks on misses, madT for write-back of Modified blocks, (1-m)awsuI for invalidate cycles, and bW for waiting time to acquire the bus.

Now we account for cache interference from other processors. If no dual cache directory is assumed the performance degradation due to cache interference can be extremely severe. Therefore, we have assumed dual directories in cache. In this case, the cache interference will occur only in the following situations:

1. A given processor receives invalidate requests from (N-1) other processors at the rate of (N-1)(1-m)awsu. We assume that all invalidates are effective and that, on the average, one cache is invalidated. The penalty for an invalidate is assumed to be one cache cycle.
2. Transfer requests occur at the rate (N-1)ma, of which (N-1)mas are for Shared blocks. We again assume that, on the average, one cache responds to the request. The penalty for a transfer is T cycles.

We define Q to be the sum of these two effects, namely

$$Q = (1 - m)awsu + masT$$

Cache interference is assumed to be distributed over the processor execution time, yielding

$$Z = 1 + bA + maT + madT + (1-m)awsuI + bW + \frac{Q}{Z^2} \quad (1)$$

where Z is the real execution time for 1 useful unit of work. The unit request rate for each of the N processors as seen by the bus is

$$\frac{Z - 1 - bA - Q/Z^2}{Z}$$

The probability that no processor is requesting the bus is given by

$$\left(1 - \frac{Z - 1 - bA - Q/Z^2}{Z}\right)^N$$

Therefore, the probability that at least one processor is requesting the bus, that is, the average bus utilization B, is,

$$B = 1 - \left(1 - \frac{Z - 1 - bA - Q/Z^2}{Z}\right)^N \quad (2)$$

To solve for B, W and Z, we need one more expression for the bus utilization. That can be obtained by multiplying N by the actual bus time used, averaged over the execution period, giving

$$B = \frac{N\ (Z - 1 - bA - bW - Q/Z^2)}{Z} \quad (3)$$

Now we can solve for B, W and Z using equations (1), (2) and (3). Similar derivations exist for the case of no coherence, no coherence and no bus contention (infinite crossbar), and Goodman's scheme. The processor utilization U is simply 1/Z.

IV. DISCUSSION OF RESULTS

In this section we present the analytical results to demonstrate the effect of various parameters on the cpu performance and bus traffic. The values of cache parameters used span a reasonable range covering most practical situations. In some cases we have chosen pessimistic values to emphasize the fact that our cache coherence solution still gives good performance. The following values were used as default cache parameters:

m = 5% Miss ratio: It may actually be lower for reasonable cache sizes, so this is a pessimistic assumption. Lower miss ratios would be appropriate for single-tasking processors, while the 7.5% figure may be appropriate for multi-tasking environments involving many context switches.

a = 90% Processor to memory access rate: Here we assume 90% of cpu cycles result in a cache request, although a smaller fraction is more likely in processors with a large register set.

d = 50% Write-back probability: Assume here that approximately half of all blocks are locally modified before eviction, although 20% and 80% are tried in order to see the effect of this parameter.

w = 20% Write frequency: Assumed to be about 20% of all memory references. This is a fairly standard number. Since it only appears as a factor in the generation of invalidate requests with u and s, its actual value is not critical.

u = 30% Fraction of writes to unmodified blocks: Assume that roughly one third of all write hits are first-time writes to a given unmodified block and the remainder are subsequent writes to the modified block.

s = 5% Degree of sharing: In most cases we have assumed that 5% of writes are to a block which is declared Shared-Unmod. This should be a pessimistic assumption except for programs which pass large amounts of data between processors in which case s = 15% is more reasonable. In systems where most sharing occurs only on semaphores, the 1% figure is more likely.

A = 1 Bus arbitration time: Assume that the logic for determining the next bus master settles within one cache cycle.

T = 2 Block transfer time: In a microprocessor environment blocks are likely to be small. Therefore, in most cases we have assumed that it takes approximately two cache cycles to transfer a block to a cache. We have also considered the effect of varying block transfer times due to differing technologies or larger cache blocks.

I = 2 Block invalidate time: We have assumed that the time taken for an invalidate cycle should be only slightly longer than a normal cache cycle, since the invalidate operation consists only of transmitting an address and modifying the affected cache directories.

The analytical method was verified using a time-driven simulator of the performance model. In all cases tested, the predicted performance differed by no more than 5% from the simulated performance. This error tended to approach 0 with heavier bus loading. Because of the comparative ease of generating data using the analytical solution, all results shown have been derived analytically. On each graph, all parameters assume their default values except the one being varied.

