Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 65 (2005) 289–306
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Reducing coherence overhead and boosting performance of high-end SMP multiprocessors running a DSS workload

Pierfrancesco Foglia[a], Roberto Giorgi[b], Cosimo Antonio Prete[a,*]

[a] *Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Via Diotisalvi 2, 56126 Pisa, Italy*
[b] *Dipartimento di Ingegneria dell'Informazione, Università di Siena, Via Roma 56, 53100 Siena, Italy*

Received 19 February 2003; received in revised form 4 June 2004
Available online 17 January 2005

Abstract

In this work, we characterized the memory performance—and in particular the impact of coherence overhead and process migration—of a shared-bus shared-memory multiprocessor running a DSS workload.

When the number of processors is increased in order to achieve higher computational power, the bus becomes a major bottleneck of such architecture. We evaluated solutions that can greatly reduce that bottleneck. An area where this kind of optimization is important regards data base systems. For this reason, we considered a DSS workload and we setup the experiments following TPC-D specifications on the PostgreSQL DBMS in order to explore different optimizations on same kind of workloads as evaluated in the literature.

In this scenario, we compare possible solutions to boost performance and we show the impact of process migration on coherence overhead.

We found that the consequences of coherence overhead and process migration on performance are very important in machines with 16 or more processors. In this case, even little sharing, as in DSS applications, can become crucial for system performance.

Another important result of our analysis regards the interaction between the coherence protocol and the scheduler. The basic cache affinity scheduling is useful in reducing migration, but it is not effective in every load condition. Specific coherence protocols can help reduce the effects of process migration, especially in situations when the scheduler cannot apply the affinity requirement. In these conditions, the use of a write-update protocol with a selective invalidation strategy for private data improves performance (and scalability) of about 20% with respect to a classical MESI-based solution. This advantage is about 50% in the case of high cache-to-cache transfer.

Keywords: Shared-bus multiprocessors; Cache memory; Coherence protocol; Sharing analysis; Sharing overhead; Scalability

1. Introduction

Shared-bus shared-memory multiprocessors (basically *Symmetric Multi Processor*, or SMP, architectures) are a simple solution to speed up complex workloads demanding for high performance like, just to give some examples, data bases, file servers and application servers [35]. Moreover, Internet and Intranet services rely more and more on data base applications and require high-performance systems.

In shared bus architectures, processors access the shared memory through a shared bus. The 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 [22,9]. Cache memories introduce the coherency problem and the need for adopting adequate coherence protocols [38,39]. Coherence protocols generate several bus transactions, thus accounting for a non-negligible overhead in the system (coherence overhead). Coherence overhead may have a negative effect on the performance and different optimizations have been proposed [13,38,43,16]. Optimizations can act at compile

* Corresponding author. Fax: +39 050 2217600.
E-mail addresses: foglia@iet.unipi.it (P. Foglia), giorgi@unisi.it (R. Giorgi), prete@iet.unipi.it (C.A. Prete).

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2004.10.003

time (e.g., by redesigning the layout of shared data structure [24]) and architectural level (through the adoption of adequate coherence protocols [40,16]).

Widely used applications for these kinds of systems include Data Base Management Systems (DBMS) [35]. Important categories in data base applications and systems are decision support system (DSS) and on-line transaction processing systems (OLTP) [35,22,9]. From a multiprocessor design point of view, data base applications fall in the class of *commercial workloads* [20]. In the past, technical workloads were widely used to drive the design of current multiprocessor systems [35,10]. Different studies have shown that commercial workloads exhibit different behavior from technical ones, both in the general case [20] and in the OLTP system case [25]. Therefore, our focus has shifted to the category of commercial workloads.

A few papers considered DSS workloads running on SMPs. The behavior of these workloads has been analyzed for the aspects concerning processor architecture [32,27] and the memory system [46,3,28]. These studies mainly regarded four-processor systems that used a MESI coherence protocol and the multiprocessor architectures considered were either SMP or CC-NUMA. For these architectures, the analysis of coherence overhead in case of DSS workloads showed that, as cache size is increased, coherence traffic also increases, although the coherence overhead remains relatively low. Only some evaluation analyzed more in detail this coherence overhead [46]. Differently from those studies, our analysis allows us to quantitatively show the various types of sharing, show the effects of process migration (needed to achieve load balancing) on the memory subsystem, and highlight the influence of the architectural parameters with a relatively high number of processors. Solutions, such as modifications of coherence protocol, to increase performance of DSS systems—without changing the processor architecture—are particularly interesting and cost effective to implement high-performance SMPs.

In our evaluation, the DSS workload of the system is generated by running all the TPC-D queries [45] (through the PostgreSQL [51] DBMS) and several Unix utilities, which both access the file system and interface the DBMS with system services running concurrently. Our methodology relies on trace-driven simulation, by means of the "Trace Factory" environment [18], and on specific tools for the analysis of coherence overhead [14].

The performance of the memory subsystem—and therefore of the whole system—depends on cache parameters, coherence management techniques, and it is influenced by operating system activities like process migration, cache affinity scheduling, kernel/user code interference and virtual memory mapping. The importance of considering operating system activity has been highlighted in previous works [6,5,42]. Process migration, needed to achieve load balancing in such systems, increases the number of cold/conflict misses and also generates useless coherence overhead, known as *passive sharing* overhead [16]. Hence, our analysis starts from a reference case, and explores different architectural choices for cache, coherence protocol, and number of processors.

The results of our analysis are much more significant in high-end SMPs (16 or more processors), but we started the analysis with a four-processor system in order to compare our results with SMP configurations that are mostly used in current machines [22].

Our results show that, in the case of a four-processor configuration, performance of DSS workload is moderately influenced by cache parameters and the influence of coherence protocol is minimal. In 16-processor configurations, the performance differences due to the adoption of different architectural solutions are significant. Even a low coherence overhead, as observed in the DSS workload case, cannot be neglected. In such case, MESI protocol is not the best choice. Workload changes can nullify the action of affinity-based scheduling algorithms. Architectures based on a write-update protocol with a selective invalidation strategy for private data outperform the ones based on MESI by about 20%. We will also show solutions that are less sensitive to cache-to-cache transfer latency.

