

EVALUATING OPTIMIZATIONS FOR MULTIPROCESSORS E-COMMERCE SERVER RUNNING TPC-W WORKLOAD

Pierfrancesco Foglia, Roberto Giorgi, Cosimo Antonio Prete
*Dipartimento di Ingegneria dell'Informazione
Facolta' di Ingegneria,
Universita' di Pisa
Via Diotisalvi, 2 – 56126 PISA (Italy)
{foglia,giorgi,prete}@iet.unipi.it*

Abstract

In this paper, the performance of an Electronic Commerce server, i.e. a system running Electronic Commerce applications is evaluated in the case of shared-bus multiprocessor architecture. In particular, we focused on the memory subsystem design and the analysis of coherence related overhead when the running software is setup as specified in the TPC-W benchmark.

Our aim is to individuate main factors that limit performance in such system, and the main optimization that can be done to speed-up the execution of E-Commerce workload on SMP architecture.

Our results show that: i) we need an accurate redesign of kernel data structure for large cache size; ii) cache affinity is useful in reducing cold and replacement miss, but it is not effective in every load-conditions; iii) passive sharing, i.e. the sharing induced by process migration, is a cause of performance degradation. A Write-Update protocol that correctly treats passive sharing (namely PSCR) permits two beneficial effects: increases performance in every situation and increases system scalability (up to 20 processor are permitted in our configuration)

1. Introduction

Many have thoughted the three-tiered architecture to best suite the typical architecture of E-Commerce systems [14], [3], [38], [36], [2], [16]. On tier one, the user machine runs a client program, typically a web-browser and/or Java applets; the client sends its requests to the server and receives the results to be shown to the end-user. Tier two includes a web-server that satisfies application specific requests, takes care of the load balancing and delivers standard services such as transaction management and site activity log. Tier three contains data and their managers, typically DBMS systems, to furnish credit-card

information, catalog information, shipping information, user information and so on. Tier two and three elements can be merged onto a single platform, or they can be distributed on several computers (clustered solution [22]).

In the following, we shall consider an E-Commerce server based on shared-bus shared-memory multiprocessor, and in particular, we shall focus on the core architecture related problems, rather than on software, network, and I/O related issues.

When dealing with E-Commerce server based on shared-bus shared-memory multiprocessor systems, design issues are scalability and speedup, due to the high variability of the load in these systems. These goals can be achieved by using cache memories, in order to hide the memory latency, and reduce the bus traffic (the main causes that limit speed up and scalability). Unfortunately, multiple cache memories introduce the coherence problem [18], [27], [28]. The coherence protocol has a great influence on the performance. Indeed, to guarantee cache coherence, the protocol needs a certain number of bus transactions (known as *coherence overhead*) that add up to the basic bus traffic of cache-based uniprocessors. Thus, a design issue is also the minimization of the coherence overhead. A typical solution adopted in commercial system for the coherence problem is the MESI protocol. This protocol might not be performance effective for shared-bus architecture, and in particular when process migration is allowed to maintain the load balance.

In this paper, we shall analyze hardware and software optimizations to improve the performance of a multiprocessor used as E-Commerce server. In our evaluation, the workload has been setup as specified in the TPC-W benchmark [33]. TPC-W simulates the activities of a business-oriented transactional web server. In our implementation, we used as component the Apache daemon [21], [5] several Unix utilities which both access file system and interface the various programs running on

the system, and a SQL server, namely PostgreSQL [37], [34]. The methodology relies on trace-driven simulation, by means of the “Trace Factory” environment [8], [19].

In the base case evaluation, we considered the MESI protocol since it is a widely employed solution. MESI is a Write Invalidate protocol [25], and it is used in most of the actual high-performance microprocessors, like the AMD K5 and K6, the PowerPC series, the SUN UltraSparc II, the SGI R10000, the Intel Pentium, Pentium Pro, Pentium II, Pentium III and Merced. MESI coherence overhead (that is the transactions needed to enforce coherence) is due to and *invalidate* transactions and *Invalidation Misses*.

We wish to relate that overhead with the kind of data sharing, in order to detect the causes for the coherence overhead. Three different types of data sharing can be observed: i) *true sharing*, which occurs when the same cached data item is referenced by processes running on different processors; ii) *false sharing* [30], which occurs when several processes running on different processors reference different data items belonging to the same memory block; iii) *passive* [28], [20] or *process-migration* [1] *sharing*, which occurs when a memory block, though belonging to a private area of a process, is replicated in more than one cache as a consequence of the migration of the owner process. Whilst true sharing is unavoidable, the other two forms of sharing are useless. The relevant overhead they produce can be reduced [29], [23], [17], [4], [12], [30] and possibly avoided [9].

2. E-Commerce Server and Workload Setup

We considered general cases of workloads suited for a multiprocessor, and not depending on the specific E-Commerce system. To this end, we setup the experiments as specified by the TPC-W benchmark [33], which specifies how to simulate the activities of a business-oriented transactional web server and exercises the breadth of system component associated with such environments. The TPC-W benchmark is particularly well suited to our evaluation since it does not specify the exact software architecture, nor the hardware architecture used to distribute the workload.

