# Performance Evaluation of Memory Caches in Multiprocessors *

Yung-Chin Chen

MIPS Technologies, Inc.
Silicon Graphics, Inc.
Mountain View, CA 94039
ychen@mti.sgi.com

Alexander V. Veidenbaum

Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, IL 61801
alexv@csrd.uiuc.edu

*Large-scale MIN-based shared-memory multiprocessor systems have long shared memory latency. Private caches can improve memory access latency but they may suffer from the cache coherence problem and potentially lower data locality due to data sharing and multiprocessor scheduling. These two problems also increase shared memory load and may result in frequent memory stalls. In this paper, we evaluate the performance of memory caches, a cache memory placed in front of shared memory, in a large-scale multiprocessor system in the presence of processor caches. The memory cache is shown to have good performance and scalability.*

## 1 Introduction

Large-scale multiprocessor systems in which processors and memory modules are linked by interconnection networks have been proposed and designed in recent years. One of the major problems associated with this type of architecture is the speed of shared memory access due to long network latency and slow shared memory. Private caches and local memory that are associated with each processor are used to reduce the frequency of shared memory accesses. The use of private caches in multiprocessor systems introduces the cache coherence problem, and caches may not perform as effectively as those in uniprocessor systems.

The cache coherence problem may also increase the shared memory load because coherence maintenance can cause additional locality loss and memory updates. For software coherence schemes, there could be frequent cache invalidations. For directory schemes, the false sharing effect may induce unnecessary cache misses. Parallel processing can also cause spatial and temporal locality losses due to data sharing and multiprocessor scheduling. Our previous study [4] showed that a memory stall is still one of the major performance bottlenecks even when processor caches are present.

*This work was supported in part by the National Science Foundation under Grant No. NSF 89-20891, NASA Ames Research Center under Grant No. NASA NCC 2-559, and MIPS company.

In addition, the speed gap between processors and memory is increasing. The problem will become even worse in the future. Faster memory can serve memory requests more efficiently but at a higher cost. More interleaving can increase memory bandwidth but it is expensive and is not always effective, e.g. hot spots. A cost-effective approach is to place a cache memory in front of a shared memory module. The memory cache with high hit ratios turns a system having a slow shared memory access latency into one having a fast shared memory access latency.

The memory cache is not a new idea. It was proposed [9] as an alternative way to use cache memory without the cache coherence problem. It was also implemented in the Alliant FX/2800 machine. However, the improvement in performance due to memory caches is not as large as for processor caches because a memory request still has to traverse the interconnection networks to access data.

It is possible to have both types of caches in the system. The processor cache can eliminate most of the requests from a processor to shared memory, and the memory cache can reduce the access latency for these requests. In this paper, we evaluate the performance of memory caches in a large-scale multiprocessor system with processor caches. We study the performance improvement from the use of the memory cache, the effects of its configurations and its scalability.

## 2 Memory Caches

The use of a memory cache has several advantages. First, the memory cache does not induce any coherence problem, and its design is transparent to processor cache coherence protocols and architectures. Second, the implementation of memory caches can utilize the latest technology such as low-cost high-speed CDRAMs (Cache DRAM) [6]. A CDRAM integrates a cache into a large DRAM with fast block transfer and options of set associativity. Third, the memory cache has very high hit ratios, as will be shown in this paper.

We assume that the shared memory is interleaved in such a way that a cache line belongs to a single memory module. If we use a memory cache line size equal to that of the processor cache, the memory cache uses mainly the temporal locality of each memory line. It could utilize the spatial locality if it had a larger line size. The temporal locality depends mainly on the degree of sharing, either true sharing or false sharing. Therefore, it is affected by the number of processors, the line size, scheduling algorithm, and program behavior.

The temporal locality is high for a multiprocessor system due to data sharing and the cache coherence problem, the two major problems that reduce the hit ratios of processor caches. Part of the performance losses can be compensated by the memory cache. The sharing of lines causes unavoidable cold misses for each processor. When the memory cache is used, the first cold miss can bring the shared data into the memory cache, and subsequent cold misses by other processors can hit in the memory cache.

