Performance Analysis of Parallel Applications Running on SMP

Pierfrancesco Foglia, Cosimo Antonio Prete
Dipartimento di Ingegneria dell'Informazione
Università di Pisa
Pisa, Italy

Roberto Giorgi
Dipartimento di Ingegneria dell'Informazione
Università di Siena
Siena, Italy

Abstract

In this work, by using dynamic analysis techniques, we analyze how a workload can be accelerated in the case of a shared-bus shared-memory multiprocessor. It is well known that, in this kind of systems, the bus is the critical element that can limit the scalability of the machine. Nevertheless, many factors that influence bus utilization have not been yet investigated for this kind of workload, in particular the effects of thread migration. The operating system effects are also considered in our evaluation.

We analyzed a basic four-processor and a high-end sixteen-processor machine, implementing three different coherence protocols (including MESI and another solution from the literature). We show that even in the four-processor case, the overhead induced by the sharing of private data, as a consequence of process migration, namely passive sharing, cannot be neglected. Indeed, the analysis shows that a protocol based on a selective strategy for dealing with private and shared data has a better performance than protocols either relying on the detection of migratory access-pattern or purely using a Write-Invalidate strategy, like MESI.

We varied the architectural parameters to show how passive sharing and other coherence overhead are influenced by different cache choices. Then, we considered the sixteen-processor case, where the effects on performance are more evident.

We also end up that performance can take advantage of large caches and cache affinity scheduling. However, even with affinity scheduling, a selective protocol delivers better performance.

1 Introduction

Symmetric Multi Processor (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, E-business, Web-servers, and DSS workloads [19]. In these system 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 and local operations that produce global actions, 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, which is necessary to access the main memory [9].

Coherence overhead can be further classified in the following categories: *True Sharing* [24] overhead is the coherence overhead originated by processors updating the same shared word. In this condition, the system needs to provide each reading processor with the word most current value and the consequent coherence actions are unavoidable. *False Sharing* overhead [4], [24] is the overhead caused by multiple processors accessing different words within the same cache block. *Passive Sharing* [23], [16], [9] or *process-migration* [1] *sharing* overhead is the coherence overhead generated by private data as a consequence of process migration. These types of overhead may have a negative effect on the performance has shown in the literature [4], [22], [24], [9].

The performance analysis of the multiprocessor memory-subsystem has to account for that overhead. In this phase we can use dynamic analysis techniques. From static analysis we can detect False Sharing and adopt the necessary optimizations [24], but we cannot conclude anything about passive sharing, since the migration of private data is due to the process migration, that is a run-time event. We would like also to quantify the entity of false and true sharing, which again depends on run-time conditions and are highly influenced by architectural parameters.

As for possible dynamic analysis techniques: we observe that we cannot use the typical counters that are already present in many recent processors, since they provide information regarding private caches. Hardware tools for analyzing shared data are yet to come [19], so that this type of analysis should be performed through execution driven, or trace-driven simulation, that includes specific tools for analyzing the sharing behavior.

In this paper, we utilize trace driven simulation (by means of the "Trace Factory" environment [8]) and specific tools for the analysis of coherence overhead [6], to analyze memory hierarchy behavior and justify performance issues of a DSS (Decision Support System) workload in a shared-bus shared-memory multiprocessor server. In our evaluation the DSS 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 in these systems larger caches and cache affinity improve the performance. Anyway, a coherence protocol, namely PSCR [16], which adopts a selective invalidation strategy based on the type of data (private or shared) achieves better performance than a pure Write-Invalidate protocols (like MESI) or other protocol specifically designed to deal with the migration of data (rather than process migration), like the protocol of Stenstrom and Cox [18], [3], in the following tagged as AMSD (Adaptive Migratory Sharing Detection) protocol.

2 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 simulations, we implemented three different coherence protocol: MESI, AMSD and PSCR [9]. 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 system 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 at least in one other cache; *Invalid*, when the cache block holds no valid information. It is implemented in most of the commercial high-performance microprocessor like AMD K5 and K6, the PowerPC series, the SUN UltraSparc II, SGI R10000, Intel Pentium, Pentium Pro, Pentium II, III and IA-64.

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.

PSCR (Passive Shared Copy Removal) [9] adopts a selective 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 the same block. The selective invalidation mechanism allows PSCR to gain the benefits of an update mechanism in shared bus architectures.

