OS Effects on Memory Hierarchy of a SMP Multiprocessor Running a DBMS Workload

Pierfrancesco Foglia, Cosimo Antonio Prete

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
Pisa, Italy
{foglia, prete}@iet.unipi.it

Roberto Giorgi

Dipartimento di Ingegneria dell'Informazione
Università di Siena
Siena, Italy
giorgi@acm.org

Abstract

In this work, we characterized the impact of operating system activities like process migration on a shared-bus shared-memory multiprocessor running typical DBMS workload.

Our workload has been set-up utilizing the TPC-D benchmark on the PostgreSQL DBMS. Analysis has been performed via trace driven simulation enhanced technique which includes most important operating system activities and analyzes the sharing overhead in detail.

We evaluated a basic four-processor and a high-end sixteen-processor machine, implementing MESI and other coherence protocols that deal with migration of processes and data. Our results show that even in the four-processor case operating system effects may not be neglected. In fact, different coherence protocols can more effectively reduce the effects of process migration.

The consequences on performance become more important in high-end machines (16 or more processors). In this case, even little sharing, as we found in DBMS applications can become crucial for system performance. Better speed up may be achieved adopting several alternatives including redesign of kernel data structure. Cache affinity is somewhat useful in reducing migration effect, but it is not effective in every load conditions.

Keywords: Shared-Bus Multiprocessors, DBMS, Cache Memory, Coherence Protocol, Sharing Analysis, Sharing Overhead.

1 Introduction

Shared-bus shared-memory multiprocessors (basically Symmetric Multi Processor, or SMP, architectures) are becoming more and more widespread since they are a simple and quite cheap solution to speed up complex workloads demanding for high performance like, just to give some examples, databases, file servers and application servers [19]. In shared bus architectures, processors access the shared memory through a shared bus. This bus is the bottleneck of the system, since it can easily reach a saturation condition, thus limiting the performance and the scalability of the machine.

The classical solution to overcome this problem is the use of per-processor cache memories [13]. Cache memories introduce the coherency problem and the need for adopting adequate coherence protocols [22], [23]. Coherence protocols generate a certain number of bus transactions, thus accounting for a non-negligible overhead in the system (coherence overhead). The traffic induced by the coherence overhead adds up to the basic bus traffic, necessary to access the main memory [9]. Coherence overhead may have a negative effect on the performance as shown in the literature [4], [22], [24], [9] and different optimizations have been proposed, both at compile time, such as the redesign of shared data structure [33] and at the architectural level, as the adoption of adequate coherence protocols [32] [9].

An important case of these systems include databases [19]. Due to their increasing importance, the behavior of these workloads has been extensively analyzed for the aspects concerning processor architecture [30], [31] and the memory system [26], [29]. However, as of our knowledge, there are no studies that include operating system effects on memory performance of these workloads. Operating system activity, in shared bus shared memory multiprocessor, mainly acts in two way [34], [35]: shared kernel data structures generates coherence overhead; process migration, needed to achieve load balancing in such systems, increases the number of cold/conflict miss and generates also coherence overhead, known as *passive sharing* overhead [9].

In this work we compare memory hierarchy behavior and justify performance issue of SMP architectures based on MESI and other coherence protocols.

Our methodology relies on trace driven simulation (by means of the "Trace Factory" environment [8]) and on specific tools for the analysis of coherence overhead [6]. In our evaluation the database server activity is reproduced through a workload made of applications like

the PostgreSQL [28] DBMS, which handles some of the TPC-D [25] queries, and several Unix utilities, which both access file system and interface the various programs running on the system.
The analysis starts from a reference case, and explores different architectural choices for cache, coherence protocols and number of processors. The scheduling algorithm has been varied, considering both a random and a cache affinity policy [17].
Our results show that even in the four-processor case OS effects may not be neglected. In fact, in consequence of migration, AMSD outperforms MESI, while in consequence of migration and of the absence of invalidation misses, most of which are due to kernel activity, PSCR appears to be the optimal solution. The effects on performance become more important in high-end machines (16 or more processors). Indeed, in that case, when adopting miss reduction techniques like the increase of block size, the percentage of bus occupancy due to kernel activity may be quantified in almost 30%. In this case, better speed up may be achieved adopting redesign of kernel data structure. Cache affinity is somewhat useful in reducing migration effect, but it is not effective in every load conditions.