The application portrayed by the benchmark is a retail store with customer browse and order scenario. Customer visit the company web site, the store-front, to look at products, find information, place an order, or request the status of an existing order. The majority of the visitor activity is to browse the site. Some percentage of all visits result in submitting a new order.

The activity of a site client is described through 13 possible *web interactions* specified by the benchmark. Each web interaction describes both the web page content and the values to submit in case of forms. These values

are generally the inputs of queries invoked through the CGI model. A static diagram specifies the activations of next web interactions. The effective path followed by the client on that diagram is specified through probabilities defined in the benchmark.

TPC-W specifies that a certain number of entities (denominated Emulated Browser or EB) dynamically produce typical client activities for the server. Each activity generates a certain number of web interactions, and consequently, the exchange of a certain number of web objects. The number and the type of these exchanges are benchmark implementation specific.

In our experiment, 20 EB clients run on several workstations, connected to the simulated server via a LAN. In the benchmark, this number and the number of entries of ITEM tables define the dimension and the initial population of the DB. That population varies during the execution of the benchmark. In our case, the number of entries in ITEM tables is about 100K. This corresponds to a dimension of 80 MB for the ITEM table and a total dimension for the DB of 200MB.

3. Methodology

The methodology used in our analysis is based both on trace-driven simulation [24], [19], [35], and on the simulation of the three kernel activities that most affect performance: *system calls*, *process scheduling*, and *virtual-to-physical address translation*. We used the Trace Factory environment [8]. The approach used in this environment is to produce a process trace (a sequence of user memory references, system-call positions and synchronization events in case of multiprocess programs) for each process belonging to the workload by means of a modified version of Tangolite [10]. Then, the environment models the execution of workloads by combining multiple process-traces, generating the references of system calls, and by simulating process scheduling, and virtual-to-physical memory address translation. Trace Factory furnishes the references to a memory-hierarchy simulator [19].

Table 1. Statistics of source traces for some UNIX utilities (32-byte block size)

APPLICATION	DISTINCT BLOCKS	CODE (%)	DATA READ	DATA WRITE
AWK	9876	76.23	14.94	8.83
CP	5432	77.21	13.91	8.88
GZIP	7123	82.32	14.91	2.77
RM	2655	86.18	11.71	2.11
LS-AR	5860	80.23	13.98	5.79

Process management is modeled by simulating a scheduler that dynamically assigns a ready process to a processor. The process scheduling is driven by time-slice for uniprocess applications, whilst it is driven by time-slice

and synchronization events for multiprocess applications. Virtual-to-physical address translation is modeled by mapping sequential virtual pages into non-sequential physical pages.

Table 2. Statistics of multiprocess application source traces (Apache and SQL) and workload (EC-Server), in case of 32-byte block size.

WORKLOAD	NUMBER OF TASK	DISTINCT BLOCKS	CODE (%)	DATA(%)		SHARED BLOCKS	SHARED(%)	
				READ	WRITE		ACCESSES	WRITE
Apache	13	34311	73.84	19.18	6.99	1105	1.84	0.6
SQL	8	24141	71.94	18.17	9.89	5838	2.70	0.79
EC-Server	26	112183	75.49	17.12	7.39	6101	1.68	0.54

Table 3. Numerical values of timing parameters for the multiprocessor simulator (timings are in clock cycles).

CLASS	PARAMETER	TIMING	
		32 BYTES	128 BYTES
CPU	READ/WRITE CYCLE	2	2
BUS	INVALIDATE TRANSACTION	5	5
	WRITE TRANSACTION	5	5
	MEMORY-TO-CACHE READ -BLOCK TRANSACTION	68	80
	MEMORY-TO-CACHE READ-AND-INVALIDATE-BLOCK TRANSACTION	68	80
	CACHE-TO-CACHE READ-BLOCK TRANSACTION	12	24
	CACHE-TO-CACHE READ-AND-INVALIDATE BLOCK TRANSACTION	12	24
	UPDATE-BLOCK TRANSACTION	6	18

The ‘EC-Server’ workload is constituted of 13 processes spawned by the Apache daemon, 8 by PostgreSQL, and 5 processes are Unix utilities. Table 1 (for the uniprocess applications) and Table 2 (for the multiprocess ones) contain some statistics of the process traces used to generate the workload for a 32-Byte block size.

Trace Factory includes a multiprocessor simulator, which characterizes a shared-bus multiprocessor in terms of CPU, cache and bus parameters. The simulated processors are MIPS-R10000 ones; paging relays on 4-KByte page size (the default size of MIPS R10000 [26]). Each processor uses a write buffer thus implementing a relaxed model of memory consistency, in particular the processor consistency [7], [39]. Finally, the bus parameters are the number of CPU clock cycles for each kind of transaction: write, invalidation, update-block, memory-to-cache read-block, and cache-to-cache read-block. The bus supports transaction splitting.

The simulator classifies the coherence overhead by analyzing the access patterns to shared data (true, false e passive sharing [20]). The type of access pattern to the cache block determines the type of the invalidation-miss. The classification is based on an existing algorithm [11],

extended to the case of passive sharing, finite size caches, and process migration.