Figs. 4 through 6 illustrate the effects of different miss ratios on bus utilization, system performance, and processor utilization as function of the number of processors. System performance is expressed as NU, where N is the number of processors and U is the single processor utilization. The system performance is limited primarily by the bus. From Fig. 4 we see that for 7.5% miss ratio the bus saturates with about 8 processors. As the miss ratio decreases to 2.5% the bus saturates with about 18 processors. The effect of bus saturation on system performance can be seen in Fig. 5. Note that, in general, bus utilization and system performance increase almost linearly with N until the bus reaches saturation. At this point, processor utilization begins to approach a curve proportional to 1/N as seen in Fig. 6. If a 1% miss ratio could be achieved, performance would top out with NU=29.

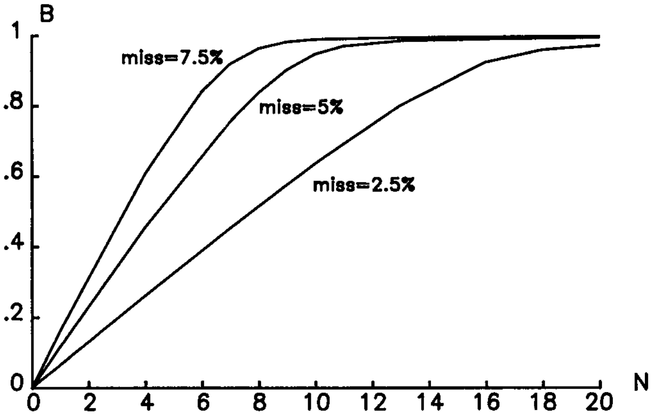

Fig. 4 Effect of Miss Ratio m:
Bus Utilization vs. Number of Processors

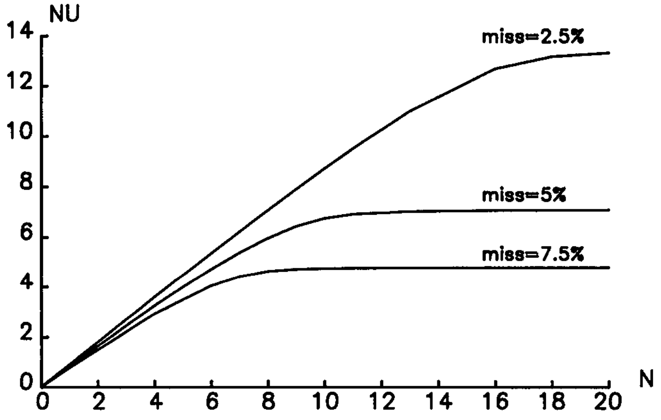

Fig. 5 Effect of Miss Ratio m:
System Performance vs. Number of Processors

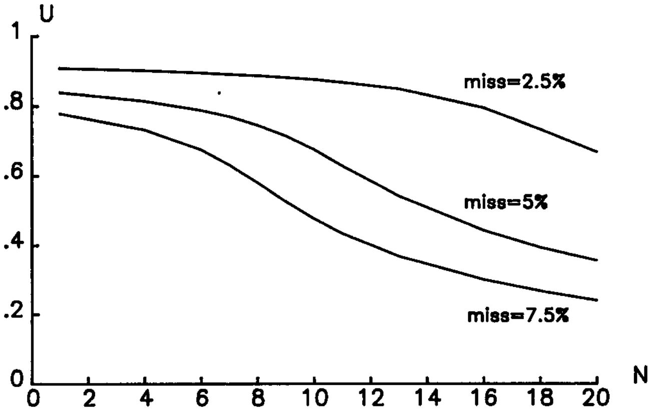

Fig. 6. Effect of Miss Ratio m:
Processor Utilization vs. No. of Processors

Figs. 7 and 8 illustrate the effect of differing degrees of interprocessor sharing. The effect of sharing factor, s, on system performance is relatively small compared with the effect of miss ratio. It is the factor (1-m)awsu that is responsible for the generation of invalidation traffic, which is generally smaller than the miss traffic. These graphs are also demonstrative of the effects of variations in the write frequency (w) and the percentage of first writes (u). The value of w is relatively fixed between 20% and 30%, and u should be fairly constant as well, except when moving large quantities of data. The s = 100% case corresponds to a standard write-back coherence scheme in which any block is potentially sharable. With a write-back frequency of 30% instead of 50% to compensate for initial write-throughs, the curves for Goodman's scheme are almost identical to those for s = 100%.

Fig. 9 illustrates the effect of different write-back frequencies. The results here are fairly predictable. Write-back is yet another factor which contributes to the bus traffic. A write-through policy would contribute much more traffic than this.

Fig. 10 illustrates the degradation due to increasing block transfer times. System performance is so limited by transfer times of 4 cycles or more that it is absolutely necessary to be able to bring a block into cache in one or two cycles.