2 Coherence Protocols and Coherence Overhead

The main coherence protocol classes are Write-Update (WU) and Write-Invalidate (WI) [22]. WU protocol updates the remote copies on each write involving a shared copy. Whereas, a WI protocol invalidates remote copies in order to avoid updating them.
In our evaluation, we considered three different coherence protocol: MESI, AMSD and PSCR.
The MESI protocol is the most widely adopted protocol, so it is a reference point for our evaluations. We did not evaluate a pure WU protocol, like Dragon, since it is known from the literature [16] that a pure WI protocol outperforms a pure WU protocol when we adopted a scheduling which permits process migration. For this reason, we considered a selective invalidation protocol, which has a WU strategy for the truly shared data (PSCR). We also considered a protocol specifically designed to treat the data migration (AMSD), since process migration involves also data migration.
MESI is a Write Invalidate MOESI class protocol [21], based on Goodman's Write-Once 4-state protocol [11]. The protocol has four states: *Modified*, when the cache block is the only modified copy with respect to main memory; *Exclusive*, when the cache block holds the only valid copy that is identical to the block in main memory; *Shared*, when the cache block holds a valid copy that is identical to the block in main memory and may be present in other caches; *Invalid*, when the cache block holds no valid information. It is implemented in most of the commercial high-performance microprocessor like AMD series, the PowerPC series, the SUN UltraSparc II, SGI R10000, Intel Pentium series.
AMSD is a protocol using Adaptive Migratory Sharing Detection [3], [18]. Migratory sharing is characterized by the exclusive use of data for a long time interval. Typically, the control over these data migrates from one process to another [12]. The protocol identifies migratory-shared data dynamically in order to reduce the cost of moving them. The implementation is an extension of a common MESI protocol. 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 necessary consequent coherence transaction.
PSCR (Passive Shared Copy Removal) [9] adopts a invalidation scheme for the private data, and a WU scheme for the actively shared data. A cached copy belonging to a process private area is invalidated locally as soon as another processor fetches such block. The selective invalidation mechanism allows PSCR to gain the benefits of an update mechanism in shared bus architectures.
Coherency maintaining involves a number of operations. Some of them are overhead that adds up to the basic bus traffic, which is necessary to access the main memory. For WU protocols, this overhead is made of *write transactions*, while for WI protocols we have to consider *invalidation requests* and *invalidation misses*.
In order to evaluate the impact of OS on performance, we analyzed in detail the sources for the overhead and, therefore, we classified the coherence transactions based on the following scheme. Three different sources may be observed: i) *active sharing* [24], which occurs when the same cached data item is referenced by different processes concurrently running on different processors; ii) *false sharing* [24], which occurs when several processors reference always separately different data items belonging to the same memory block separately; iii) *passive* [16] 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 active sharing generates unavoidable overhead, the other two sharing overhead can be eliminated or decreased.
The coherence overhead is classified by means of an extension of an existing algorithm [14] to the case of passive sharing, finite size caches, and process migration. This algorithm [6] is based on the analysis of access patterns to data.

3 Methodology and Workload

The methodology that we used is based on trace-driven simulation [20], [15], [27] and on the simulation of the three kernel activities that most affect performance: *system calls*, *process scheduling*, and *virtual-to-physical address translation*. Memory references include both user and kernel references, and they are produced "on-demand" [8].

The approach used is to produce a *source* trace (a sequence of memory references, system-call positions and synchronization events in case of multithreaded programs) by means of a tracing tool (a modified version of Tangolite [10]). Trace Factory then models the execution of complex workloads by combining multiple source traces and simulating system calls (which could also involve I/O activity), process scheduling and virtual-to-physical address translation. Finally, Trace Factory produces the references (*target* trace) furnished as input to a memory-hierarchy simulator [15]. Trace Factory generates references according to an on-demand policy: it produces a new reference when the simulator requests one, so that the timing behavior imposed by the memory subsystem conditions the reference production [7]. 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 plus synchronization events for multiprocessor applications. Virtual-to-physical address translation is modeled by mapping sequential virtual pages into non-sequential physical pages. An evaluation of this methodology has been carried out in [8].

Table 1. Statistics of source traces for some Unix commands (64-byte block size 10,000,000 references per process).