4. Simulation Results

We considered the following multiprocessor configurations: a 4-processor machine and several other ‘‘high-end’’ architectures (8, 12, 16 processors). Each processor has a private cache, whose size has been varied between 128 KBytes and 2 MBytes, whilst for block size we considered 32, 64, 128 and 256 Bytes. We considered a 128 bit shared bus. For the scheduling policy two solutions have been analyzed: random and cache-affinity [31]; scheduler time-slice is equivalent to about 200,000 references. The bus timing relative to these case studies are reported in Table 3.

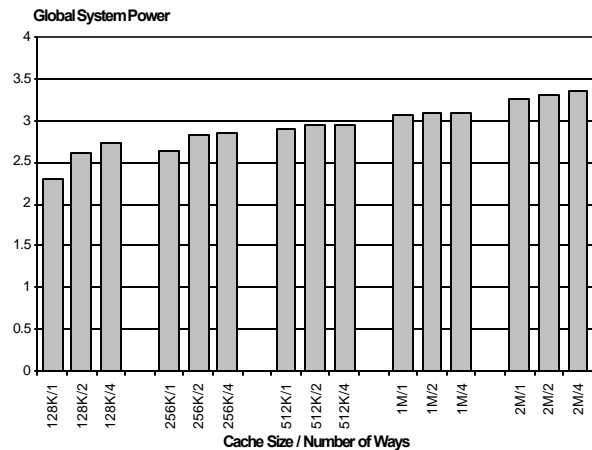


Figure 1. Global System Power (GSP) versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) and number of ways (1, 2, 4), for a 4-processor system, random scheduling policy, and 32-Byte block size.

4.1. Initial Analysis

In our initial analysis, we considered a 4-processor system having a 32-Byte cache block size, when cache capacity and associativity are varied. The system adopts MESI protocol, and thus has the following bus transactions: *read-block*, *read-and-invalidate*, *invalidate*, and *update* transactions.. Therefore, the main part of traffic is due to classical misses (sum of cold, conflict, and capacity misses [39]) and coherence traffic, constituted of misses due to the invalidation of actual shared copies and *invalidate* transactions.

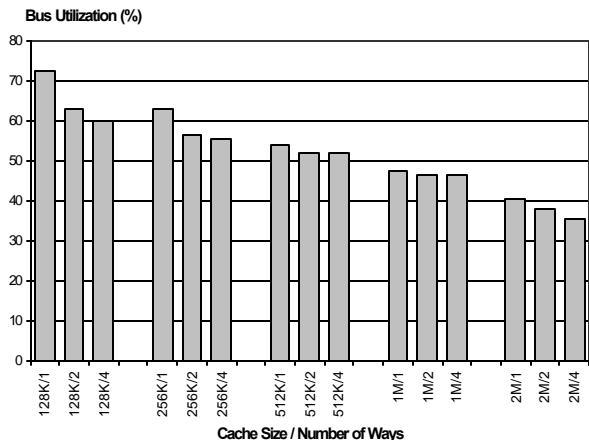


Figure 2. Bus Utilization (in percentage) versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) and number of ways (1, 2, 4), for a 4-processor system, random scheduling policy, and 32-Byte block size. The less the bus utilization, the more system power can be gained by adding new processors in the system.

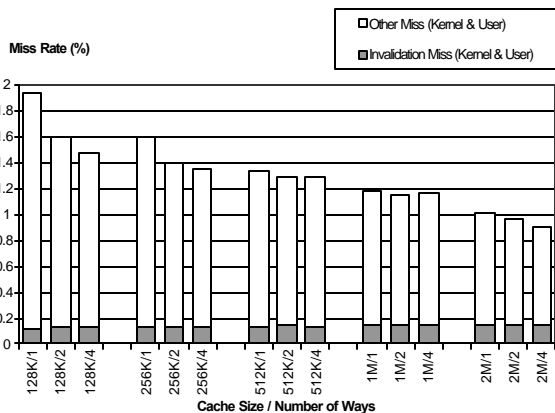


Figure 3. Breakdown of miss rate versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) and number of ways (1, 2, 4), for a 4-processor system, random scheduling policy, and 32-Byte block size. Miss Rate decreases with the cache size, mainly because of Other Miss. "Other Miss" includes cold miss, capacity miss, and conflict miss.

The GSP graph (Figure 1) shows, as expected, that we can obtain a more powerful machine by increasing the cache size. The larger are the caches, the more scalable is the machine. Indeed, we can define the scalability of a multiprocessor system up to N processors as the number N of processors that causes the GSP to drop by more than 0.5 when the processors are increased from N to N+1 (we verified that this definition is equivalent to the definition of 'critical point' in [9].) By using this definition, we calculated that the machine we are considering is scalable

up to 4 processors in the case of 128-KByte direct access cache, and up to 9 processors in the case of 2-MByte 4-way cache. The higher scalability is essentially due to lower bus utilization when adopting larger caches (Figure 2). The reduction of bus traffic with the cache size and associativity is due to the lower miss rate, and in particular to the lower 'other-miss' rate (including cold, conflict and capacity misses) (Figure 3).