The rest of the paper is organized as follows. In Section 2, we discuss architectural parameters and issues related to the coherence overhead (this section could be skipped by the reader already expert in this area). In Section 3, we report the results of studies related to the analysis of workloads or machines similar to ours, differentiating our contribution. In Section 4, we present our experimental setup and methodology. In Section 5, we present and discuss the results of our experiments. In Section 6, we draw the conclusions.

2. Issues affecting coherence overhead

Cache coherency maintaining involves a number of bus operations. Some of them are overhead that adds up to the basic bus traffic (the traffic necessary to access main memory). Three different sources of sharing may be observed: (i) *true sharing* [41,43], which occurs when the same cached data item is referenced by different processes concurrently running on different processors; (ii) *false sharing* [41,43], which occurs when several processors reference a different data item belonging to the same memory block separately; (iii) *passive* [30] 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 types of sharing overhead can be eliminated or decreased.

The main coherence protocol classes are Write-Update (WU) and Write-Invalidate (WI) [38]. WU protocols update the remote copies on each write involving a shared copy. Whereas, WI protocols invalidate remote copies in order to avoid updating them. In our evaluation, we considered MESI

protocol as baseline. Then, we included a selective invalidation protocol, which reduces most effects of process migration (PSCR [16]) and a WI protocol specifically designed to treat the data migration (AMSD [8,34]).

Our implementation of MESI uses the classical MESI protocol states [37] and the following bus transactions: *read-block* (to fetch a block), *read-and-invalidate-block* (to fetch a block and invalidate any copies in other caches), *invalidate* (to invalidate any copy in other caches), and *update-block* (to write back dirty copies, when they need to be replaced). This is mostly similar to the implementation of MESI in Pentium Pro (Pentium II) processor family [33]. The invalidation transaction used to obtain coherency has, as a drawback, the need to reload a certain copy, if a remote processor uses again that copy, thus generating a miss (*Invalidation Miss)*. Therefore, MESI coherence overhead (that is the transactions needed to enforce coherence) is due both to *Invalidate Transactions* and *Invalidation Misses*.

Passive shared copy removal (PSCR) adopts a selective invalidation scheme for private data, and uses a Write-Update scheme for shared data, although it is not a purely Write-Update protocol. A cached copy belonging to a process private area is invalidated locally as soon as another processor fetches the same block [16]. This technique reduces coherence overhead (especially passive sharing overhead), otherwise dominant in a pure WU protocol [30]. Invalidate transactions on the shared-bus are avoided and coherence overhead is due to *Write Transactions* and *Invalidation Misses* caused by local invalidation (*Passive Sharing Misses*).

AMSD [8,34] is designed for Migratory Sharing, which happens when the control over shared data migrates from one process to another running on a different processor. The protocol identifies migratory-shared data dynamically to reduce the cost of moving them. The implementation relies on an extension of a common MESI protocol. Though designed for migratory sharing, AMSD may have some beneficial effects also on passive sharing. AMSD coherence overhead is due to *Invalidate Transactions* and *Invalidation Misses*.

The process scheduling strategy, cache parameters, and the bus features also influence the coherence overhead and, therefore, the multiprocessor performance. The process scheduling guaranties the load balancing among the processors by scheduling a ready process on the first available processor. Even when cache affinity is used, a process may migrate on a different processor rather than on the last used processor, producing at least two effects: (i) some misses when that process restarts on a new processor (context-switch misses); (ii) useless coherence transactions due to passive sharing. These situations may become frequent due to the dynamicity of a workload driven by the user requests, like DSS ones. Process migration influences also access patterns to data and, therefore, the coherency overhead.

All the factors described above have important consequences on the global performance that we shall evaluate in the Section 5.

3. Related work

In this section, we consider a summary of main results of evaluations for DSS and similar workloads on several multiprocessor architectures and operating systems. The research of a realistic evaluation framework for shared memory multiprocessor evaluations [35] motivated many studies that consider benchmarks like TPC-series (including DSS, OLTP, WEB-server benchmarks) representative of commercial workloads [3,4,25,27,28,32,46].

3.1. TPC-D workload evaluations

Trancoso et al. study the memory access patterns of a TPC-D-based DSS workload [46]. The DBMS is Postgres95 running on a simulated four-processor CC-NUMA multiprocessor. For cache block sizes ranging from 4 to 128 bytes and cache capacities from 128 K-bytes to 8 M-bytes, the main results are that both large cache blocks and data pre-fetching help, due to spatial locality of index and sequential queries. Coherence misses can be more than 60% of total misses in queries that uses index scan algorithms for select operations.

Barroso et al. evaluate an Alpha 21164-based SMP memory system through hardware counter measurements and SimOS simulations [3]. Their system was running Digital-UNIX and Oracle-7 DMBS. The authors consider a TPC-B data base (OLTP benchmark), TPC-D (DSS queries), and the Altavista search engine. They found that memory is accounting for 75% of stall time; the workloads spend about 18% in kernel mode in the case of TPC-B, less than 3% in the case of TPC-D. They found that TPC-B workload is more sensitive to outer level caches parameters. In the case of TPC-D and Altavista workloads, the size and latency of the on-chip caches influences mostly the performance. The number of processes per processor, which is usually kept high in order to hide I/O latencies, also significantly influences cache behavior. When using 2-, 4-, 6-, and 8- processor configurations, they found that coherency miss stalls increase linearly with the number of processors. Beyond an 8 M-bytes outer level cache, they observed that true sharing misses limit performance.

Cao et al. examine a TPC-D workload executing on a Pentium-Pro four-processor system, with Windows NT and MS SQL Server [4]. Their goal is to characterize this DSS system on a real machine, in particular, regarding processor parameters, bus utilization, and sharing. Their methodology is based on hardware counters. They found that kernel time is negligible (less than 6%). Major sources of processor stalls are instruction fetch and data miss in outer level caches. They found lower miss rates for data caches in comparison with other studies on TPC-C [3,25]. This is due to the smaller working set of TPC-D compared with TPC-C.