APPLICATION	DISTINCT BLOCKS	CODE (%)	DATA (%)	
			READ	WRITE
awk (beg)	4963	76.76	14.76	8.48
awk (mid)	3832	76.59	14.48	8.93
Cp	2615	77.53	13.87	8.60
Gzip	3518	82.84	14.88	2.28
Rm	1314	86.39	11.51	2.10
Ls –aR	2911	80.62	13.84	5.54
Ls -ltR (beg)	2798	78.77	14.58	6.65
Ls -ltR (mid)	2436	78.42	14.07	7.51

Table 2. Statistics of multiprocess application source trace (PostgreSQL TPC-D queries, 180,000,000 references) and target trace (DB, 280,000,000 references) in case of 64-byte block size.

WORK-LOAD	NO.OF PRO-CESSES	DISTINCT BLOCKS	CODE (%)	DATA (%)		SHARED BLOCKS	SHARED DATA (%)	
				READ	WRITE		ACCESS	WRITE
TPC-D	18	139906	72.82	17.07	10.12	7806	2.27	0.67
DB	26	179862	74.59	16.26	9.15	7806	1.76	0.53

The workload considered in our evaluation includes DB activity reproduced by means of an SQL server, namely PostgreSQL [28], which handles TPC-D [25] queries and some Unix utilities. These utilities can both access file system and interface the various programs running on the system.

PostgreSQL is a public domain DBMS, which relies on server-client paradigm. It consists of a front-end process that accepts SQL queries, and a back-end that forks processes, which manage the queries. A description of PostgreSQL memory and synchronization management scheme can be found in [26]. TPC-D is a benchmark for DSS developed by the Transaction Processing Performance Council. It simulates an application for a wholesale supplier that manages sells and distributes a product worldwide. The data is organized in several tables and TPC-D includes 17 read-only queries and 2 update queries. Following TPC-D specifications, we populate via the *dbgen* program the database with a scale factor of 0.1. Most of the queries are complex, and perform different operations on database tables.

For completing our workload, we considered some glue-processes that can be generated by shell scripts. To this end, Unix utilities (`ls`, `awk`, `cp`, `gzip`, and `rm`) have been added to the workload. In a typical situation, various requests may be running, requiring the support of different system commands and other applications. To take into account that requests may be using the same program at different times, we traced some commands in shifted execution sections: initial (beg) and middle (mid). Table 1 contains some statistics of the uniprocess traces used to generate the combined workloads. Table 2 contains statistics of multiprocess source trace and target trace i.e. the workload used in the simulation (DB).

4 Results

Performance of a machine executing a workload depends on the execution time of the programs constituting the workload. Execution time of a program depends on the processor waiting time that, in turn, depends on the time necessary to complete a bus operation and on the waiting time before obtaining bus access (bus latency). The bus latency depends on bus traffic and the kind of traffic. The time for completing a bus operation depends on the cost of *read block* transactions (Table 3), essentially due to the misses. Miss and coherence transactions affect the bus latency. Therefore, as performance metrics to compare the several system configurations, we considered the "miss rate", which includes the *invalidation miss* rate and the classical misses (sum of cold and replacement misses), and the "number of coherence transaction per 100 memory references", which includes either *write-transactions* or *invalidations* depending on the protocol. The rest of traffic is due *update transactions*. Update

transactions are only a negligible part of bus-traffic (lower than 8% of *read-block* transactions) and thus they do not influence greatly our analysis.

In terms of global performance we used the following single figure, which expresses the computational power delivered by the machine: the Global System Power (GSP) as done in previous studies [2], [15], [8]. The GSP represents the number of the processors of an ideal machine that does not have delay in accessing memory:

$$GSP = \Sigma U_{cpu}$$

where

$$U_{cpu} = (T_{cpu}-T_{delay})/T_{cpu}$$

T_{cpu} is the time needed to execute the workload, and T_{delay} is the total CPU delay time due to waiting for memory operation completion. We believe that this metric provide a better measurement than execution time, since we do not execute a single program, in our simulations, but a combination of portions of programs. In this condition, GSP gives the necessary comparability when the performance evaluation requires varying the number of processors and other system parameters.

The simulated system consists of N processors, which are interconnected to a single 128-bit shared bus for accessing shared memory. The following coherence schemes have been considered: AMSD, MESI, and PSCR. As for the number of processors, two configurations have been considered: a basic machine with 4 processors and a high-performance machine with 16 processors. The scheduling policy can be: random and cache-affinity; scheduler time slice is 200,000 references. Cache size has been varied between 512K and 2M, while for block size we tried 64 bytes and 128 bytes. The simulated processors are MIPS-R10000-like; paging relays on 4-KByte-page size; the bus logic supports transaction splitting, and processor-consistency memory model [5]. The simulation time analyzed corresponds to 280,000,000 references. The base case study timings and parameter values for the simulator are summarized in Table 3.