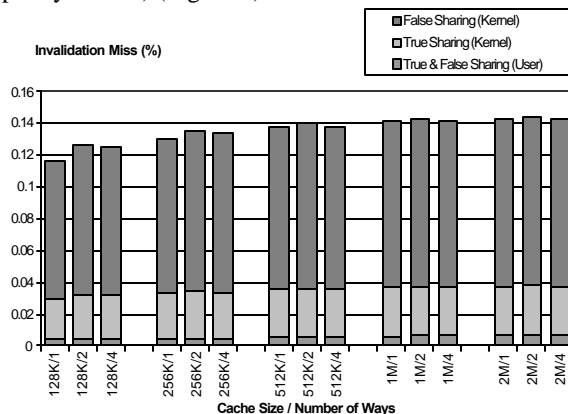


Figure 4. Breakdown of invalidation miss rate versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) and number of ways (1, 2, 4), for a 4-processor system, random scheduling policy, and 32-Byte block size

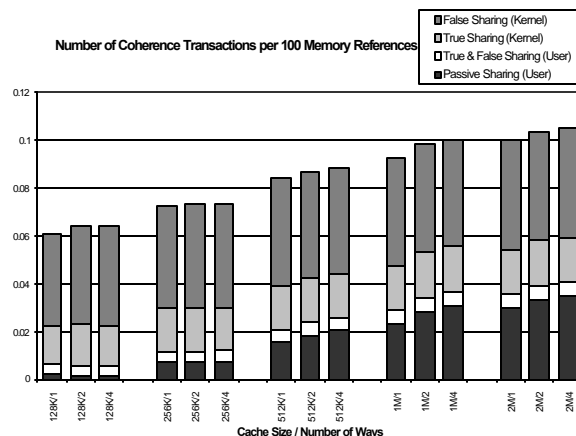


Figure 5. Number of coherence transactions (invalidate transactions) versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) and number of ways (1, 2, 4), for a 4-processor system, random scheduling policy, and 32-Byte block size.

At this point, it is clear how the reduction of miss rate plays an essential role to determine system performance. We can reduce the traditional misses (the 'other-misses') by using classical techniques [39], [6] (in particular modifying cache associativity, cache size, and cache block size,) or by using program restructuring techniques [13],

[32], [15]). On the other hand, the effects of these techniques on invalidation misses may be more unpredictable: we know we can intervene on them by using an appropriate coherence protocol. In our case study, invalidation misses do not decrease (Figure 4 and, more in detail, Figure 5) with the cache size. We observe, on the contrary, a slight decrease.

Coherence overhead (invalidation misses and coherence transaction) increases with the cache size and associativity (Figures 4 and 5,) and it weighs, in percentage, more and more on the performance. In this case, most of the coherence overhead (Figure 5) is due to false sharing generated in the kernel. True sharing is present in the kernel, whilst it is limited in the application user area. Passive sharing increases as the cache capacity is increased, since the average lifetime of a cache copies increases as well (Figure 5).

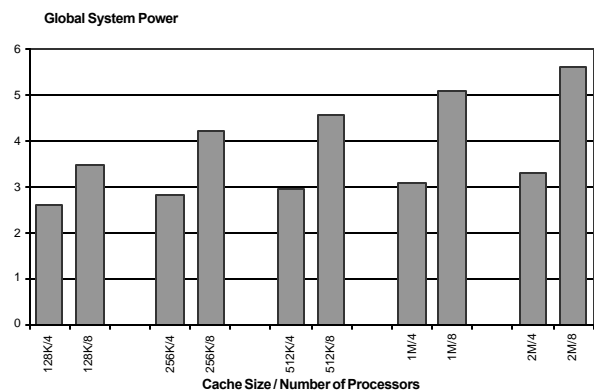


Figure 6. Global System Power versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes), in case of 4 and 8 processors, random scheduling policy, 32-Byte block size and two-way set associative cache. The GSP increase is higher in the 8-processor case as the cache size increases. This is due to the higher bus saturation in the 8-processor case: in this situation, advanced techniques for reducing bus utilization and miss rate are crucial.

4.2. Scaling Up the Architecture

Let us consider the 8-processor configuration. We have seen that this configuration is near the scalability limit of the machine. We considered only the case of a 2-way set associative cache for the sake of simplicity. After several experiments we found that an optimal block size for the system is 128 byte. The system is working with the bus almost in full saturation (Figure 7.) The GSP can be increased, and the bus utilization reduced, by using larger caches (Figure 6) and higher associativity.

We observe a miss rate increase (Figure 8) and an increase in the number of invalidations (Figure 9). The

miss increase is due both to the increased invalidation miss rate, in turn due to the higher parallelism of the system (causing a higher probability that a shared block is used by a higher number of processors,) and to the higher ‘other miss’ rate caused by the higher number of context-switch misses (the misses generated when reloading the working set of a newly scheduled process) related to the higher number of migrating processes.

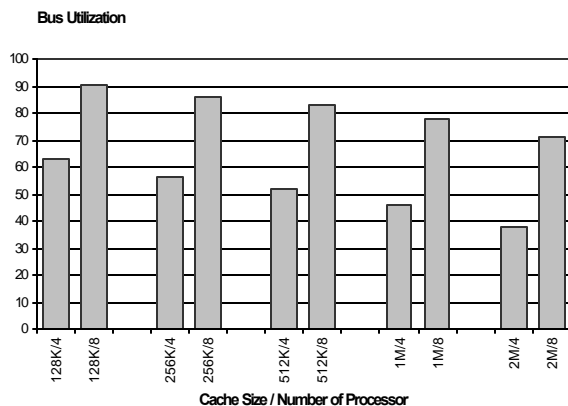


Figure 7. Bus Utilization versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes), in case of 4 and 8 processors, random scheduling policy, 32-Byte block size, and two-way set associative cache.

