A Superassociative Tagged Cache Coherence Directory

David J. Lilja
Department of Electrical Engineering
Shanthi Ambalavanan
Department of Computer Science
University of Minnesota, Minneapolis

Abstract

Dynamically tagged directories are memory-efficient mechanisms for maintaining cache coherence in shared-memory multiprocessors. These directories use special-purpose caches of pointers that are subject to two types of overflow: 1) pointer overflow, which limits the maximum sharing of a memory block, and 2) set overflow, which forces the premature invalidation of cached blocks. We propose a superassociative tagged directory that can preserve some of the cached copies of a memory block when a set overflows by allowing multiple address tags in the same set to contain the same address value. Verilog descriptions are used to estimate its implementation cost and timing delay, and a multiprocessor cache simulator is used to evaluate its performance.

1. Introduction

Private data caches have long been recognized as an effective means for reducing the average memory delay in shared-memory multiprocessors. Since each cache can have its own copy of a shared variable, some *coherence* mechanism is required to ensure that when a processor reads a shared variable, it always receives the most current value [9]. One type of coherence mechanism uses directories located in the shared memory to keep track of which processors have cached copies of which blocks [2-4]. The number of entries in these traditional directories is proportional to the size of the main memory because the bits used to point to the processors with cached copies of a block are associated with each block in the memory. Since the data caches are significantly smaller than the main memory, however, most of the memory blocks will not be cached at any given time. As a result, most of these pointer bits will be unused.

The *tagged directories* [5, 8, 10, 12], in contrast, dynamically allocate pointers from a special purpose *pointer cache* to individual memory blocks when the block is moved from the memory to a data cache. Each entry in the pointer cache requires two fields: 1) an address tag to identify the memory block to which the pointer is allocated, and 2) a vector of bits to point to the processors with a cached copy of the block.

This work was supported in part by NSF grant no. CCR-9209458.

A variety of structures have been proposed for tagged directories, differing primarily in the organization of the pointer field, and in the policy used to handle pointer replacements on overflow. The example shown in Figure 1 associates n pointers with each address tag to point to the first n processors that request the block. When more than n processors attempt to share the same block, an *invalidate-on-overflow* policy randomly selects one of the n pointers to the given block (i.e. $P0$ - $P3$) for replacement. The processor to which it points then is sent a message to invalidate its cached copy. Several studies [1, 7, 11, 14] have shown that relatively few processors simultaneously share a block so that n=2 to 4 typically is sufficient.

When more than a active blocks map to the same set in an a-way set associative implementation, one of the tag entries is randomly selected for replacement. Invalidation messages must be sent to all of the processors with a cached copy of the block since the address tag will be changed. Thus, this tagged coherence directory suffers from two types of overflow: 1) *pointer overflow*, which limits the simultaneous sharing of a block to n processors, and 2) *set overflow*, which forces all cached copies of a block to be invalidated. These overflows can directly increase the miss ratios of the data caches by prematurely invalidating active data cache entries.

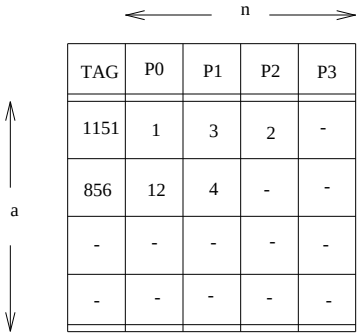

Figure 1: Tagged directory with n=4 pointers per tag, and a=4-way set associativity. Processors 1, 2, and 3 have a cached copy of memory block 1151; processors 4 and 12 have a copy of block 856.

2. A superassociative tagged directory

To preserve some of the processors' cached copies of a memory block when a set overflows due to limited associativity in the cache of pointers, we propose the concept of *superassociativity*. As shown in Figure 2, an a-way superassociative tagged directory allows up to a tags in a set to have the same address value. When it is associatively searched, up to a entries may indicate a match so that up to na processors can simultaneously cache copies of the same block. When a pointer overflow occurs, one of the n processor pointers is chosen at random and the corresponding processor is sent an invalidation command for that block. When a set overflow occurs, one of the a tag entries is chosen at random and invalidation commands are sent to the n corresponding processors. Unlike a standard implementation, however, there still could be up to $(a-1)$ additional tag entries in the set with the same address. These entries will be unaffected by the set overflow allowing up to $n(a-1)$ copies of the block to remain in the processors' data caches. Thus, this superassociative structure provides greater flexibility for block sharing than a standard a-way set associative structure.