Table 3. Numerical values of some input parameters for the multiprocessor simulator (times are in clock cycles)

Class	Parameter	Timings
CPU	Read cycle	2
	Write cycle	2
Cache	Cache size (Bytes)	512K, 1M, 2M
	Block size (Bytes)	64, 128
	Associativity (Number of Ways)	1, 2, 4
Bus	Write transaction (PSCR)	5
	Write for invalidate transaction (AMSD, MESI)	5
	Invalidate transaction (AMSD)	5
	Memory-to-cache read-block transaction	72 (block size 64 bytes), 80 (block size 128 bytes)
	Cache-to-cache read-block transaction	16 (block size 64 bytes), 24 (block size 128 bytes)
	Update-block transaction	10 (block size 64 bytes), 18 (block size 128 bytes)

In Figures 1 and 2 we analyze the sources of overhead in the DB workload. In our reference case, we considered a 4-processor machine with 128-bit bus, 64-byte block size and we varied cache size (from 512K to 2M byte) and cache associativity (1, 2, 4).

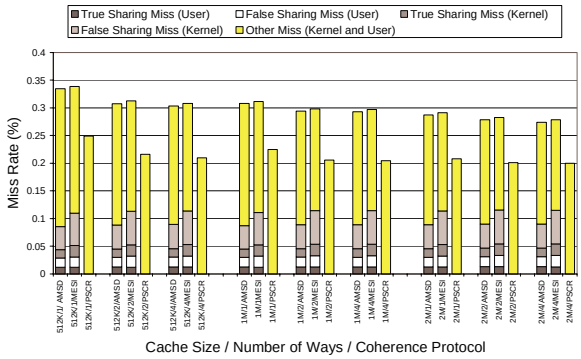

Figure 1. Breakdown of miss rate versus cache size (512 K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Miss Rate assumes 4 processors, a random scheduler and 64-byte block. Other Miss includes cold, capacity and replacement miss. Invalidation misses (i.e. the sum of false sharing and true sharing miss) are absent in PSCR. Kernel contribution to invalidation misses for AMSD and MESI is dominant, and it is of false sharing type. total miss are almost the same in the two protocols..

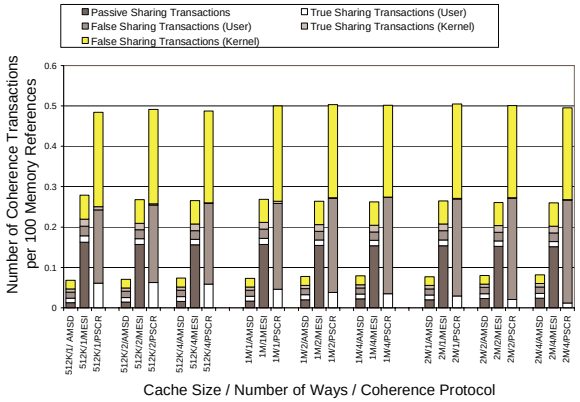

Figure 2. Number of coherence transactions versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR) Coherence transactions are write-for-invalidate transactions in MESI, write-for-invalidate and invalidate transactions in AMSD, write transactions in PSCR. Data assume 4 processors, 64-byte block size and a random scheduler. As we increase cache size and associativity, we have more data sharing, except for MESI. In AMSD, the the invalidation strategy in case of migratory data causes a reduction of coherence transactions.

The results of our simulations for the three protocols, AMSD, MESI, and PSCR, show the contribution of each kind of sharing both to the Miss-Rate and the Coherence-Transaction Rate. We obviously differentiated between kernel and user overhead. In this reference case, we observe a great amount of cold/conflict/replacement misses (*other misses*).

From Figures 1 and 2, we can see that, the unavoidable overhead due to *true sharing* weighs differently depending on the protocol class. All the three

protocols have to face out with this overhead either as invalidation miss, as is the case for AMSD and MESI, which are WI class protocols, or as write-update traffic as is the case for PSCR which is a WU class protocol. Again from Figures 1 and 2, we wish now to concentrate on the unnecessary part of the coherence overhead.