Ranganathan et al. consider both an OLTP workload (modeled after TPC-B [44]) and a DSS workload (query 6 of TPC-D [45]) [32]. Their study is based on trace-driven

(only user-level traces) simulation, where traces are collected on a four-processor AlphaServer4100 running Digital Unix and Oracle 7 DBMS. The simulated system is a CC-NUMA shared-memory multiprocessor with advanced ILP support. Results, on a four-processor system and an ILP configuration with four-way issue, 64-entry instruction window, four outstanding misses, provide already significant benefits for OLTP and DSS workload. Such configurations are even less aggressive than ILP commercial processor like Alpha 21264, HP-PA 8000, MIPS R10000 [49]. The latter processor, used in our evaluation, makes us reasonably safe that this processor architecture is sound for investigation in the memory subsystem.

Another performance analysis of OLTP (TPC-B) and DSS (query 6 of TPC-D) workloads via simulation is presented in [28]. The simulated system is a commercial CC-NUMA multiprocessor, which is constituted by four-processor SMP nodes connected using a scalable coherent interface (SCI)-based coherent interconnects. Coherence protocol is directory-based. Each node is based on a Pentium Pro processor with a 512 K-byte, four-way set associative cache. Results show that TPC-D miss rate is much lower than in the OLTP (TPC-B) experiments and performance is less sensitive to L2 miss latency. They analyze also scalability exhibited by such workloads. The speed-up of the DSS workload is near to the theoretical ones. Results show that scalability strongly depends on miss rate.

Lo et al. [27] analyze the performance of data base workloads running on simultaneous multithreading processors [12]—an architectural technique to hide memory and functional units latencies. The study is based on trace-driven simulation of a four processor AlphaServer4100 and Oracle 7 DBMS as in [3]. They consider both an OLTP workload (modeled after the TPC-B [44] benchmark) and a DSS workload (query 6 of the TPC-D [45] benchmark). Results show that while DBMS workloads have larger memory footprints, there is a substantial data reuse of small working sets.

Most of the conclusions of these evaluations [3,4,27,28,32,46] present analogies with our work, especially in the case of four-processors. In particular, large caches, more associativity, and larger blocks help reduce classical misses produced by large working sets.

Summarizing, these studies considered DSS workloads but they were mostly limited to four-processor systems, did not consider the effects of process migration, and did not correlate the amount of sharing to the performance of the system. As we have more processors, it becomes crucial to characterize further the memory subsystem. In this work, we investigated both 4- and 16- processor configurations, finding that larger caches may have several drawbacks due to coherence overhead. This is mostly related to the use of shared structures like indices and locks in TPC-D workload. In Section 5, we will classify the sources of this overhead and propose solutions to overcome limitations to the performance related to process migration.

3.2. Other workload evaluations related with our experiments

Other papers considered data base workloads and multiprocessor system configuration similar to ours [5,10,20,25,42,48]. However, their main focus deviates from ours.

Many studies compared the behaviors of technical and commercial workloads [20,25], finding several differences. Technical workloads are highly code optimized, the execution time is mostly spent in user mode, rather than in kernel mode, the working set is particularly small compared to commercial-workload or TPC-like benchmarks [10,20,25]. Moreover, technical workloads are usually single-user, whilst commercial workloads are usually multi-user and have significant amount of kernel activity [20]. Keeton et al. notice that commercial applications exhibit high context-switch rate compared to technical ones [25].

Woo et al. find that coherence overhead influence greatly the performance. Their study is based on the SPLASH-2 benchmark and a 32-processor CC-NUMA system [48].

The effects of OS scheduling and page migration policies on performance are considered in [5]. In this case, the system is the Stanford DASH CC-NUMA running multiple parallel applications. The machine has 16 processors organized as four clusters of four processors each one, and it uses a modified version of SGI IRIX. They find that affinity scheduling is particularly useful in large-scale CC-NUMA multiprocessors. Our results on cache affinity scheduling agree with this work. In our case, we also found that affinity scheduling is not so effective when the number of processes is near to the number of processors.

Torrellas et al. [42] found that operating system can slow down software-development applications and commercial workloads by 17–21%. They used a commercial four-processor system running complex parallel workloads (including an Oracle data base). We found similar influences of operating system, especially in the case of four-processor configuration. Anyway, we analyze also other aspects that become critical in different architectural configurations (e.g. cache affinity with a higher number of processors). Other differences are due to our specific DSS workload.

4. Methodology and workload

The methodology that we used is based on trace-driven simulation [19,31,36,47] and on the simulation of the three kernel activities that most affect performance: *system calls*, *process scheduling*, and *virtual-to-physical address translation*.

The approach used is to produce a *source* trace (Table 1)—a sequence of memory references, system-call positions (and synchronization events if needed)—by means of a tracing tool. Trace Factory then models the execution of complex multiprogrammed workloads

Table 1
Statistics of source traces for some Unix commands and daemons and for multiprocess source traces (PostgreSQL, TPC-D queries) both in case of 64-byte block size and 10,000,000 references per process

Application	No. of processes	Distinct blocks	Code (%)	Data (%)		Shared blocks	Shared data (%)	
				Read	Write		Access	Write
awk (beg)	1	4963	76.76	14.76	8.48	N/A	N/A	N/A
awk (mid)	1	3832	76.59	14.48	8.93	N/A	N/A	N/A
cp	1	2615	77.53	13.87	8.60	N/A	N/A	N/A
rm	1	1314	86.39	11.51	2.10	N/A	N/A	N/A
ls -aR	1	2911	80.62	13.84	5.54	N/A	N/A	N/A
telnetd	1	463	82.75	12.96	4.29	N/A	N/A	N/A
crond	1	2464	75.86	16.35	7.79	N/A	N/A	N/A
syslogd	1	2848	80.41	14.96	4.63	N/A	N/A	N/A
TPC-D_{18}	18	139,324	73.05	16.89	10.06	7224	1.58	0.43
TPC-D_{12}	12	93,657	73.04	16.91	10.05	5662	1.55	0.41

Table 2
Statistics of multiprogrammed target traces (DSS_{26}, 260,000,000 references; DSS_{18}, 180,000,000 references) in case of 64-byte block size

Workload	No. of processes	Distinct blocks	Code (%)	Data (%)		Shared blocks	Shared data (%)	
				Read	Write		Access	Write
DSS_{26}	26	179,862	74.59	16.26	9.15	7806	1.76	0.53
DSS_{18}	18	124,268	74.00	16.11	8.89	6242	1.63	0.48

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, Table 2) furnished as input to a memory-hierarchy simulator [31]. Trace Factory generates references according to an on-demand policy: it produces a new reference when simulator requests one, so that the timing behavior imposed by the memory subsystem conditions the reference production [17,18]. Process management is modeled by simulating a scheduler that dynamically assigns a ready process. Virtual-to-physical address translation is modeled by mapping sequential virtual pages into non-sequential physical pages. A careful evaluation of this methodology has been carried out in [18].