We used Verilog models and the Magic design tools to estimate the cost of both a standard tagged directory and a superassociative tagged directory in terms of the transistor count. Hspice was used to estimate the cycle time. The control logic for the superassociative directory is only slightly more complex than the standard directory with equivalent timing delays. The cost of the control logic is relatively low compared to the cost of the tag and pointer entries, and the tag-matching logic. Consequently, we approximate the cost of an a-way superassociative directory as $C = [2\log_2 s + n(18+\log_2 p) + 11m + 23]sa + (18n + 18m + 7)a$ transistors, where there are s sets, n pointers per address tag, p processors, and m address tag bits.

Table 1 compares the characteristics of three directory structures with equal implementation costs. Since the superassociative directories have fewer processor pointers

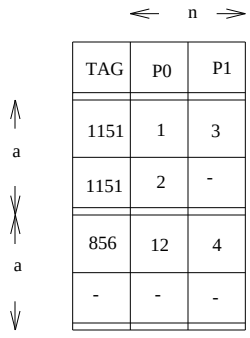

Figure 2: A tagged directory with n=2 pointers per tag and a=2-way *superassociativity.*

Table 1: Tagged directory characteristics with equal transistor counts (p=16; m=32). D_0: standard directory (a=4, n=4); D_1: superassociative directory (a=4, n=1); D_2: superassociative directory (a=2, n=2).

Transistor count ($\times 10^6$)	Total tag entries (d)		
	D_0	D_1	D_2
0.50	1024	1188	1126
8.1	16384	19005	18022
130	262144	304087	288358
	Total processor pointers (nd)		
0.50	4096	1188	2252
8.1	65536	19005	36044
130	1048576	304087	576716

per address tag than the standard directory, the superassociative implementations can have more tag entries for a fixed number of transistors. For the same implementation cost, D_1 has approximately 16 percent more tag entries than D_0, while D_2 has 10 percent more. The penalty for these extra tag entries, and for the superassociative structure itself, is fewer total pointers to processors. Thus, for equivalent costs, the superassociative directories allow less total sharing of memory blocks among the processors than the standard directory, but they do allow much more flexible sharing.

3. Simulation methodology

The simulations use traces generated from several of the Perfect Fortran benchmark programs [6]. The Parafrase-2 [13] source-to-source parallelizing compiler finds all parallel *Doall* loops in each program and inserts calls to a special trace-generation function for each program statement. When the source code is compiled and executed, an interleaved memory trace is generated for all of the processors. Parallel loop iterations are statically assigned to the processors with processor 0 executing all of the sequential sections. A *fork-join* parallelism model is used in which all synchronization is performed by special hardware. Thus, no synchronization variables appear in the memory reference trace. We reduced the loop count of the outermost sequential loop in the test programs to reduce the program traces to reasonable lengths.

The traces drive a shared-memory multiprocessor simulator with p=16 processors connected to p memory modules via a 32-bit packet-switched multistage interconnection network. One tagged directory is located within each of the memory modules. The network traffic generated between the processors and the memory modules for each type of memory transaction is shown in Table 2. To maintain a realistic relationship between the working set size of the test programs and the size of the data caches, a data cache of 4 kbytes is used in each of the processors. The results are relatively insensitive to the

Table 2: Network traffic from memory operations (*b* words per block; 4 bytes per word).

Memory Operation	Forward (bytes)	Reverse (bytes)
READ		
hit	-	-
miss *shared*	8	8+4*b*
miss *exclusive*	16+4*b*	16+4*b*
WRITE		
hit *shared* in *c* caches	8+8*c*	8+8*c*
hit *exclusive*	-	-
miss in memory	8	8+4*b*
miss *shared* in *c* caches	8+8*c*	8+8*c*
miss *exclusive*	16+4*b*	16+4*b*
DIRECTORY OVERFLOW		
invalidate *j* blocks	8*j*	8*j*

actual size of the data cache, however, since the performance of the tagged directories is influenced primarily by the number of pointers per tag, n, and the associativity, a.

The total number of tag entries in the standard directory is varied from d=64 to 1024. The number of tag entries in D_1 is $1.16D_0$, while the number of entries in D_2 is $1.10D_0$. These relationships compare the performance of the three different configurations when implemented with an equal number of transistors and equal cycle times.

4. Performance results

As shown in Figure 3 for a representative subset of the benchmark programs, the D_0 directory tends to produce the largest number of invalidations since each set overflow forces the invalidation of all of the currently cached copies of the memory block allocated to the tag entry that has been selected for replacement. In the D_1 superassociative directory, only one cached memory block will be invalidated when a set overflows. Up to three additional copies of the same block can remain cached in other processors. Similarly, up to two copies of a memory block can remain cached when a set overflows in the D_2 configuration. The network traffic produced by these three equal-cost configurations is shown in Figure 4. In most of the test programs, the D_0 directory tends to produce higher network traffic than the superassociative directories due to its high number of invalidations. These frequent invalidations in turn generate a large number of data cache misses, which adds to the network traffic and produces higher miss ratios in the data caches.