Finally, Fig. 11 shows that the proposed coherence solution is very close to the ideal achievable system performance for a timeshared bus. The top curve represents a system not constrained by a bus, while the second corresponds to a system with no coherence overhead. The bottom curve, representing the proposed solution, is very close to the middle curve, clearly showing that little system performance is lost in maintaining cache consistency using our algorithm.

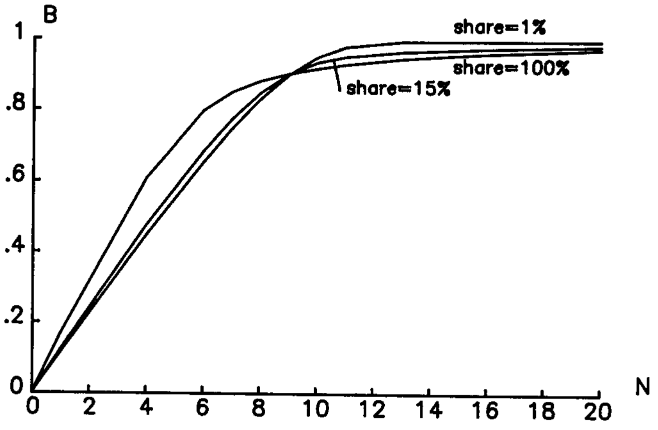

Fig. 7. Effect of Degree of Sharing s:
Bus Utilization vs. Number of Processors

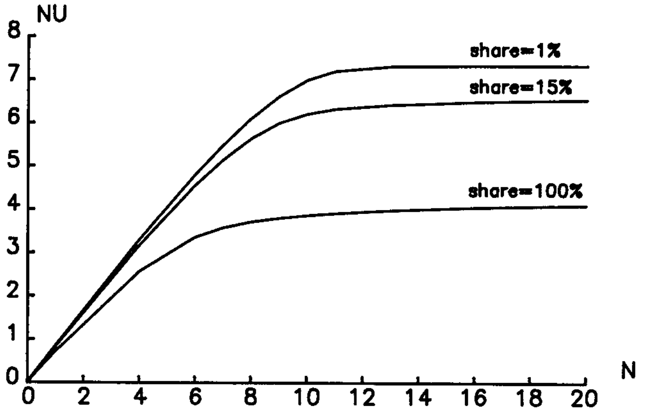

Fig. 8. Effect of Degree of Sharing s:
System Performance vs. Number of Processors

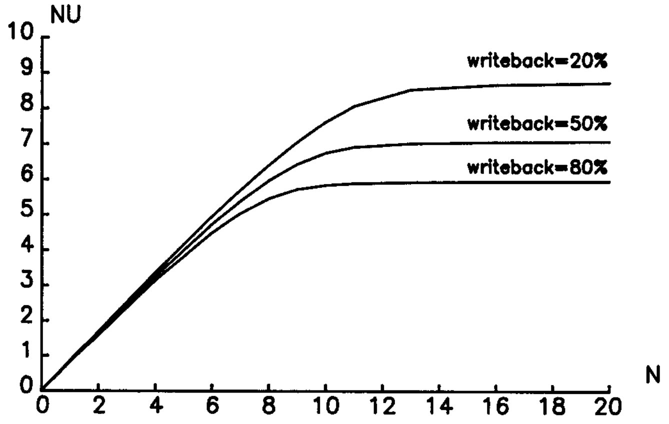

Fig. 9. Effect of Write-Back Probability d:
System Performance vs. Number of Processors

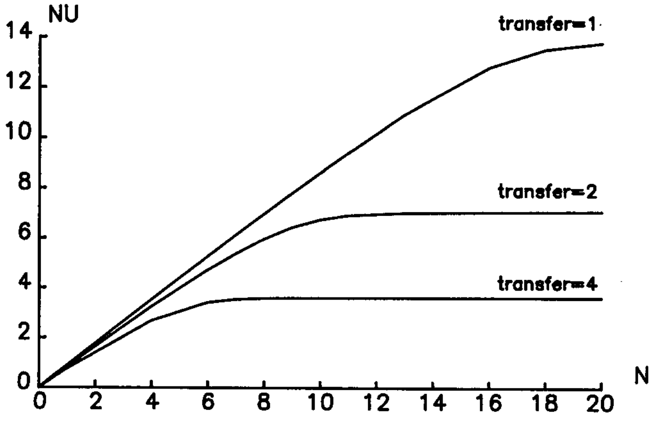

Fig. 10. Effect of Block Transfer Time T:
System Performance vs. No. of Processors