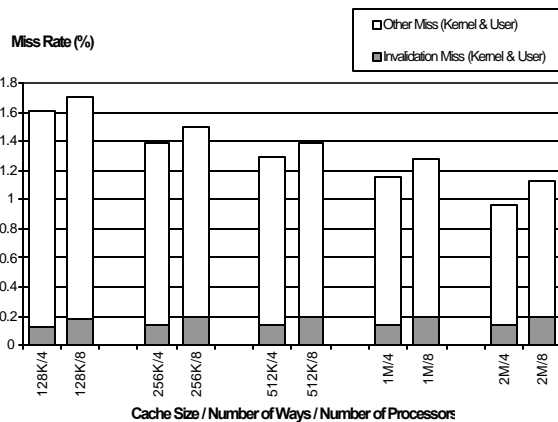


Figure 8. Breakdown of miss rate versus cache sizes (128 K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) in case of 4 and 8 processors, random scheduling policy, 32-Byte block size, and two-way set associative caches. Both Invalidation and other misses increases on the 8-processor configuration. The higher migration causes the ‘other miss’ rate increase. Invalidation misses increase due to the higher parallelism of the machine: the probability that a certain shared block is used by more processors increases.

The invalidation miss increase (Figure 9) is essentially due to the kernel activity, and in particular to the false sharing. Consequently, we notice that a special effort is needed to organize kernel data structures. This could be easily accomplished since the kernel is a completely known part of the system at design time. False sharing can be eliminated either by using special coherence protocols [29], or by properly allocating the involved shared data structures [12] or by means of data restructuring through profiling information [12], [30].

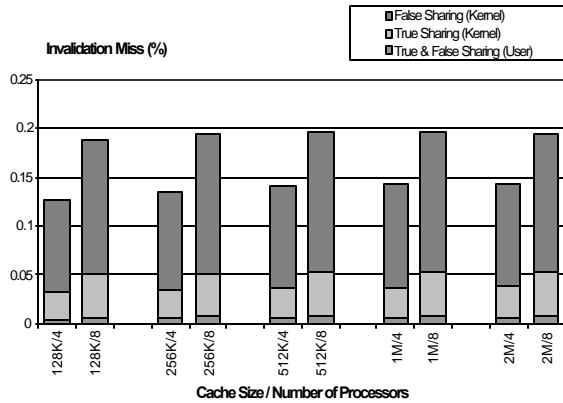


Figure 9. Breakdown of Invalidation miss rate versus cache sizes (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes) in case of 4 and 8 processors, random scheduling policy, 32-Byte block size for two-way cache.

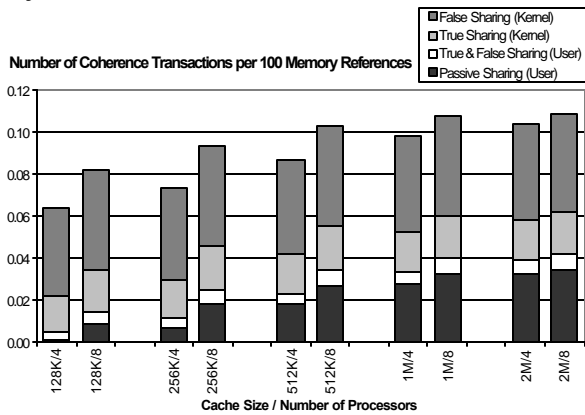


Figure 10. Number of coherence transactions (invalidate transactions) versus cache size (128K Bytes, 256K Bytes, 512K Bytes, 1M Bytes, 2M Bytes), in case of 4 and 8 processors, random scheduling policy, 32-Byte block size and two-way set associative cache. Each component of this overhead increases in the 8-processor configuration.

Also coherence transactions increase (Figure 10), essentially due to the increased passive sharing. As in the

4-processor case, passive sharing becomes more significant with larger caches. Thus, the larger caches adopted in current systems enhance the passive sharing overhead.

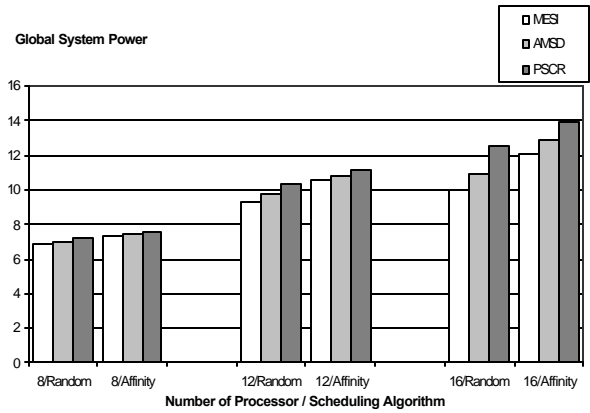


Figure 11. Global System Power versus number of processor (8, 12, 16) and scheduling algorithm (random, affinity). Data assume 2-MByte cache size. Cache is a 2-way set associative with 128-Byte block size.