When the tagged directories are very small relative to the number of unique block entries in the data caches, there are not enough tag entries to store the addresses of all of the memory blocks that the processors attempt to cache. These small directories then produce frequent set overflows in all three directories, which generates a large number of invalidation messages. As the number of tag entries increases, the performance of the superassociative directories tends to improve faster than the standard directory since they allow more flexible sharing of memory blocks. Additionally, since the superassociative directories have a few more tag entries for a given cost than the standard directory, they tend to have fewer set overflows. The D_2 directory generally outperforms the D_1 directory for most of the test programs primarily because of the sharing characteristics of the programs.

5. Conclusion

We have proposed a *superassociative* tagged directory structure that preserves some of the processors' cached copies of a memory block when a set overflow forces a replacement. Our Verilog models and trace-driven simulations show that a superassociative tagged directory can outperform a standard directory with the same implementation cost and cycle time. We conclude that superassociativity is an effective means of enhancing the memory performance of shared-memory multiprocessors that use a tagged directory to maintain coherence.

Acknowledgements

Thanks to Yat-tung Lam for his help with the Verilog and Hspice simulations, and to Farnaz Mounes-Toussi and Trung Nguyen for their assistance with the multiprocessor cache simulations.

References

1. A. Agarwal and A. Gupta, ''Memory-Reference Characteristics of Multiprocessor Applications Under MACH,'' *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pp. 215-225, 1988.
2. A. Agarwal, et al, ''An Evaluation of Directory Schemes for Cache Coherence,'' *Intl. Symp. Computer Architecture*, pp. 280-289, 1988.
3. J. Archibald and J.-L. Baer, ''An Economical Solution to the Cache Coherence Problem,'' *Intl. Symp. Computer Architecture*, pp. 355-362, 1984.
4. L. M. Censier and P. Feautrier, ''A New Solution to Coherence Problems in Multicache Systems,'' *IEEE Tran. Computers*, **C-27**(12):1112-1118, Dec. 1978.
5. D. Chaiken, J. Kubiatowicz, and A. Agarwal, ''LimitLESS Directories: A Scalable Cache Coherence Scheme,'' *Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 224- 234, 1991.
6. G. Cybenko, ''Supercomputer Performance Trends and the Perfect Benchmarks,'' *Supercomputing Review*, pp. 53-60, April 1991.
7. S. J. Eggers and R. H. Katz, ''A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation,'' *Intl. Symp. Computer Architecture*, pp. 373-382, 1988.
8. A. Gupta, W.-D. Weber, and T. Mowry, ''Reducing Memory and Traffic Requirements for Scalable

Directory-Based Cache Coherence Schemes,'' *Intl. Conf. Parallel Processing*, **I**:312-321, 1990.

9. D. J. Lilja, ''Cache Coherence in Large-Scale Shared Memory Multiprocessors: Issues and Comparisons,'' *ACM Computing Surveys*, **25**(3):303-338, Sept. 1993.
10. D. J. Lilja and P.-C. Yew, ''Improving Memory Utilization in Cache Coherence Directories,'' *IEEE Tran. Parallel and Distributed Systems*, **4**(10):1130-1146, Oct. 1993.
11. D. J. Lilja, ''The Impact of Parallel Loop Scheduling Strategies on Prefetching in a Shared-Memory Multiprocessor,'' *IEEE Tran. Parallel and Distributed Systems*, **5**(6):573-584, June 1994.
12. B. W. O'Krafka and A. R. Newton, ''An Empirical Evaluation of Two Memory-Efficient Directory Methods,'' *Intl. Symp. Computer Architecture*, pp. 138-147, 1990.
13. C. D. Polychronopoulos, et al, ''Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessors,'' *Intl. Conf. Parallel Processing*, **II**:39-48, 1989.
14. W.-D. Weber and A. Gupta, ''Analysis of Cache Invalidation Patterns in Multiprocessors,'' *Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 243-256, 1989.