The improvement on performance by the memory cache can be thought of as increase of the processor cache hit ratio. Without considering the effects of overlapped memory accesses and resource conflicts, such an increase can be expressed by

$$\delta = (1 - h_{pc}) \times h_{mc} \times \frac{t_{mem} - t_{mc}}{t_{mem} - t_{pc}}. \qquad (1)$$

where $t_{pc}$ is the access time for a processor cache hit,
$t_{mc}$ is the access time for a memory cache hit,
$t_{mem}$ is the access time to shared memory,
$h_{pc}$ is the processor cache hit ratio,
and $h_{mc}$ is the memory cache hit ratio.

The improvement is proportional to the memory cache hit ratio and the processor cache miss ratio. In addition, it is related to the access times of the shared memory and caches. When the shared memory becomes slower or the memory cache becomes faster, the memory cache is more effective. In practice, the actual improvement in shared memory latency is hard to estimate without doing simulation.

## 3 System Architecture

The system organization that we simulate consists of an equal number of 32-bit RISC processors and shared memory modules interconnected by unidirectional, multistage Omega networks. The weak consistency model [7] is used. The processor data cache is a 64-KB direct-mapped cache, and the line size is 32 bytes.

We use the simple software scheme [8] to maintain processor cache coherence in most of our experiments. We also use the directory scheme in one experiment. The directory scheme in this study is based on Censier and Feautrier's distributed full-map directory scheme [2]. For the software scheme, we use a write-through write-allocate cache with a small write-back cache as its write buffer [3]. This write policy was shown to have better performance than pure write-through or write-back caches for software schemes.

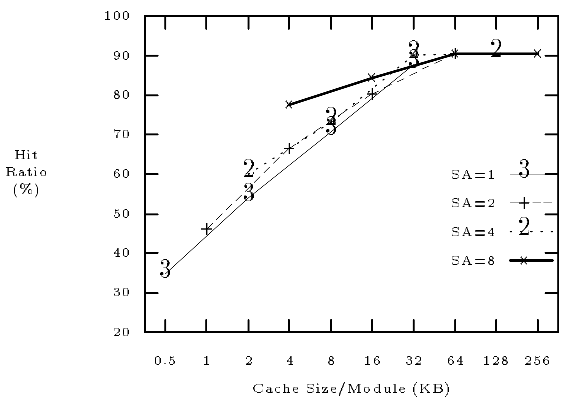

Figure 1: Average hit ratios of memory caches (SA: set associativity).

A memory cache is placed in front of each shared memory to handle all of the requests to shared memory. We assume that the memory cache uses the write-back policy. We observed that the write-through policy yields worse performance. The LRU (least-recently-used) replacement policy is used.

For more detailed description of these modules and their timing models, see [4].

Six Fortran programs are used as our benchmarks: *ARC3D, OCEAN, FLO52, TRFD, MDG,* and *MG3P.* Details about these benchmarks can be found in [1] [5]. Event-driven timing simulation is used in our study. More details about the trace generation, simulation methodology, and benchmark characteristics can refer to [4].

## 4 Performance Evaluation

In this section, we evaluate the performance of memory caches. Unless otherwise specified, we assume a 32-processor system; both processor and memory caches use 32-byte lines.

Figure 1 shows the overall hit ratios for the memory cache averaged over six benchmarks for different memory cache configurations. The set associativity is important only when the cache size is small. A direct-mapped 32KB cache is a cost-effective configuration, whose average hit ratio is approximately 88%, while the maximal average hit ratio is 91% for a larger cache or a higher set associativity. Most of the misses are read misses, and most of the read misses are cold start misses (see [4]).

The memory cache can have a line size larger than that of the processor cache and therefore capture spatial locality. A longer processor cache line will increase data trans-

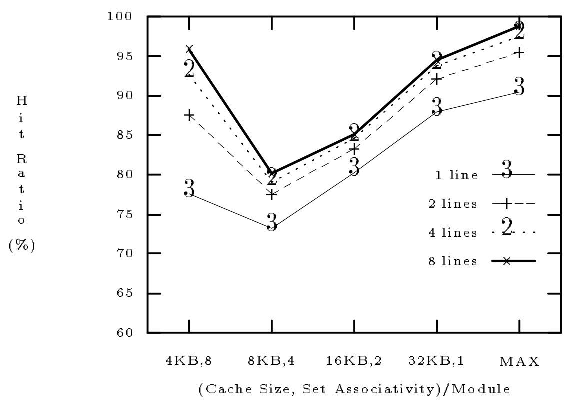

Figure 2: Hit ratios of memory caches with different line sizes (in number of processor cache lines).

fer time and cause more contention for caches, networks, and memory. It also increases false sharing for directory schemes. Consequently, the processor cache line size cannot be made too long. On the other hand, there is not so much overhead for memory caches because the physical location of shared memory and memory caches are close, and the data transfer does not have to traverse the networks.