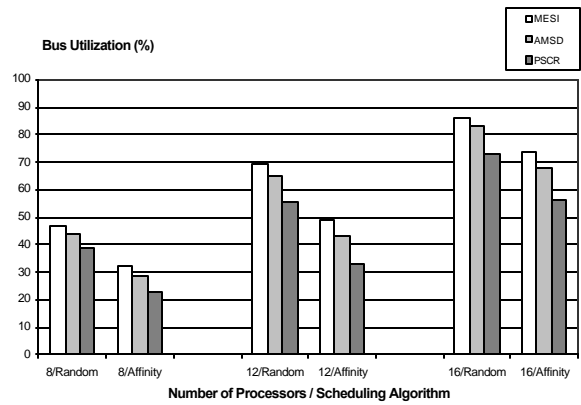


Figure 12. Bus utilization versus number of processor (8, 12, 16) and scheduling algorithm (random, affinity). The cache has 128-Byte block size, 2-MByte cache size, and it is 2-way set associative.

Thus, we can increase the performance of the 8-processor system by intervening on several aspects: i) on the ‘other misses’, ii) on the kernel false sharing, iii) by limiting the effects of process migration. We can intervene on the point i) and ii) by increasing the block size. As for the effects on performance caused by the process migration we can modify both on the scheduling policy and the coherence protocol. In the following we analyze how we can increase the system scalability by intervening on the coherence protocol.

As we observe little sharing in the user area, we can avoid to use a specific coherence protocol for the true and false sharing, and we can reduce kernel false sharing by using data restructuring techniques for kernel data [40] [30]. Considering that process migration may be unavoidable since it allows for a load balancing among processors, and that process migration becomes more significant as the number of processors increases and larger caches are used, it appears convenient to use coherence protocols that help reduce passive sharing. For this reasons, we considered two coherence protocols that reduce or eliminate passive sharing. The first is based on Write-Update technique and the second on a Write-Invalidate technique. They are respectively, PSCR [9] and AMSD (Adaptive Migratory Sharing Detection) [23], [4].

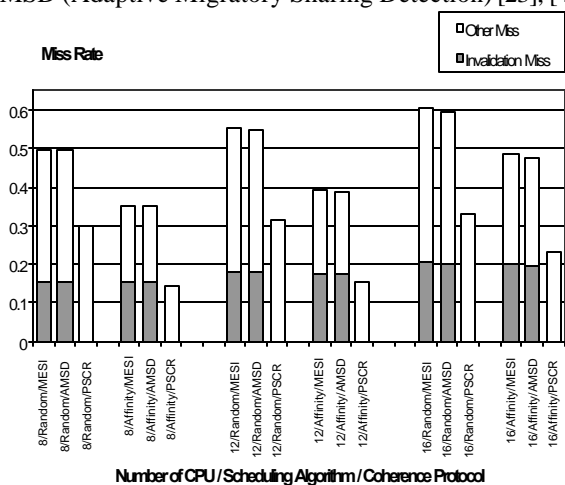


Figure 13. Miss Rate versus number of processor (8, 12, 16) and scheduling algorithm (random, affinity). The cache has 128-Byte block size, 2MByte cache size, and it is 2-way set associative.

To locate scalability limits, our analysis has been conducted by varying both scheduling policy and the number of processors. As can be observed (Figure 11) as the number of processors increases, the performance difference among protocols becomes more evident. In particular, the choice of MESI protocol appears the most penalizing. This is due to the non-selective invalidation technique of MESI.

AMSD has beneficial effects on passive sharing although it does not eliminate it completely. The benefits on passive sharing are due the little reduction of total misses (Figure 13) and to a decrease of coherence transactions (Figure 14.) The reduction of coherence transaction is due to the behavior of AMSD. When AMSD detects a block that has to be treated exclusively for a long time interval, it invalidates the copy locally

during the handling of a remote miss, thus avoiding a necessarily consequent bus transaction.

PSCR is based on an update of a effectively shared copy, thus avoiding invalidation misses. By using the write-update technique, the number of coherence transactions result higher compared to other protocols (Figure 14.) On the other side, the total number of misses produces a more consistent bus utilization reduction (Figure 12.) Moreover, the cost of the coherence overhead is somewhat limited by the lower cost of the coherence maintaining write operations (cfr. Table 3.) The use of write operation is also more advantageous since that operation can be performed asynchronously, without a direct processor delay. Finally, the write cost is independent from the block size.

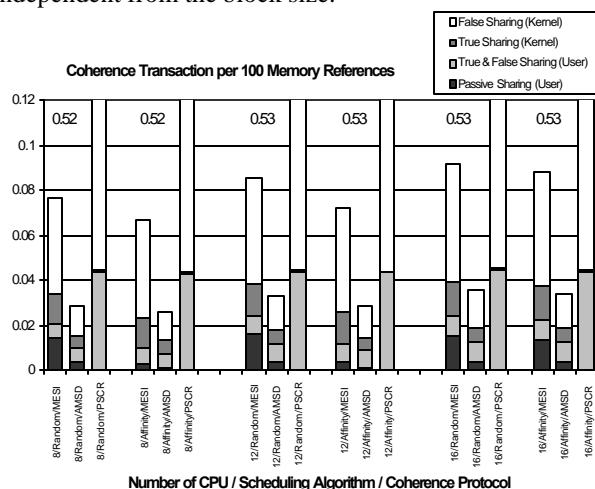


Figure 14. Coherence Transactions versus number of processor (8, 12, 16) and scheduling algorithm (random, affinity). The cache has 128-Byte block size, 2-MByte cache size, and it is 2-way set associative.