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 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). Moreover, we considered a protocol specifically designed to treat the data migration (AMSD), since process migration involves also data migration.

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 the *invalidation requests* and the *invalidation misses*.

Three different sources of sharing overhead 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* [22][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.

In order to classify the coherence overhead generated by the applications running on multiprocessors, we have extended an existing algorithm [14] to the case of passive sharing. We also improved the algorithm in order to manage the case of finite size caches, and the case of process migration. This algorithm [6] is based on the analysis of access patterns to shared data. Our pattern sensitive classification extends previous definitions including sharing patterns on private data. This kind of pattern is a non-obvious consequence of the process migration.

3 Methodology and Workload Description

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 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 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].

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

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 multithreaded 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].

We considered a DSS workload as a benchmark for our analysis, as a generic possible case of real workload for our performance evaluation and example of use of dynamic analysis. The DSS activity is 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 database, 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 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. Most of the queries are complex, and perform different operations on database tables. We traced the read-only queries; some queries are considered more than once in order to reach a high number of processes in the system, but their execution is time-shifted.

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 the statistics of the multiprocess source trace and target trace i.e. the workload used in the simulation (DSS).

Table 2. Statistics of multiprocess application source trace (PostgreSQL TPC-D queries, 180,000,000 references) and target trace (DSS, 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
DSS	26	179862	74.59	16.26	9.15	7806	1.76	0.53

4 Results

Our first goal is to show the results from dynamic analysis related to our framework for evaluating the performance of the DSS workload on the shared-bus multiprocessor.

As performance metrics to compare the several outputs, we considered the “miss rate”, which includes the *invalidation miss* rate, and the “number of coherence transaction per 100 memory references”, which includes either *write-transactions* or *invalidations* depending on the chosen protocol. The rest of traffic is due to classical misses (sum of cold and replacement misses) and *update transactions*. Update transactions are only a negligible part of bus-traffic (lower than 8% of *read-block* transactions in our simulations) and thus they do not influence greatly our analysis. Note that *read-block* transactions are essentially due to the misses.

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, 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.

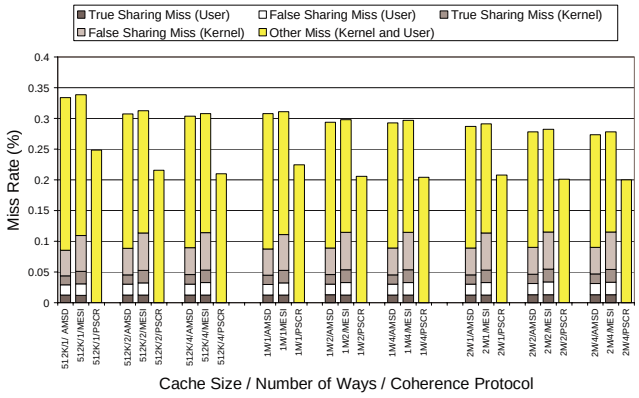

Figure 1. Breakdown of miss rate versus cache size (512 K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Miss Rate assumes 4 processors, a random scheduler and 64-byte block. Other Miss includes cold miss, capacity miss and replacement miss. Invalidation misses (i.e. the sum of false sharing and true sharing miss) are absent in PSCR, due to its write-update behavior; they are lower in AMSD rather than in MESI; however, due to the other miss contribution, total miss are almost the same in the two protocols. Miss rate decreases for all the protocol with cache size and associativity.

In Figures 1 and 2 we analyze the sources of overhead in the DSS 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). 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 also differentiated between kernel and user overhead. In this reference case, we observe a great amount of cold/conflict/replacement misses (*other misses*). However, our analysis concentrate on how to reduce coherence related overhead rather than classical misses.