The workload considered in our evaluation includes DB activity reproduced by means of an SQL server, namely PostgreSQL [51], which handles the TPC-D [45] queries. We also included Unix utilities that access the file system, interface the various programs running on the system, and reproduce the activity of typical Unix daemons. With the introduction of new data base technology such as materialized views, TPC-D was split into two separate benchmarks, TPC-H and TPC-R [29]. TPC-R differs from TPC-D and TPC-H benchmarks because it uses pre-computed structures (namely materialized views). Therefore, TPC-H will supersede the TPC-D version used in this paper.

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. TPC-D is a benchmark for DSS developed by the Transaction Processing Performance Council [45]. It simulates an application for a wholesale supplier that manages, sells, and distributes a product worldwide. Following TPC-D specifications, we populated the data base via the *dbgen* program, with a scale factor of 0.1. The data are organized in several tables and accessed by 17 read-only queries and 2 update queries. Most of the queries are complex, and perform different operations on data base tables.

DSS queries are long running, typically scanning large amounts of data and in most part read-only. Consequently, as also observed in [46,50], coherence activity and transactions are due to the sharing of DBMS metadata (i.e. data structure that are needed by the DBMS to work rather than to store the data) and particularly data-structures needed to implement software locks. PostgreSQL uses a coarse grain locking mechanism (at the table level instead of record level [51]), which impact performance of updating queries, but DSS workloads involve essentially read-only queries.

In a typical situation, application and management processes can require the support of different system commands and ordinary applications. To this end, Unix utilities (ls, awk, cp, and rm) and daemons (telnetd, syslogd, crond) have been added to the workload. These utilities are important because they model the ‘glue’ activity of the system software. These utilities: (i) do not have shared data and thus they increase the effects of process migration, as discussed in detail in the Section 5; (ii) they may interfere with shared data and code cache-footprint of 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).

In our experiments, we generated two distinct workloads. The first workload (DSS_{26} in Table 2) includes the TPC-D_{18} source trace (Table 1). TPC-D_{18} is a multiprocess source trace taking into account the activity of 18 processes. One process is generated by DBMS back-end execution, whilst the other processes are generated by the concurrent execution of the 17 read-only TPC-D queries. Since we wanted to explore critical situations for the affinity scheduling—when the number of process changes,—we also generated a second workload (DSS_{18} in Table 2) that includes a subset of TPC-D queries (TPC-D_{12}). Table 1 contains some statistics of the uniprocess and multiprocess source traces used to generate the DB experimental workloads (target traces, Table 2). Both source traces related to DBMS activity (TPC-D_{18}, TPC-D_{12}) and the resulting workloads (DSS_{26} e DSS_{18}) present similar characteristics in terms of read, write and shared accesses.

Based on comparison with previous studies [3,4,46], the total number of simulated reference is appropriate to draw conclusions regarding the behavior of the memory system, in presence of load balancing and other operating system activities. Caches were partially warmed up in order to reduce the influence of cold-start misses [31].

5. Results

In this section, we wish to show the memory subsystem performance, for our DSS workload, with a detailed characterization of coherence overhead and process migration problems. To this end, we included results for several values of the most influencing cache-architecture parameters. Finally, we considered critical situations for the affinity scheduling and we analyzed the sensitivity to cache-to-cache latency.

We considered and compared several solutions to achieve a higher scalability for a shared-bus shared-memory multiprocessor. Since we are focusing on the memory subsystem performance, we will use global performance figures like GSP (whose definition is recalled below), and show how this is connected to memory performance penalty factors such as misses and coherence overhead. The latter factors are closely related to the types of sharing generated by the DSS application and system software, and in particular by possible failures of affinity scheduling algorithm.

5.1. Design space of our system

The simulated system consists of N processors, which are interconnected to a 128-bit shared bus for accessing shared memory. The following coherence schemes have been considered: AMSD, MESI, and PSCR (more details are in Section 2). We considered two main configurations: a basic machine with four processors and a high-performance one with 16 processors. The scheduling policy is based on cache-affinity; scheduler time-slice is 200,000 references. Cache size has been varied between 512 K bytes and 2 M bytes, while for block size we used 64 and 128 bytes for block size. The simulated processors are MIPS-R10000-like; paging relays on 4-KByte-page size; the bus is pipelined, supports transaction splitting, and processor-consistency memory model [15]; up to eight outstanding misses are allowed [26]. The base case study timings and parameter values for the simulator are summarized in Table 3. In particular, ideal cache access timing is reported as CPU Read/Write Cycle.

5.2. Performance metrics

As done in past works [2,16,18,28,31] rather than using the execution time to compare the performance of a multiprocessor system, we measured the global performance by using a single figure, which expresses the computational power delivered by the machine: the global system power (GSP). The GSP represents the number of processors of an ideal machine that does not have delay in accessing memory:

$$\mathrm{GSP} = \sum U_{\mathrm{cpu}},$$

where

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

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 provides a better measurement than execution time, since: (i) we are focusing on the memory performance rather than on the processor-core performance, and (ii) we do not execute a single program, in our simulations, but a combination of portions of programs. In these conditions, the machine is fully stressed, and GSP gives the necessary comparability when the performance evaluation requires varying the number of processors and other system parameters [2,16,31], while assuming a fixed internal processor architecture.