Let us now analyze the scalability offered by the various protocols. As observed previously, the system is in saturation when the GSP does not increase of a minimal quantity as the number of processors is increased. In our case, when switching from 8 to 12 processor (or from 12 to 16) the threshold beyond which the system is in saturation corresponds to a GSP increase of 2 (Figure 11.) Based on these considerations, we can conclude that the system adopting the MESI protocol (in case of random scheduling policy) is already at the saturation threshold when switching from 8 to 12 processors. The system is in full saturation when switching from 12 to 16 processors with both scheduling policies.

As for AMSD, the situation is a slightly better when switching from 8 to 12 processor, whilst we observe again full saturation when switching from 12 to 16 processors (with both scheduling policies).

PSCR is never in saturation in the shown configuration, thus justifying its adoption when higher performance (GSP) is needed. We also observe that in configurations with a lower number of processors, the choice of a different protocol is less critical. When the performance is pushed to the limits (and consequently the system works near saturation) the designer should take advantage of more optimization techniques like smart coherence protocols.

The combination of all analyzed techniques (adequate block size, cache affinity, and PSCR) allows us to push system scalability up to 20 processors with a corresponding GSP of about 16.

5. Conclusion

In this paper, we analyzed some techniques that improve the performance of a shared-bus multiprocessor used as an Electronic Commerce server system. In particular, we have analyzed the memory subsystem, whose performance depends heavily on the miss rate and bus traffic induced on the shared-bus.

Our workload has been set up by considering software components like an HTTP server (Apache), PostgreSQL DB-server, and typical UNIX shell commands, according to the specification of the TPC-W benchmark.

As the number of processor increases, the goal of reducing coherence overhead and bus traffic becomes essential, in order to achieve good performance. In this case, we can use classical techniques to reduce the 'other-miss' rate, but it is crucial also to reduce coherence overhead. In case of false sharing, coherence overhead can be reduced by means of static restructuring techniques. In case of passive sharing, a specific coherence protocol has to be preferred. The adoption of PSCR allows us to extend the multiprocessor scalability at least up to 20 processors for the experiments that we carried out.

From the evaluations carried out, we can extract useful suggestions for application developers, kernel and architecture designers. First of all, when designing E-Commerce Server systems, the reduction of classical misses has to be achieved by using techniques that can enhance the locality of the program, and other traditional solutions.

Then, kernel designers should take into account false sharing and thus false sharing misses have to be reduced by using kernel structure restructuring techniques. This could be easily achieved, since the kernel is a well-know part of the system at design time.

As for architectural aspects, in the case of bus-based multiprocessors, MESI protocol is sufficient for configurations having a not so high number of processors

(8 in our experiments). If a higher performance is needed, the increase of number of processor really produces benefits, if other miss reduction techniques are considered. In particular, coherence protocols like PSCR produce performance benefits by eliminating coherence overhead due to passive sharing, without generating useless invalidation misses.

6. References

- [1] A. Agarwal and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under Mach". *Proc. ACM Sigmetrics*, Santa Fe, NM, pp. 215-225, May 1998.
- [2] J.M. Andreoli, Francois Paculli, and R. Pareschi, "XPECT: A Framework for Electronic Commerce", *IEEE Internet Computing*, vol. 1, no. 4, pp. 40-48, July-August 1997.
- [3] R. Brandau, T. Confrey, A. D'Silva, C.J. Matheus, R. Weihmayer, "Reinventing GTE with Information", *IEEE Computer*, vol. 32, no. 3, pp. 50-58, March 1999.
- [4] A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data," *Proc. 20th Int'l Symp. on Computer Architecture*, San Diego, California, pp. 98-108, May 1993.
- [5], J. Edwards, "The changing Face of Freeware". *IEEE Computer*, vol. 31, no. 10, pp. 11-13, October 1998.
- [6], M. J. Flynn, *Computer Architecture*, Pipelined and Parallel Processor Design. Jones and Bartlett Publishers, 1995.
- [7] K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors". *Proc. of the Fourth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, California, pp. 245-357, Apr. 1991.
- [8] R. Giorgi, C. Prete, G. Prina and L. Ricciardi, "Trace Factory: a Workload Generation Environment for Trace-Driven Simulation of Shared-Bus Multiprocessor". *IEEE Concurrency*, vol. 5, no. 4, pp. 54-68, Oct-Dec 1997.
- [9] R. Giorgi, C.A. Prete, "PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, pp. 742-763, vol. 10, no. 7, July 1999.
- [10] S. R. Goldschmidt and J. L. Hennessy, "The Accuracy of Trace-Driven Simulations of Multiprocessors". *Proc. of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 146-157, May 1993.
- [11] R. L. Hyde and B. D. Fleisch, "An Analysis of Degenerate Sharing and False Coherence". *Journal of*

Parallel and Distributed Computing, vol. 34, no. 2, pp. 183-195, May 1996.

[12] T. E. Jeremiassen and S. J. Eggers, "Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations", *ACM SIGPLAN Notice*, vol. 30, no. 8, pp.179-188, August 1995.