False sharing is a great source of overhead for PSCR, moderately for MESI and not so much for AMSD. On the other hand PSCR avoids completely the *passive sharing*, whilst MESI suffers greatly from it. AMSD can cope with both *false* and *passive sharing* but only in a certain extent, thus we may expect that it is performing better than MESI.

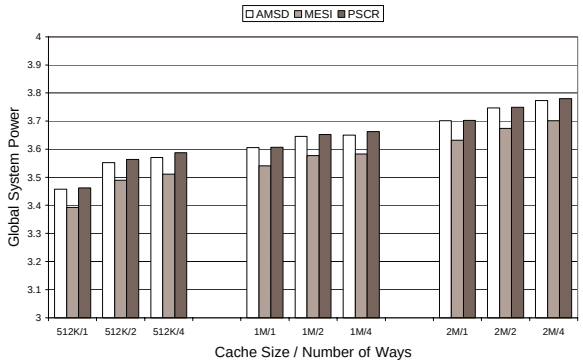

Figure 3. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 4 processors, 64-byte block size and a random scheduler. PSCR presents the highest GSP, whilst MESI the lowest.

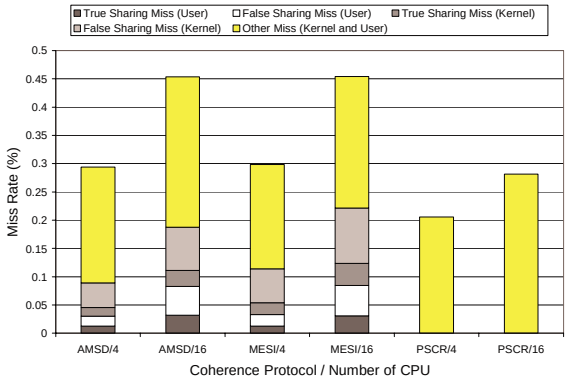

Figure 4. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume, a random scheduler, 64-byte block, 1M-cache size two-way set associative. The higher number of processor causes more coherence misses (false plus true sharing) and more 'other misses'. The differences among protocols are clearer than in the 4-processor case.

The cost of misses is dominating the performance and indeed we show in Figure 3 that PSCR is able to achieve the best performance compared with the other protocols. The reason is that what PSCR looses in terms of extra coherence traffic, is then gained as saved misses. AMSD outperforms MESI. This is basically due to the reduction of *passive sharing* overhead (Figure 2) as miss rate is the same as MESI (Figure 1). In conclusion, the main difference between AMSD and MESI rely, in this case, on *passive sharing* overhead, which is a OS effect, while between PSCR and the other two protocols on the invalidation miss. From Figure 3, we can see that the kernel contribution to these miss is dominant, and particular the *false sharing* contribution.

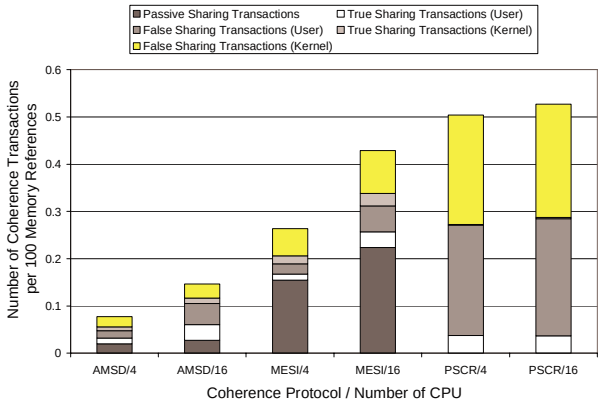

Figure 5. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4, 16). Data assume 64-byte block size, 1M-cache size two-way set associative and a random scheduler. There is an increment in the sharing overhead in all of its components. This increment is more evident in the WI class protocol, also because there is more passive sharing overhead.

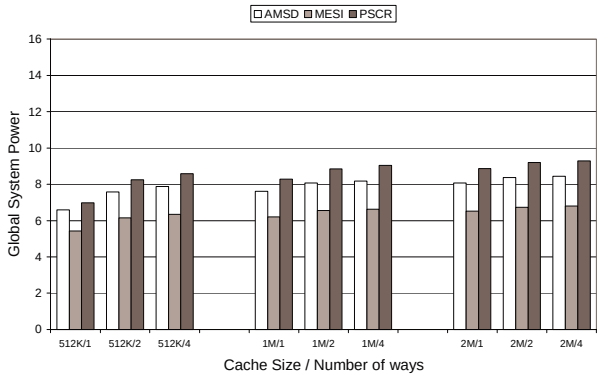

Figure 6. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 64-byte block size and a random scheduler.