To investigate the effects on the performance due to memory subsystem operations, we analyzed the causes influencing memory latency, and thus total execution time (normalized as GSP). The memory latency, in turn, depends on the time necessary to perform a bus operation (Table 3) and on the waiting time to access bus (bus latency). Bus latency depends on the amount and kind of bus traffic. Bus traffic is constituted by *read-block transactions* (issued for each miss), *update transactions* and *coherence transactions* (*write transactions* or *invalidate transactions*, depending on the coherence protocol). Consequently, miss and coherence transactions affect performance because they affect the processor waiting-time directly (in the case of read misses) and contribute to bus latency. Therefore, to investigate the sources of performance bottlenecks, we reported a breakdown of misses—which includes *invalidation misses* and classical misses (sum of cold, capacity and

Table 3
Input parameters for the multiprocessor simulator (timings are in clock cycles)

Class	Parameter	Timings
CPU	Read cycle	2
	Write cycle	2
Cache	Cache size (Bytes)	512 K, 1 M, 2 M
	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)

conflict misses)—and the "number of coherence transaction per 100 memory references"—which includes either *write-transactions* or *invalidate transactions* depending on the coherence protocol. The rest of traffic is due to *update transactions*. Update transactions are a negligible part of bus-traffic (lower than 7% of read-block transactions or about 1% of bus occupancy) and thus they do not influence greatly our analysis.

In our analysis, we separated the contributions due to Cold Misses, Capacity+Conflict Misses and Invalidation Misses [22]. Cold Misses include first access misses by a given process when it is scheduled either for the first time or it is rescheduled on another processor (the latter are also known as *context-switch misses*). Capacity+Conflict Misses include misses caused by memory references of a process competing for the same block (intrinsic interference misses in [1]), and misses caused by references of sequential processes, executing on the same processor and competing for the same cache block (extrinsic interference misses in [1]). Invalidation Misses are due to accesses to data that are going to be reused on the same processor, but that have been invalidated in order to maintain coherence. Invalidation Misses are further classified, along with the coherence transaction type, by means of a generalization of an existing classification algorithm [23]. Our algorithm extends this classification to the case of passive sharing, finite size caches, and process migration [14].

In particular, Invalidation Misses are differentiated as true sharing misses, false sharing misses, passive sharing misses. True sharing misses and false sharing misses are classified according to an already know methodology [11,13,43]. Passive sharing misses are invalidation misses generated by the useless coherence maintaining of private data [16]. Clearly, these private data could appear as shared to the coherence protocol, because of the process migration. Similarly, coherence transactions are classified as true, false, and passive sharing transactions, either if they are invalidation or write transactions [14].

5.3. Analysis of the reference system

We started our analysis from a four-processor machine similar to cases used in literature [3,4,32,46], running the DSS_{26}workload (Table 2). We varied classical cache parameters to show that our results are not dependent on a particular configuration and to explore some points of the design space.

In detail, the reference four-processor machine has a 128-bit bus and 64-byte block size. We varied cache size (from 512 K to 2 M byte), cache associativity (1, 2, 4), and coherence protocol.

Global System Power (Fig. 1) is affected by the cache architecture. The GSP increases with larger cache sizes and/or more associativity. Anyway, this variation is limited to an 8% between the less (512 K byte, one way) and most performing (2 M byte, four-way) configuration. In this case, the role of the coherence protocol is less important, with a difference among the various protocols, for a given setup, which is less than 1%.

We analyzed the causes for this performance improvement (Fig. 2) by decomposing the miss rate in terms of traditional (cold, conflict and capacity) and invalidation misses. In Fig. 3, we show the contribution of each kind of sharing to the Invalidation Miss Rate and, in Fig. 4, to the Coherence-Transaction "Rate" (i.e. the number of coherence transactions per 100 memory references). We also differentiated between kernel and user overhead.

As expected, cache size mainly influences capacity misses. Invalidation misses increase slightly when increasing cache size or associativity (Fig. 2). For cache sizes larger than 1 M bytes, cold misses are the dominating part and invalidation misses weigh more and more (at least 30% of total misses). This indicates that, for our DSS workload, caches larger than 2 M bytes already capture the main working set. In fact, for cache sizes larger than 2 M bytes, Cold Misses remain constant, invalidation misses increase and only Capacity+Conflict misses may decrease, but their

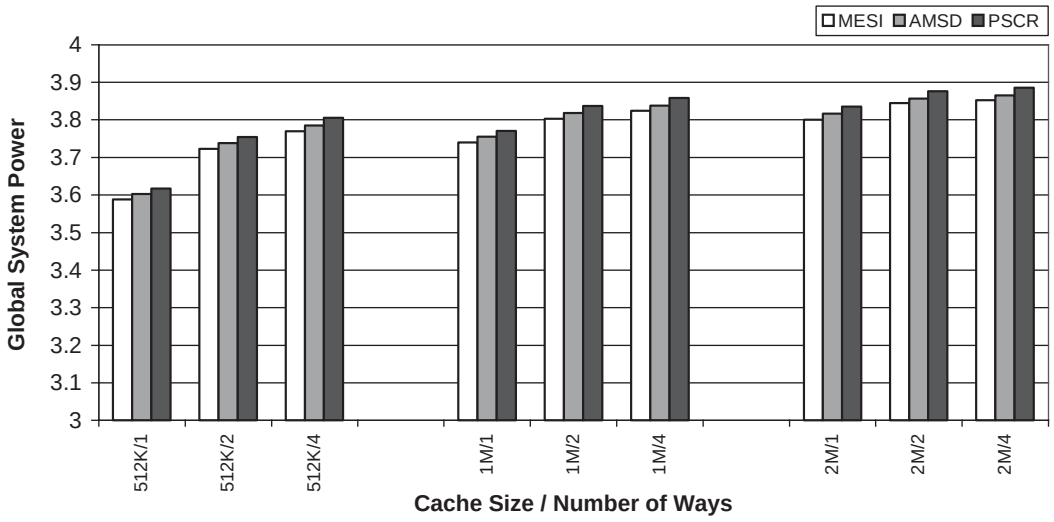

Fig. 1. Global System Power versus cache size (512 K, 1 M, 2 M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 4 processors, 64-byte block size and a cache affinity scheduler. PSCR presents the highest GSP, whilst MESI the lowest.