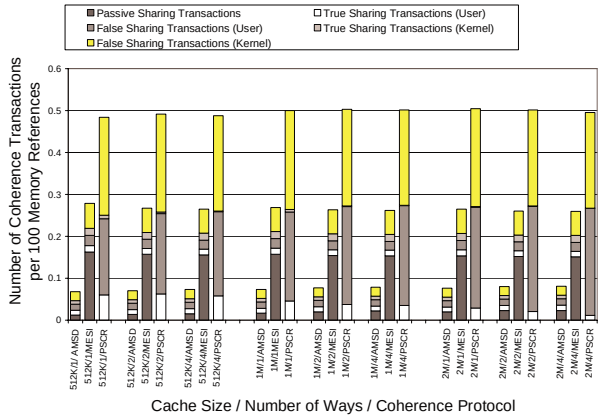

Figure 2. Number of coherence transactions versus cache size (512K , 1M, 2M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR) Coherence transactions are write-for-invalidate transactions in MESI, write-for-invalidate and invalidate transactions in AMSD, write transactions in 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 MESI, the behavior on private data causes a reduction of coherence transactions.

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 can avoid completely the *passive sharing*, since is designed for that, 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.

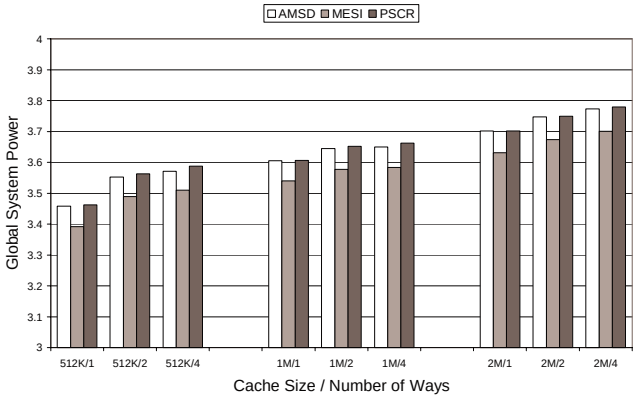

Figure 3. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Global System Power is the sum of processor utilization. Data assume 4 processors, 64-byte block size and a random scheduler. PSCR presents the highest GSP, whilst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

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 miss saved.

However, in this preliminary analysis the performance differences among protocol are small. This is due to the not so high bus utilization in the case of four processors, and consequently, the dynamic 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 out 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.

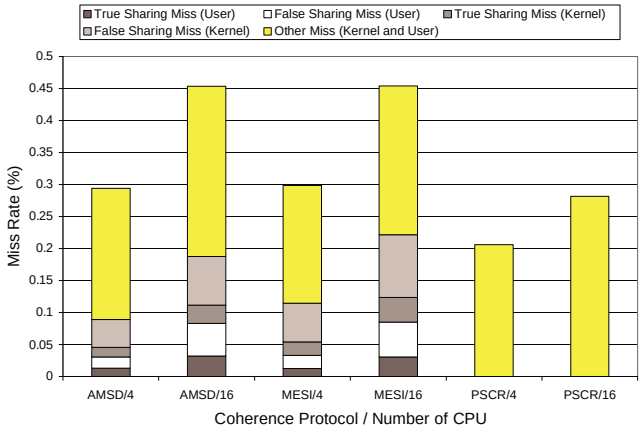

Figure 4. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume, a random scheduler 64-byte block, 1M-cache size two-way set associative. The higher number of processor causes more coherence overhead (false plus true sharing) and more 'other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

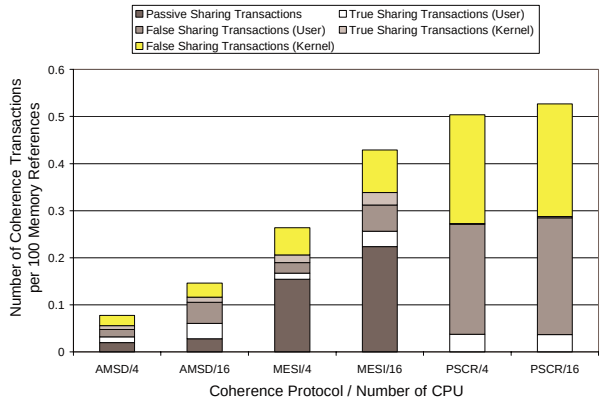

Figure 5. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4, 16). Data assume 64-byte block size, 1M-cache size two-way set associative and a random scheduler. There is an increment in the sharing overhead in all of its components. This increment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

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 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 is mainly due to a passive sharing increase in the high-end machine. This translates in a significant increase of performance when we adopt PSCR – more than 10% against the other protocols (Figure 6).