However, in this preliminary analysis the performance differences among protocol are small. This is due to the not so high bus utilization (below 40% in every condition), and consequently, the actual cost of transaction is not so high due to the lower contention.

Thus, we considered a 'high-end' 16-processor configuration. This still represents a relatively economic solution to enhance the performance. Since in this case the bus contention is higher, we found a more clear difference among the various protocols.

In the following, for the sake of clearness, we assume a 1-Mbyte cache size, a 64-byte block size, and 2-ways. In Figures 4 and 5, we compare the miss rate and coherence transactions for the three protocols and for the reference case and the 16-processor configuration. When switching to 16 processors, the 'other miss' contribution increases for all protocols (Figure 4). This is mainly due the higher number of compulsory misses that we have in a machine with more processors. In AMSD and MESI, the invalidation misses (both user and kernel) are definitely higher, again for the higher probability of sharing data due the increased number of processors. The combined effect is a stronger difference in the behavior of the two WI protocols and PSCR.

The two WI protocols also increase noticeably the coherence transactions (Figure 5). In PSCR this increase is very limited. The different increase of WI protocols is mainly due to a passive sharing increase in the high-end machine. This translates in a significant increase of processing power (GSP) when we adopt PSCR – more than 10% against the other protocols (Figure 6).

Finally, we applied two important optimizations to reduce the classical misses ('other miss'): i) an increase in the block size, in order to better exploit the spatial locality and ii) the use of affinity scheduling [17].

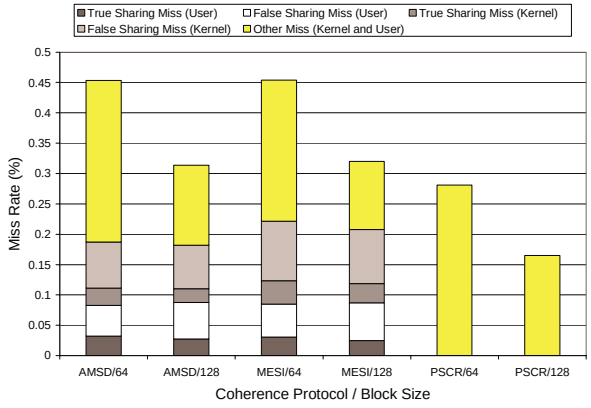

Figure 7. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume 16 processors, a random scheduler, 1M-cache size, and two ways. There is a reduction r in the other miss component, and a little reduction in the invalidation miss component.

The increase of block size may also produce an increase of false sharing. In our experiments (Figure 7 and 8), when switching from 64 to 128 bytes, we observe a reduction of 'other miss' component' and a little reduction of coherence traffic and invalidation miss rate.

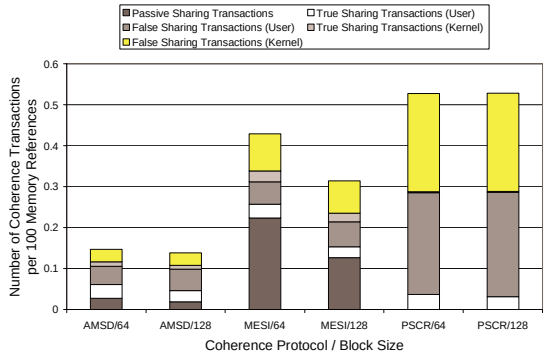

Figure 8. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and block size (64, 128 byte). Data assume 16 processors, 1M-cache size, two-way set associative, and a random scheduler. By increasing block size, both passive sharing and true sharing overhead decrease.

This further advantages PSCR in respect of the other two considered protocols, as shown in Figure 9. From Figure 7 and 8, and considering the cost of transactions (Table 3), we can estimate in almost a 30% bus occupancy due to the kernel activity for both MESI and AMSD, and of this percentage, at least 80% is due to *false sharing* transaction. We can't increase block size, because the increased cost of read-block transaction is not compensated by the reduction of miss (graph not shown), so to achieve better performance over this point, may be worthless to act on kernel data structure to reduce *false sharing* overhead.