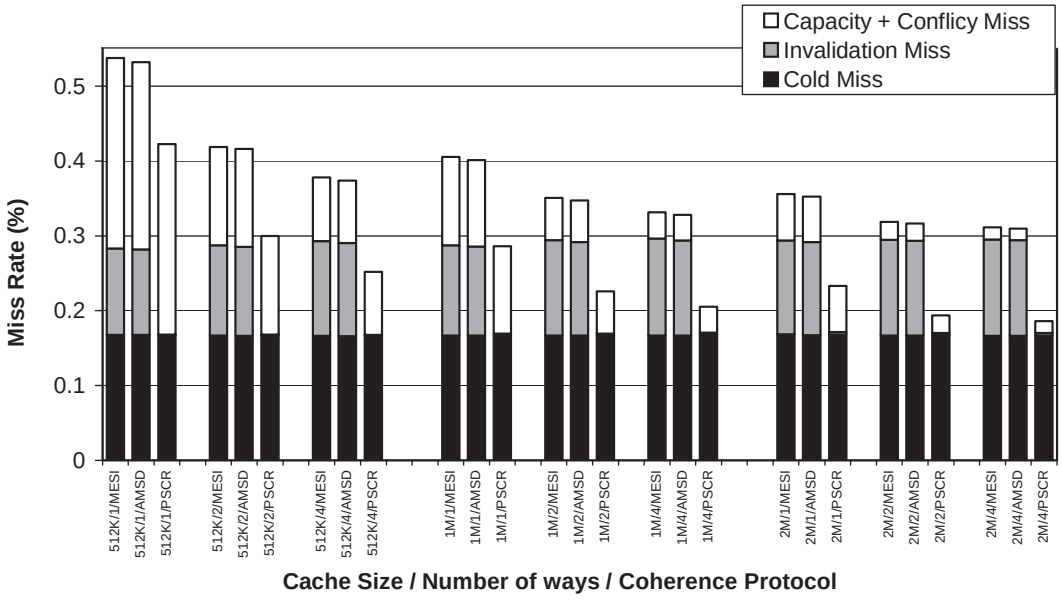

Fig. 2. Breakdown of miss rate versus cache size (512 K, 1 M, 2 M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assumes four processors, affinity scheduling and 64-byte block.

contribution to miss rate is already minimal. The solution based on PSCR presents the lowest miss rate, and negligible invalidation misses.

Our analysis of coherence overhead (Figs. 3 and 4) confirms that the major sources of overhead are invalidation misses for WI protocols (MESI and AMSD) and write-transactions for PSCR. Passive sharing overhead, either as invalidation misses, or coherence transactions, is minimal: this means that affinity scheduling performs well. In the user part, true sharing is dominant. In the kernel part, false sharing is the major portion of coherence overhead.

The cost of misses is dominating the performance and indeed we show in Fig. 1 that PSCR is able to achieve the best performance compared with the other protocols that we considered. The reason is the following: what PSCR looses in terms of extra coherence traffic, is then gained as saved misses. Indeed, misses are more costly in terms of bus timings and read-block transactions may produce a higher waiting time for the processor. Also, AMSD performs better than MESI. due to the reduction of Invalidation Miss Rate overhead (Fig. 2).

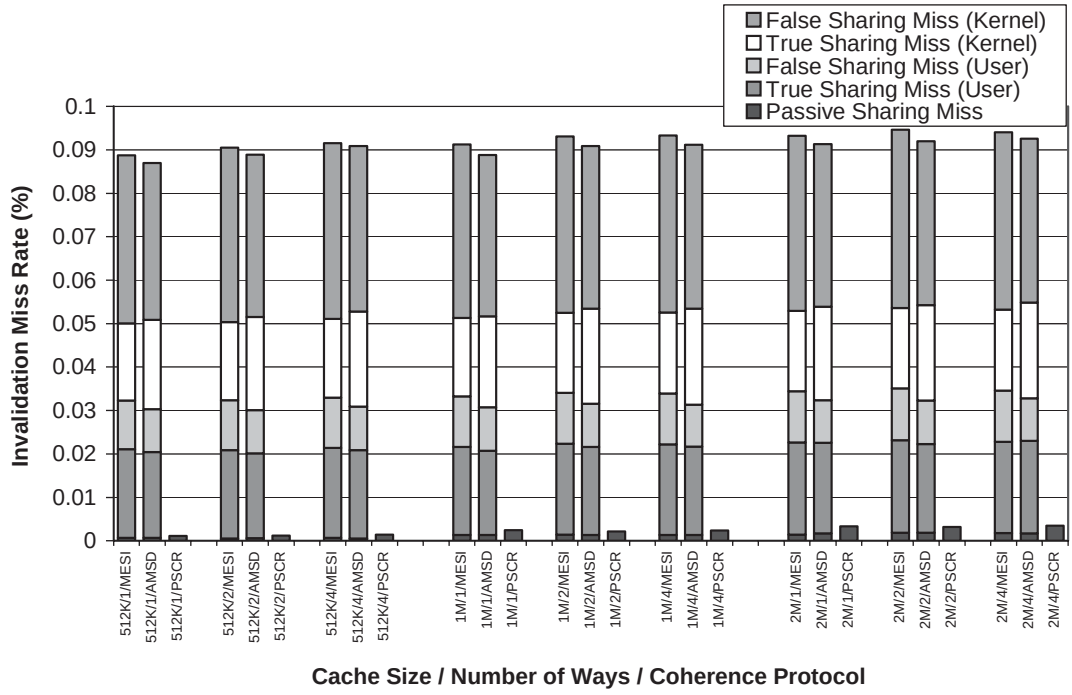

Fig. 3. Breakdown of miss rate versus cache size (512 K, 1 M, 2 M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume four processors, affinity scheduling and 64-byte block.