Figure 2 shows the average hit ratios for the memory cache using memory cache line sizes that are equal to 1, 2, 4 and 8 processor cache lines, respectively. A longer line has higher hit ratios due to higher spatial locality. Increasing the line size from 1 to 2 processor cache lines yields the largest improvement. Any further increase in line size is less important. Also, when a long cache line is used, a small cache with a high set associativity can achieve hit ratios equivalent to a large cache with a lower set associativity. This appears to be an economic choice for memory caches if the use of a long cache line does not incur a lot of overheads.

Hit ratios alone do not characterize performance of caches. Some overheads, such as write stalls and network and memory contention, are not reflected in the hit ratios. Therefore, we also measured average memory latency. The latency for a given memory access consists of the default minimal cycles required (22 cycles in our case) and extra cycles due to resource contention (called the *stall cycles*). The memory cache can reduce memory latencies for those accesses that have processor cache misses in two ways. One is to provide faster memory access (i.e., a shorter minimal latency, 15 cycles in our case) for requests having memory cache hits. The other is to reduce stalls on memory modules. The system with memory caches has fewer memory stalls than the system without memory caches because the cache memory can serve requests faster so that the average

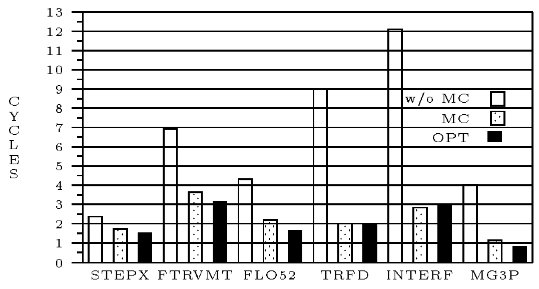

Figure 3: Comparison of average stall cycles for a read miss fetch between different memory systems.

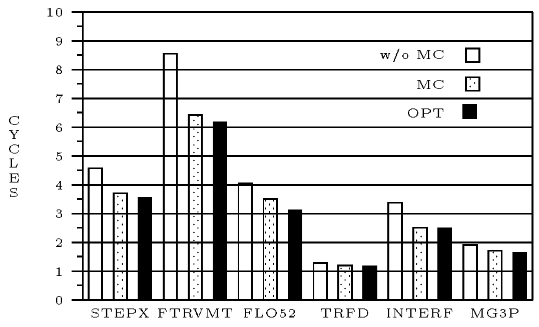

Figure 4: Comparison of average memory read latencies between different memory systems.

queuing delays of these requests would be shorter, and the load is shared among shared memory and memory caches.

To demonstrate how memory caches reduce stalls, Figure 3 shows the average stall cycles for a read miss fetch for systems with and without memory caches. The OPT system assumes 100% hit ratios for memory caches, a performance that can be thought of as the optimal performance of memory caches as well as that of a system using shared memory which is as fast as cache memory. The system with the memory cache (MC) uses a direct-mapped 64KB cache for each memory module, which yields maximal hit ratios for most of these benchmarks. In general, MC performs nearly as well as OPT because of its high hit ratios. Both eliminate most of the memory stalls that occur in the system without memory caches. However, they will increase network contention at the same time due to a higher traffic rate. The reduction in stall cycles for a read miss fetch ranges from 26% to 76% with an average 58% for MC. Write accesses to shared memory modules also benefit from the memory cache in the same way.

Figure 4 shows the average memory read latencies for the same systems. The reduction in read latency ranges from 7% to 26% with an average 17%. The reduction is not as much as that in stall cycles because there is a minimal latency required for each read miss, and the processor cache has high hit ratios that mask off the improvement coming from a shorter processor cache miss service time.