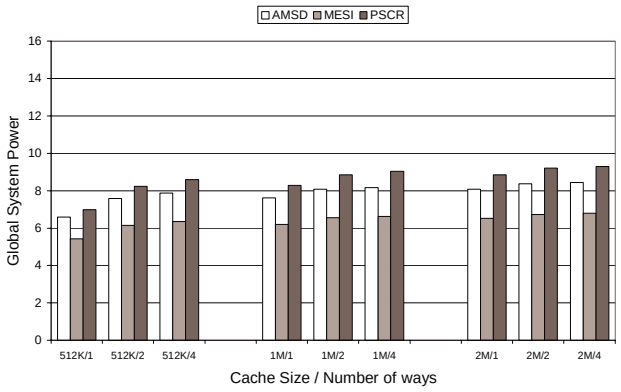

Figure 6. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 64-byte block size and a random scheduler.

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].

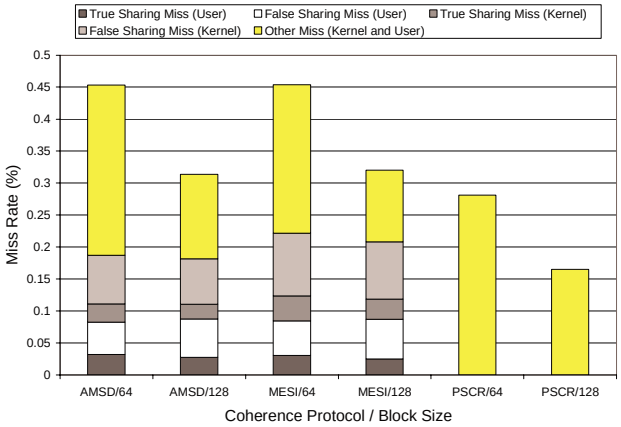

Figure 7. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume 16 processors, a random scheduler, 1M-cache size, and two ways. There is a reduction either in the other miss component, or in the invalidation miss component.

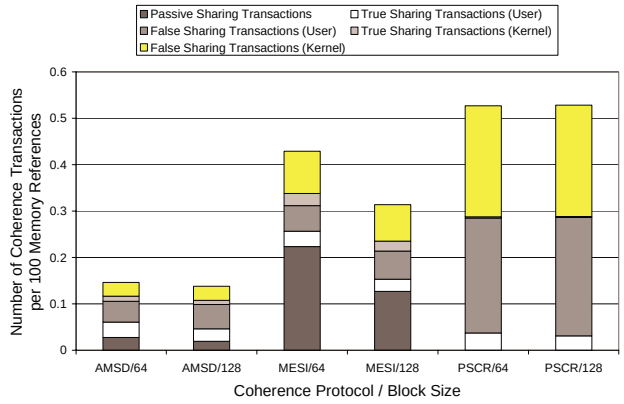

Figure 8. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and block size (64, 128 byte). Data assume 16 processors, 1M-cache size, two-way set associative, and a random scheduler. By increasing block size, both passive sharing and true sharing overhead decrease.

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 noticeable reduction of 'other miss' component' and a reduction of coherence traffic and miss rate. This further advantages PSCR in respect of the other two considered protocols.

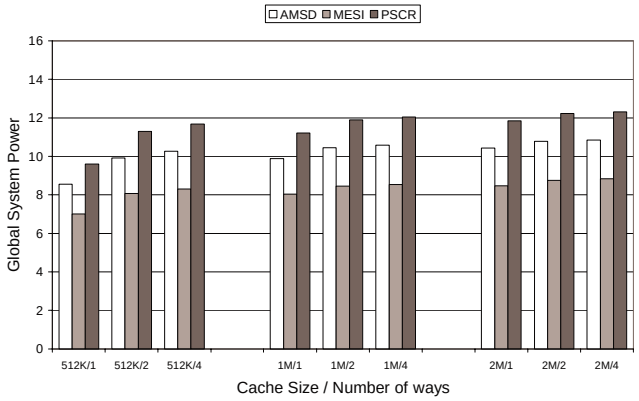

Figure 9. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 128-byte block size, and a random scheduler

When we use a cache-affinity scheduling algorithm (Figure 10 and 11), 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 (Figure 12).