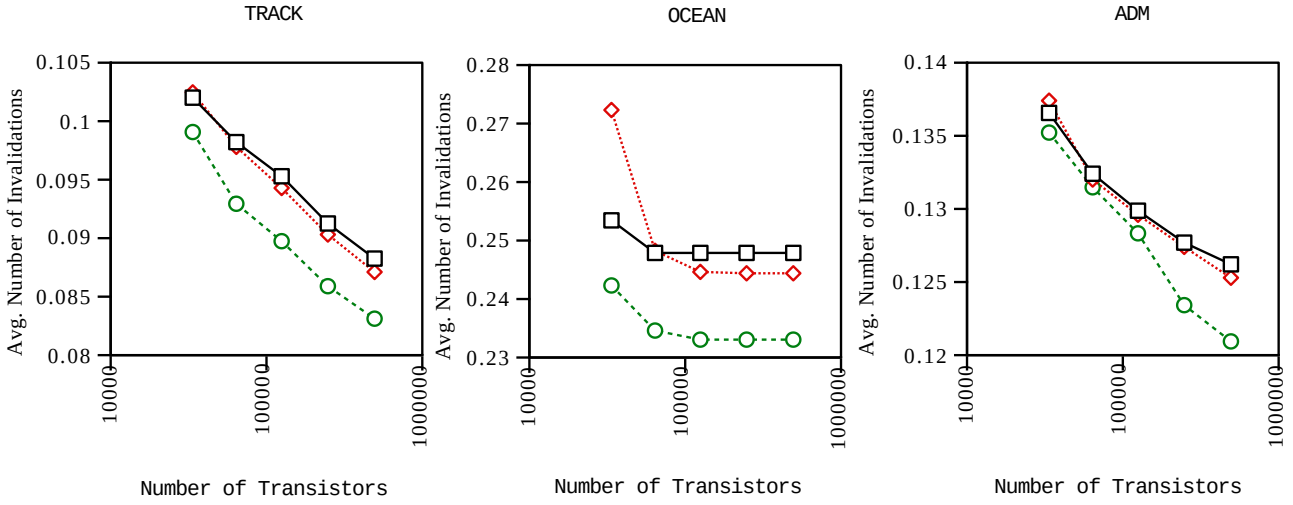

Figure 3: Average number of invalidations per memory reference.
solid line = D_0: standard directory (a=4, n=4); dotted line = D_1: superassociative directory (a=4, n=1); dashed line = D_2: superassociative directory (a=2, n=2).

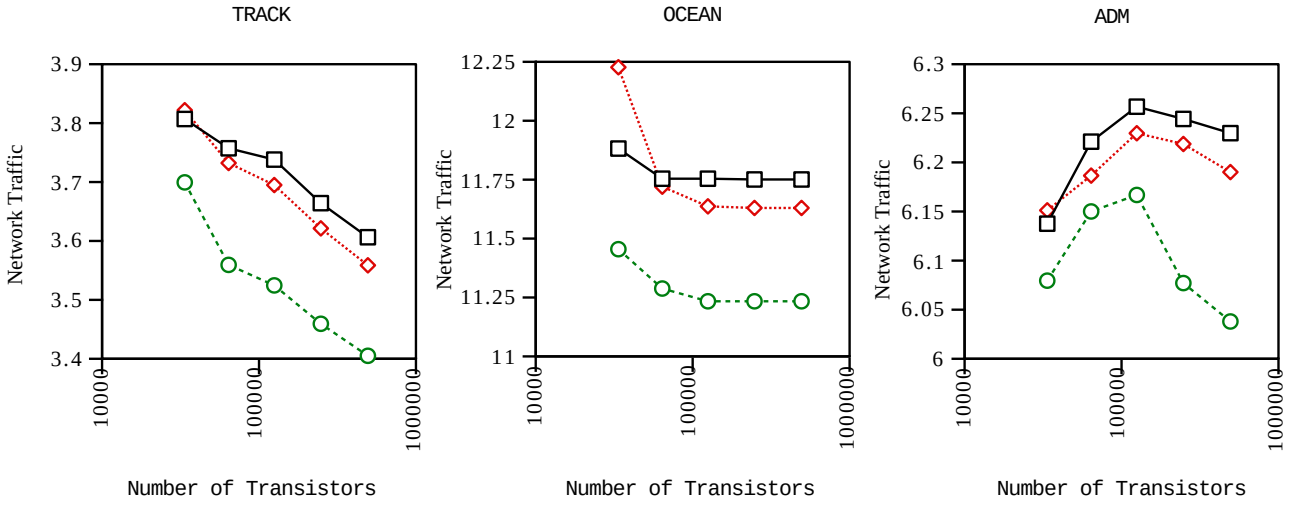

Figure 4: Average number of bytes transferred between the processors and memories per memory reference.
solid line = D_0: standard directory (a=4, n=4); dotted line = D_1: superassociative directory (a=4, n=1); dashed line = D_2: superassociative directory (a=2, n=2).