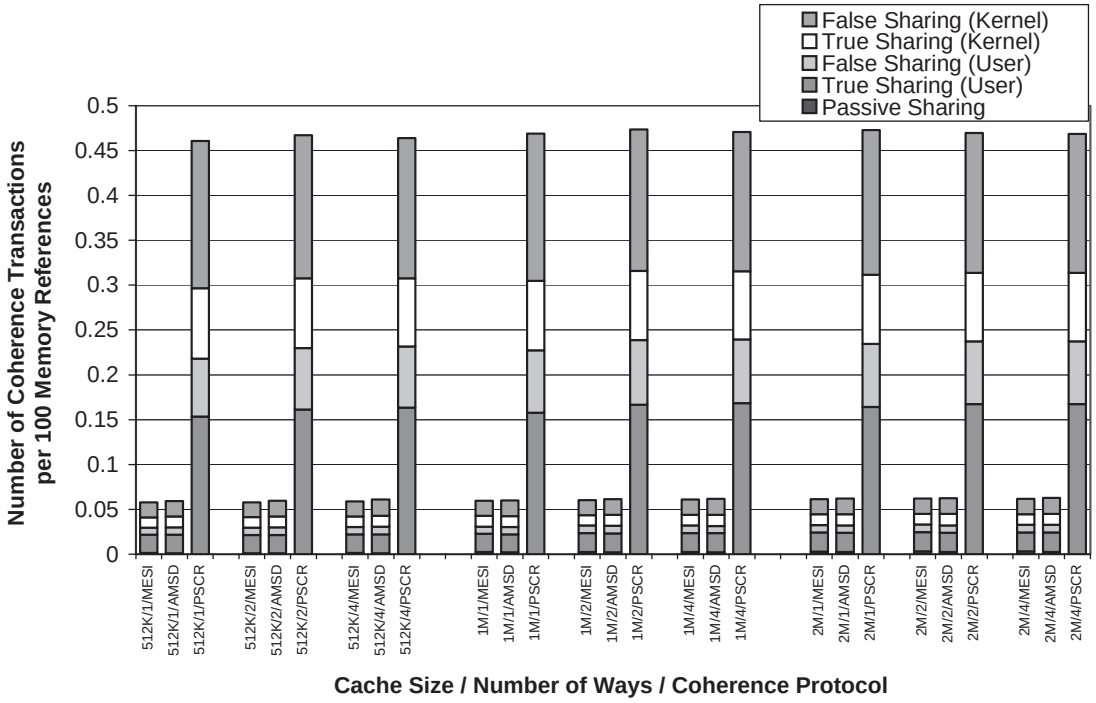

Fig. 4. Number of coherence transactions versus cache size (512 K, 1 M, 2 M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR). Coherence transactions are invalidate transactions in MESI, and AMSD, write transactions in PSCR. Data assume four processors, 64-byte block size, and an affinity scheduler.

Our conclusions, for the four-processor configuration, agree with previous studies as for the analysis of miss rate and the effects of coherence maintaining [3,4,46] (see also Section 3). Our analysis outlines also: (i) cache sizes larger than 2 M bytes already capture the working set of our workload; (ii) the ker-

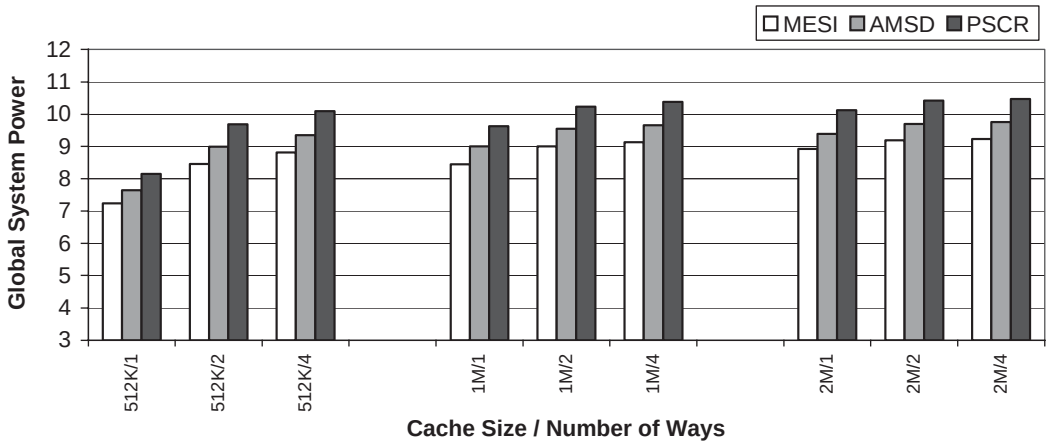

Fig. 5. Global System Power versus cache size (512 K, 1 M, 2 M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 64-byte block size, and an affinity scheduler.

nel effects account for about 50% of the coherence overhead.

5.4. Analysis of the high-end system

In the previous baseline case analysis, we have seen that the four-processor machine is efficient: architecture variations do not produce further gains (e.g. the performance differences among protocol are small). This kind of machine may not satisfy the performance needs of DSS workloads, and more performing systems are demanded. Given current processor-memory speeds, we considered a 'high-end' 16-processor configuration. A detailed analysis of the limitations of this system shows that this architecture makes sense, i.e. a high-end SMP system results efficient, if we solve the problems produced by the process migration. This architecture has not been analyzed in literature, and still represent a relatively economic solution to enhance the performance.

In the following, we will compare our results directly with the four-processor case and show the sensitivity to classical cache parameters (5.4.1). We will consider separately the case of different block size (5.4.2), the case of a different workload pressure in terms of number of processes (5.4.3), and the case of a different cache-to-cache latency (5.4.4).

5.4.1. Comparison with four-processor case and sensitivity to cache size and associativity

In Fig. 5, we show the GSP for several configurations. Protocols designed to reduce the effects of process migration achieve better performance. In particular, the performance gain of PSCR over MESI is at least 13% in all configurations. The influence of cache parameters is stronger, with a 28% difference between the most and less performing configuration.

We clarify the relationship between GSP and process migration, by showing the Miss Rate (Fig. 6) and the Number of Coherence Transactions (Fig. 8). In detail, we show the breakdown of the most varying components of Miss Rate (Invalidation Miss, Fig. 7) and breakdown of coherence transactions (Fig. 8). In the following, for the sake of clearness, we assume a 1-Mbyte-cache size, a 64-byte block size, and two-ways.