[13] J. Kalamatianos, A. Khalafi, D. Kaeli, W. Meleis, "Analysis of Temporal-Based Program Behavior for Improved Instruction Cache Performance", *IEEE Transactions on Computers*, Vol. 48, No. 2, February 1999.

[14] T. Lewis, "The Legacy Maturity Model". *IEEE Computer*, vol. 31, no. 11, pp. 125-128, November 1998.

[15] S. Lorenzini, G. Luculli, C. A. Prete, "A Fast Procedure Placement Algorithm for Optimal Cache Use", Proc. of the *MELECON'98*, Tel Aviv, Israel, pp 1279-1284, May 1998.

[16] V. Milutinovic, *System Support for Electronic Business on Internet*. <http://galeb.etf.bg.ac.yu/~vm/books/2001/ebi.html>

[17] C. A. Prete, "A New Solution of Coherence Protocol for Tightly Coupled Multiprocessor Systems," *Microprocessing and Microprogramming*, vol. 30, no. 1-5, pp. 207-214, 1990.

[18] C. A. Prete, "RST Cache Memory Design for a Tightly Coupled Multiprocessor System," *IEEE Micro*, vol. 11, no. 2, pp. 16-19, 40-52, Apr. 1991.

[19] C.A. Prete, G. Prina, and L. Ricciardi, "A Trace Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor System". *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 9, pp. 915-929, September 1995

[20] C. A. Prete, G. Prina, R. Giorgi, and L. Ricciardi, "Some Considerations About Passive Sharing in Shared-Memory Multiprocessors". *IEEE TCCA Newsletter*, pp. 34-40, Mar. 1997.

[21] D. Robinson and the Apache Group, APACHE – An HTTP Server, Reference Manual, 1995. <http://www.apache.org>.

[22] R. Short, R. Gamache, J. Vert and M. Massa, "Windows NT Clusters for Availability and Scalability", In Proceedings of the 42nd IEEE International Computer Conference, pp. 8-13, San Jose, CA February 1997.

[23] P. Stenstrom, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing". Proc. of the 20th Annual Int'l Symp. on Computer Architecture. San Diego, CA, May 1993.

[24] C. B. Stunkel, B. Janssens, and W. K. Fuchs, "Address Tracing for Parallel Machines," *IEEE Computer*, vol. 24, no. 1, pp. 31-45, Jan. 1991.

[25] P. Sweazey and A. J. Smith, "A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus". Proc. of the 13th International Symposium on Computer Architecture, pp. 414-423, June 1986.

[26], D. Sweetman, *See Mips Run*. Morgan Kaufmann Publishers, Inc. San Francisco, CA. 1999.

[27], M. Tomasevic and V. Milutinovic, "*The Cache Coherence Problem in Shared-Memory Multiprocessors – Hardware Solutions*". IEEE Computer Society Press, Los Alamitos, CA, April 1993.

[28], M. Tomasevic and V. Milutinovic, "Hardware Approaches to Cache Coherence in Shared-Memory Multiprocessors". *IEEE Micro*, vol. 14, no. 5, pp. 52-59, Oct. 1994 and vol. 14, no. 6, 61-66, Dec. 1994.

[29], M. Tomasevic and V. Milutinovic, "The Word-Invalidate Cache Coherence Protocol", *Microprocessors and Microsystems*, pp. 3-16, vol. 20, Mar. 1996.

[30] J. Torrellas, M. S. Lam, and J.L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches". *IEEE Transactions on Computer*, vol. 43, no. 6, pp. 651-663, June 1994.

[31] J. Torrellas, A. Tucker, and A. Gupta, "Evaluating the Performance of Cache-Affinity Scheduling in Shared-Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 24, no. 2, pp. 139-151, Feb. 1995.

[32], J. Torrellas, R. Daigle. "Optimizing the Instruction Cache Performance of the Operating System". *IEEE Transactions on Computers*, Vol. 47, No. 12, December 1998.

[33] TPC BENCHMARK W (Web Commerce) Specification, version 1.0.1. Transaction Processing Performance Council, February 2000.

[34], P. Trancoso, J. L. Larriba-Pey, Z. Zhang, and J. Torrellas, "The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors". *Proc. of the 3rd Intl. Symp. on High Performance Computer Architecture*, Feb 1997.

[35] R. A. Uhlig and T. N. Mudge, "Trace-Driven Memory Simulation: a survey". *ACM Computing Surveys*, pp. 128-170, June 1997.

[36] D. W. Walker, "Free-Market Computing and the Global Economic Infrastructure", *IEEE Parallel and Distributed Technology*, Vol. 4, no. 3, pp. 60-62, FALL 1996.

[37] A. Yu and J. Chen, The POSTGRES95 User Manual. Computer Science Div., Dept of EECS, University of California at Berkeley, July 1995.

[38] J. Edwards, *3-Tier Client/Server At Work*. Wiley Computer Publishing, New York, N.Y., 1999.

[39] J. Hennessy and D. A. Petterson, *Computer Architecture: a Quantitative Approach, 2nd edition*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[40] T. E. Jeremiassen and S. J. Eggers, "Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations", *ACM SIGPLAN Notice*, vol. 30, no. 8, pp.179-188, August 1995.