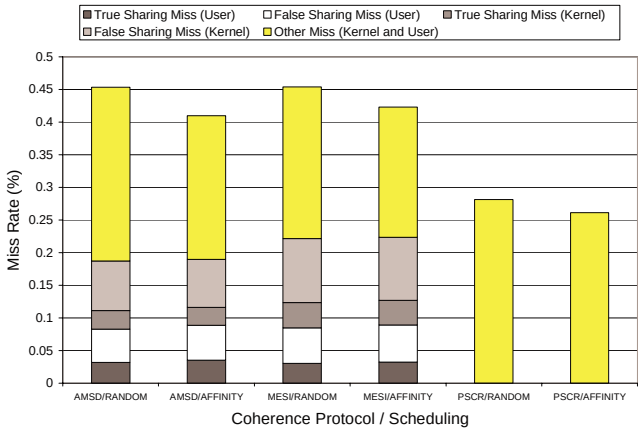

Figure 10. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and scheduling algorithm (Random, Affinity). Data assume 16 processors, 1M-cache size two-way set associative and 64 byte block. The affinity scheduling mainly reduces the other miss rate for all the protocols.

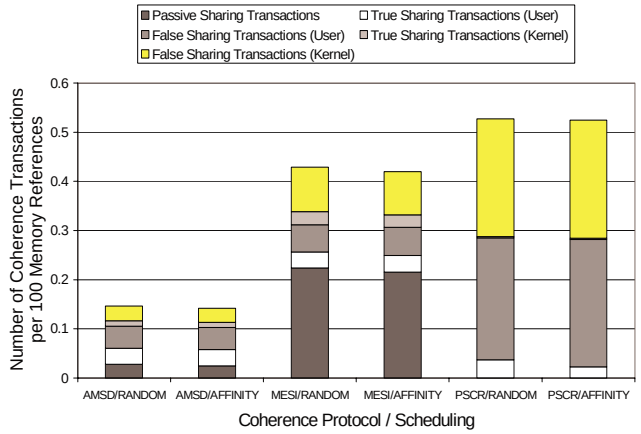

Figure 11. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and scheduling algorithm (Random, Affinity). Data assume 16 processors, 1M-cache size two-way set associative, and 64 byte cache block size. There is a slight reduction in the coherence overhead. This is more evident in WI protocol, due to the reduction of coherence transactions due to passive sharing.

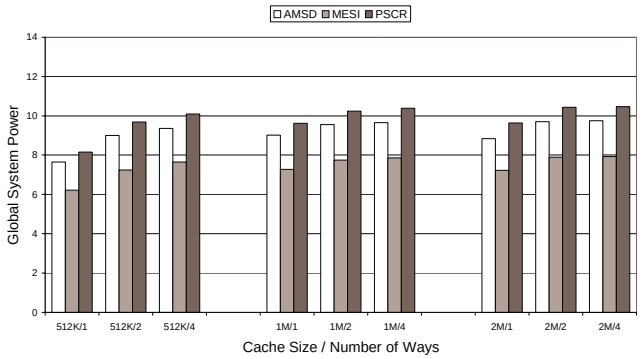

Figure 12. Global System Power versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 64-byte block size and a cache affinity scheduler.

6 Conclusions

Dynamic analysis is an indispensable tool to compare and analyze different architectural options based on shared-bus shared-memory or symmetric multiprocessors (SMP).

In this work, by using dynamic analysis we compared the performance of three 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 representative chosen benchmark has been a DSS workload. This workload has been setup by using the PostgreSQL DBMS and by tracing the execution of queries extracted from the TPC-D benchmark and typical Unix shell commands.

Through trace-driven simulation and coherence overhead analysis, we discovered the major reasons why PSCR outperforms the other protocols. The contribution of misses resulted more penalizing for the performance (those misses are higher in the WI protocols, due to the invalidation misses), when we eliminate the contribution of passive sharing as PSCR does.

Our results show that PSCR outperforms the other protocols in all our test cases. The gain of PSCR becomes more important in high-end machines (16 or more processors). Indeed, in that case the miss reduction techniques like affinity or the increase of block size end up in further advantage for PSCR. This advantage could be quantified in a 10% in the 16-processor case relatively to the other evaluated protocols.

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, September 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 Pablishers, 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 Optimezed 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*, 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.