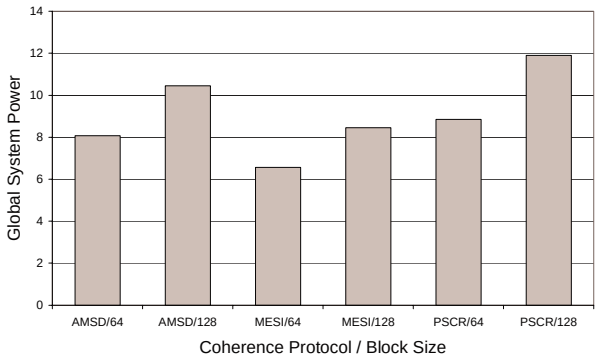

Figure 9. Global System Power versus block size size (64, 128 bytes) and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 1M byte cache size, two ways, and a random scheduler.

In other experiments not reported here, we used a cache-affinity scheduling algorithm. We have again a certain reduction of the classical misses and a slight reduction of coherence related operations. All the three protocol take a little advantage from this, and PSCR continues to deliver the best performance. However, as shown in [9], cache affinity is not effective in every load conditions.

6 Conclusions

We evaluated operating system effects on the memory hierarchy of a SMP multiprocessor, by considering several different choices that could improve the overall performance of the system. We considered different architectures based on MESI coherence protocol (a pure WI protocol, widely used in high performance processors), AMSD (a WI protocol designed to reduce effects of *data* migrations) and PSCR (a coherence protocol using an hybrid strategy: WU for shared data and WI for private data, designed to reduce the effect of *process* migration). The Database workload was setup using the PostgreSQL DBMS executing queries of the TPC-D benchmark and typical Unix shell commands.

Our results show that even in the four-processor case OS effects may not be neglected. In fact, in consequence of migration, AMSD outperforms MESI, while in consequence of migration and of the absence of invalidation misses, most of which are due to kernel activity, PSCR outperforms all the other protocols. The effects on performance become more important in high-end machines (16 or more processors). Indeed, in that case, when adopting miss reduction techniques like the increase of block size, the percentage of bus occupancy due to kernel activity may be quantified in almost 30%. In this case, better speed up may be achieved adopting redesign of kernel data structure. Cache affinity is somewhat useful in reducing migration effects, but it is not effective in every load conditions. An evaluation of how these techniques are effective with respect to the most performing protocol, PSCR, will be the object of future work.

References

[1] A. Agarwal and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under Mach". *Proceedings ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Santa Fe, NM, pp. 215-225, May 1998.

[2] J. K. Archibald and J. L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. on Comp. Systems*, vol. 4, pp. 273-298, Apr. 1986.

[3] A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data," *Proc. 20th International Symposium on Computer Architecture*, San Diego, California, pp. 98-108, May 1993.

[4] S. J. Eggers, T. E. Jeremiassen, "Eliminating False Sharing", *Proc. 1991 International Conference on Parallel Processing, Aug.1991, pp. I:377-381*.

[5] K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors", in Proceedings of the *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, California, pp. 245-357, Apr. 1991.

[6] P. Foglia, *"An Algorithm for the Classification of Coherence Related Overhead in Shared-Bus Shared-Memory Multiprocessors*", IEEE TCCA Newsletter, January 2001.

[7] R. Giorgi, C. Prete, G. Prina, L. Ricciardi, "A Hybrid Approach to Trace Generation for Performance Evaluation of Shared-Bus Multiprocessors". In *Proceedings 22nd EuroMicro International Conference*, Prague, pp. 207-241, Sept. 1996.

[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*, 5(4), pp. 54-68, Oct-Dec 1997.

[9] R. Giorgi and 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". In *Proceedings ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 146-157, May 1993.

[11] J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *In Proceedings of the. 10th International Symposium on Computer Architecture*, Stockholm, Sweden, pp. 124-131, June 1983.

[12] A. Gupta and W.-D. Weber, "Cache Invalidation Patterns in Shared-Memory Multiprocessors," *IEEE Trans. Computers*, vol. 41, no. 7, pp. 794-810, July 1992.

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

[14] R. L. Hyde and B. D. Fleisch, "An Analysis of Degenerate Sharing and False Coherence". *Journal of Parallel and Distributed Computing*, vol. 34(2), pp. 183-195, May 1996.

[15] 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 System*, vol. 6 (9), pp. 915-929, September 1995

[16] 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.

[17] M. S. Squillante and D. E. Lazowska, "Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling". *IEEE Transactions on Parallel and Distributed System*, vol. 4 (2), pp. 131-143, February 1993.