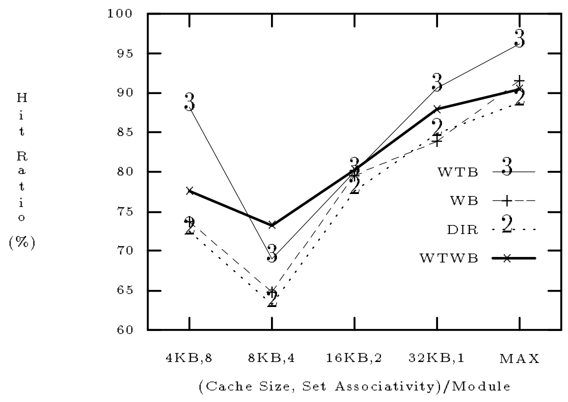

Figure 5: Comparison of hit ratios of the memory cache for different cache write policies and coherence schemes.

Figure 5 shows the average overall hit ratios of the memory cache on systems using the directory scheme (DIR) and the software scheme with write-back (WB), write-through (WT) and write-through with a write-back write-buffer (WTWB) caches, for several configurations of memory caches. In general, their hit ratios are not very different.

When the number of processors increases, the memory cache hit ratios increase for the same memory cache configuration (not shown here). When the system size grows, although the memory cache size for each module remains the same, the *combined* memory cache size becomes much larger. When the combined cache size is large enough to contain the problem size, further increase in the cache size will not improve hit ratios. We did observe that when two systems with different numbers of processors have memory caches with the same set associativity, cache line size and *combined* cache size, their memory cache hit ratios were about the same. The combined cache size that yields maximal hit ratios depends on individual programs rather than the system size. This argument is not always true for processor caches because in this case different caches may have a copy of the same memory line, whereas memory caches do not have sharing among each other.

## 5 Conclusions

We evaluated the use of memory cache in front of each memory module to further reduce shared memory access latency in the presence of processor caches. The memory cache does not have cache coherence problems. Its design is transparent to processor cache coherence protocols and architectures. We have shown that a reasonable size of memory cache can always achieve high hit ratios. Memory caches reduce average shared memory access latency in two ways. One is to reduce the minimal latency required for each processor cache miss when it has a memory cache hit. The other is to reduce memory stalls on shared memory modules. The memory cache can cut stall cycles by approximately a half.

We also evaluated the performance of memory caches for systems using directory schemes or software schemes with different processor cache write policies. They always achieve similar high hit ratios.

The memory cache hit ratios depend on the combined memory cache size, not the cache size per memory module. As the system size grows, we can reduce the memory cache size per module to maintain a constant combined memory cache size and still have similar performance.

## References

[1] D. Bailey and H. Simon. "The NAS Parallel Benchmarks". Technical report, NASA Ames Research Center, 1991.

[2] L.M. Censier and P. Feautrier. A New Solution to Coherence Problems in Multicache Systems. *Transactions on Computers*, C-27:1112–1118, November 1978.

[3] Y.-C. Chen and A.V. Veidenbaum. An Effective Write Policy for Software Coherence Schemes. In *Supercomputing' 92*, 1992.

[4] Y.-C. Chen and A. Veidenbaum. "Performance Evaluation of Memory Caches in Multiprocessors". Technical Report 1266, CSRD, University of Illinois at Urbana-Champaign, 1993.

[5] G. Cybenko et al. "Supercomputer Performance Evaluation and the Perfect Benchmarks". In *Int. Conf. on Supercomputing*, 1990.

[6] K. Dosaka et al. "A 100MHz 4Mb Cache DRAM with Fast Copy-Back Scheme". In *IEEE Int. Solid-State Circuits Conf.*, pages 148–149, 1992.

[7] M. Dubois et al. "Memory Access Buffering in Multiprocessors". In *Int. Sym. on Computer Architecture*, pages 434–442, 1986.

[8] A.V. Veidenbaum. "A Compiler-assisted Cache Coherence Solution for Multiprocessors". In *Int. Conf. on Parallel Processing*, pages 1029–1035, August 1986.

[9] P.C. Yeh et al. "Performance of Shared Cache for Parallel-Pipelined Computer Systems". In *Int. Sym. on Computer Architecture*, pages 117–131, 1983.