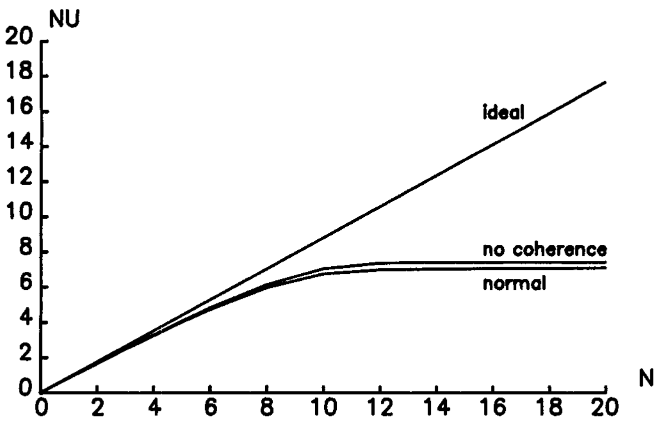

Fig. 11. Overhead of Coherence Solution: System Performance vs. No. of Processors

V. CONCLUDING REMARKS

In this paper we have introduced a new coherence algorithm for multiprocessors on a timeshared bus. It takes advantage of the relatively small amount of data shared between processors without the need for a global table. In addition, it is easily extensible to an arbitrary number of processors and relatively uncomplicated. The applications of a system of this type are many. Processing modules could be added as needed, and the system need not be redesigned for each new application. For example, an interesting application would be to allocate one processing module to each user, with one possibly dedicated to operating system functions. Again, the primary advantage is easy expandability and very little performance degradation as a result of it. For any multiprocessor system on a timeshared bus, this coherence solution is as easy to implement as any other, save broadcast-invalidate, and offers a significant performance improvement if the amount of shared data memory is reasonably small.

VI. ACKNOWLEDGEMENTS

We would like to thank Professor Faye Briggs of Rice University and Professor Jean-Loup Baer of the University of Washington for helpful discussions concerning this paper.

REFERENCES

[Archibald83]
J. Archibald and J. L. Baer, "An Economical Solution to the Cache Coherence Solution," University of Washington Technical Report 83-10-07, October, 1983.

[Bean79]
B. M. Bean, K. Langston, R. Partridge, and K. K. Sy, "Bias Filter Memory for Filtering out Unnecessary Interrogations of Cache Directories in a Multiprocessor System," United States Patent 4,142,234, February 27, 1979.

[Censier78]
L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," IEEE Trans. Comput., vol. C-27, December 1978, pp. 1112-1118.

[Conti69]
C. J. Conti, "Concepts for Buffer Storage," IEEE Comput. Group News, vol. 2, March 1969, pp. 9-13.

[Dubois82]
M. Dubois and F. A. Briggs, "Effects of Cache Coherency in Multiprocessors," IEEE Trans. Comput., vol. C-31, November 1982, pp. 1083-1099.

[Goodman83]
J. R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," Proc. 10th Annual Symp. on Computer Architecture, June 1983, pp. 124-131.

[Kaplan73]
K. R. Kaplan and R. O. Winder, "Cache-Based Computer Systems," Computer, March 1973, pp. 30-36.

[Meade70]
R. M. Meade, "On Memory System Design," AFIPS Proc. FJCC, vol. 37, 1970, pp. 33-43.

[Norton82]
R. L. Norton and J. A. Abraham, "Using Write Back Cache to Improve Performance of Multiuser Multiprocessors," Proc. 1982 Int. Conf. on Parallel Processing, August 1982, pp. 326-331.

[Patel82]
J. H. Patel, "Analysis of Multiprocessors with Private Cache Memories," IEEE Trans. Comput., vol. C-31, April 1982, pp. 296-304.

[Rao78]
G. S. Rao, "Performance Analysis of Cache Memories," J. ACM, vol. 25, No. 3, July 1978, pp. 378-395.

[Smith82]
A. J. Smith, "Cache Memories," Computing Surveys, vol. 14, No. 3, September 1982, pp. 473-530.

[Strecker76]
W. D. Strecker, "Cache Memories for PDP-11 Family Computers". Proc. 3rd Annual Symp. on Computer Architecture, January 1976, pp. 155-158.

[Tang76]
C. K. Tang, "Cache System Design in the Tightly Coupled Multiprocessor System," AFIPS Proc. NCC, vol. 45, 1976, pp. 749-753.

[Yeh83]
P. C. C. Yeh, J. H. Patel, and E. S. Davidson, "Shared Cache for Multiple-Stream Computer Systems," IEEE Trans. Comput., vol. C-32, January 1983, pp. 38-47.

[Yen82]
W. C. Yen and K. S. Fu, "Coherence Problem in a Multicache System," Proc. 1982 Int. Conf. on Parallel Processing, 1982, pp. 332-339.