[18] P. Stenstrom, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing". *In Proceedings of the 20th Annual International Symposium on Computer Architecture*. San Diego, CA, May 1993.

[19] P. Stenstrom, E. Hagersten, D. J. Li Margaret Martonosi and M. Venugopal, "Trends in Shared Memory Multiprocessing ", *IEEE Computer*, Vol. 30, no. 12, pp. 44-50, Dec. 1997.

[20] 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.

[21] P. Sweazey and A. J. Smith, "A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus". *In Proceedings of the 13th International*

Symposium on Computer Architecture, pp. 414-423, June 1986.

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

[23] 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, pp. 61-66, Dec. 1994.

[24] J. Torrellas, M. S. Lam, and J.L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches". *IEEE Transactions on Computer*, vol. 43, n. 6, pp. 651-663, June 1994.

[25] Transaction Processing Performance Council, "TPC Benchmark D (Decision Support) Standard Specification". Dec 1995.

[26] P. Trancoso, J. L. Larriba-Pey, Z. Zhang, and J. Torrellas, "The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors". *In Proceedings of the 3rd International Symposium on High Performance Computer Architecture*, pp. 250-260, Los Alamitos, CA, Feb 1997.

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

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

[29] L. Barroso and K. Gharachorloo and F. Bugnion, "Memory System Characterization of Commercial Workloads", In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 3-14, Barcellona, Spain, June 1998.

[30] P. Ranganathan K. Gharachorloo, S. V. Adve and L. Barroso, "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors", In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 307-318, San Jose, California, 1998

[31] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, H. Levy and S. Parekh, "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors", In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 39-51, Barcelona, Spain, June 1998.

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

[33] T. E. Jeremiassen, S. J. Eggers, "Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations", *ACM SIGPLAN Notices, 30 (8), pp. 179-188, August 1995*.

[34] J. Chapin, S. A. Herrod, M. Rosemblum, A. Gupta, "Memory System Performance of Unix on CC-NUMA Multiprocessors", in Proc. ACM SIGMETRICS/PERFORMANCE '95, May 1995.

[35] J. Torrellas, A. Gupta, J. Hennessy, "Characterizing the caching and Synchronization Performance of a Multiprocessor Operating System", in Proc. Of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1992.

Pierfrancesco Foglia received the degree in Computer Engineering and the Ph.D. degree in Computer Engineering both from the University of Pisa. At present, he is a research assistant in the Department of Information Engineering of the University of Pisa. His research interests include computer architecture, coherence protocols, high performance servers and infrastructure for E-Commerce. He is member of the IEEE Computer Society and ACM.

Cosimo Antonio Prete is full professor of Computer Systems at the Department of Electronics, Computer and Telecommunication Engineering at the University of Pisa, Italy. His research interests include multiprocessor architectures, cache memory, performance evaluation and embedded systems. He has performed research in programming environments for distributed systems, in commit protocols for distributed transactions, in cache memory architecture and in coherence protocols for tightly coupled multiprocessor systems. He has been project manager for the University of Pisa for the Esprit III Tracs project (Flexible Real-Time Environment for Traffic Control Systems, supported by the European Communities) and for the Cache-Sim project (a framework for the modeling and simulation of cache memories in ARM-based systems, supported by VLSI Technology Inc., San Jose, California). He has also acted as an expert on the Open Microprocessor Systems Initiative for the Commission of the European Communities. He earned his undergraduate degree in Electronic Engineering cum laude in 1982 and his PhD from the University of Pisa in 1989. He is a member of IEEE, IEEE Computer Society and ACM.

Roberto Giorgi is currently assistant professor at the Department of Information Engineering, University of Siena, Italy. He was research associate at the Department of Electrical and Computer Engineering, University of Alabama in Huntsville, AL (U.S.A.). He received his MS in electronic engineering, summa cum laude, and his PhD in Computer Engineering, both from the University of Pisa, Italy. His main academic interest is Computer Architecture and in particular multithreaded and multiprocessors systems. He is exploring coherence protocols, compile time optimizations, behavior of user and system code, architectural simulation for improving the performance of a wide range of applications from desktop, to embedded-systems, web-servers, and e-commerce servers. He took part in the ChARM project in cooperation with VLSI Technology Inc., San Jose, California, developing part of the software used for performance evaluation of ARM-processor-based embedded systems with cache memory. He is a member of the IEEE, IEEE Computer Society, and ACM.