The GSP is mainly determined by the cost of read-block transactions and the cost of coherence-maintaining transactions (see also Section 5.2). Let us take MESI as baseline. AMSD has a greater GSP because of its lower Miss Rate (Fig. 6, and in particular Invalidation Miss Rate, Fig. 7). The small variation in the Number of Coherence Transactions (Fig. 8) does not weigh much on the performance. PSCR gains strongly from the reduction of Miss Rate, and at the same time, it is not penalized too much by the high Number of Coherence Transactions (Figs. 6 and 8). In detail:

- Classical misses (Cold and Conflict+Capacity) do not vary with the coherence protocol, although they increase while switching from 4 to 16 processors because of the higher migration of the processes: for the Cold Miss portion, because a process may be scheduled on a higher number of processors, and for the Conflict+Capacity, because a process has less opportunities of reusing its working set and it destroys the cache footprint generated by other processes.
- The much lower Miss Rate in PSCR is due to its low Invalidation Miss Rate (Fig. 6). In particular, PSCR does not have invalidation misses due to true and false sharing, neither in the user nor in the kernel mode (Fig. 7). On the contrary, in the other two protocols the latter factors weigh very much. This effect is more evident in the 16-processor case as compared to the four-processor case because of the higher probability of sharing data, produced by the increased number of processors.
- The DSS workload, in the PostgreSQL implementation, exhibits an access pattern to shared data, which speeds up

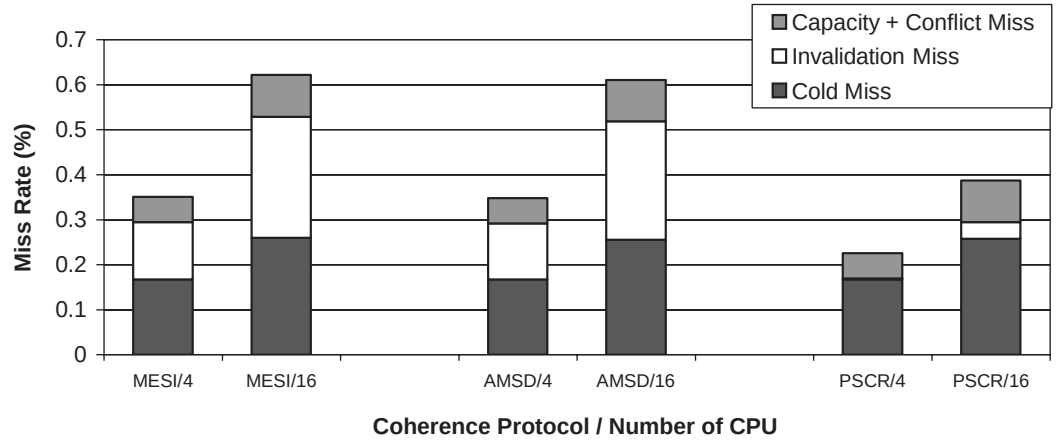

Fig. 6. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume an affinity scheduler, 64-byte block, 1 M-cache size two-way set associative. The higher number of processor causes more coherence misses (false plus true sharing) and more capacity and conflict misses. It is interesting to observe that while the aggregate cache size increases (from 4 to 16 M bytes), Capacity and Conflict misses also increase. This situation is different from the uni-processor case, where Capacity+Conflict misses always decrease when cache size increases. This effect is due to process migration.

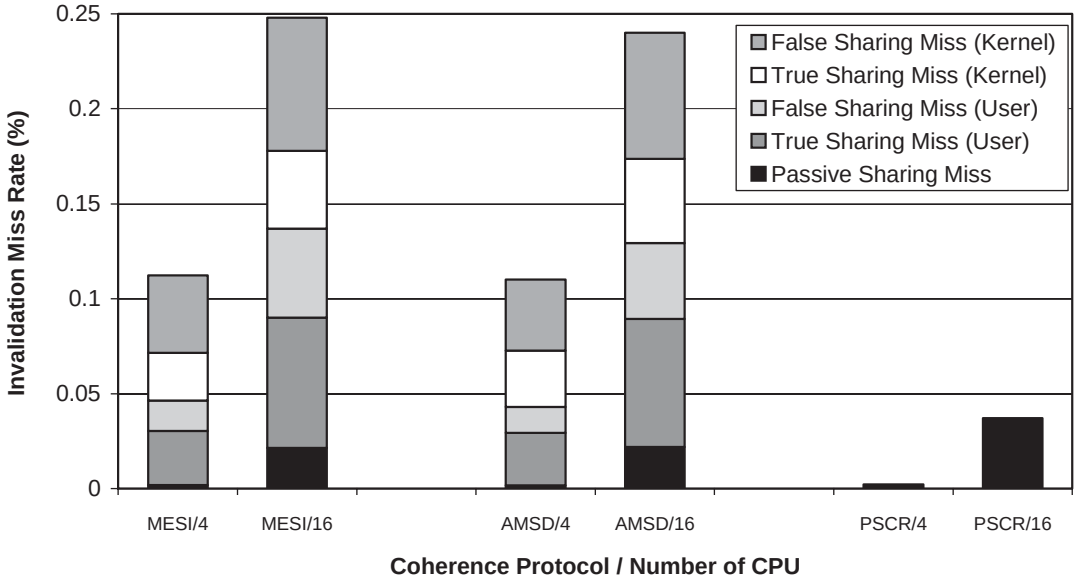

Fig. 7. Breakdown of Invalidation Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume, an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. Having more processors causes more coherence misses (false and true sharing). Passive sharing misses are slightly higher in PSCR compared to the other two protocols. This is a consequence of the selective invalidation mechanism of PSCR. In fact, as soon as a private block is fetched on another processor, PSCR invalidates all the remote copies of that block. In the other two protocols, the invalidation is actually performed on a write operation on (believed) shared data, thus less frequently than in PSCR

the WU strategy for maintain coherence among shared-data with respect to the WI strategy. We can explain such pattern with the following considerations: the TPC-D DSS queries are read-only queries and consequently, as already observed in [46], coherence misses and transactions involve PostgreSQL metadata (i.e. data structures that are needed by the DBMS to work rather than to store the data) and particularly data structures needed to implement software locks [46]. The type of access pattern to those data is one-producer/many-consumers. Such pattern advantage WU protocols, especially if there is a high number of processes (the ‘consumers’) that concurrently read shared data. When the number of processors switches from 4 to 16, the number of consumer processes increases, due to the higher number of processes concurrently in execution and, consequently, true sharing Invali-

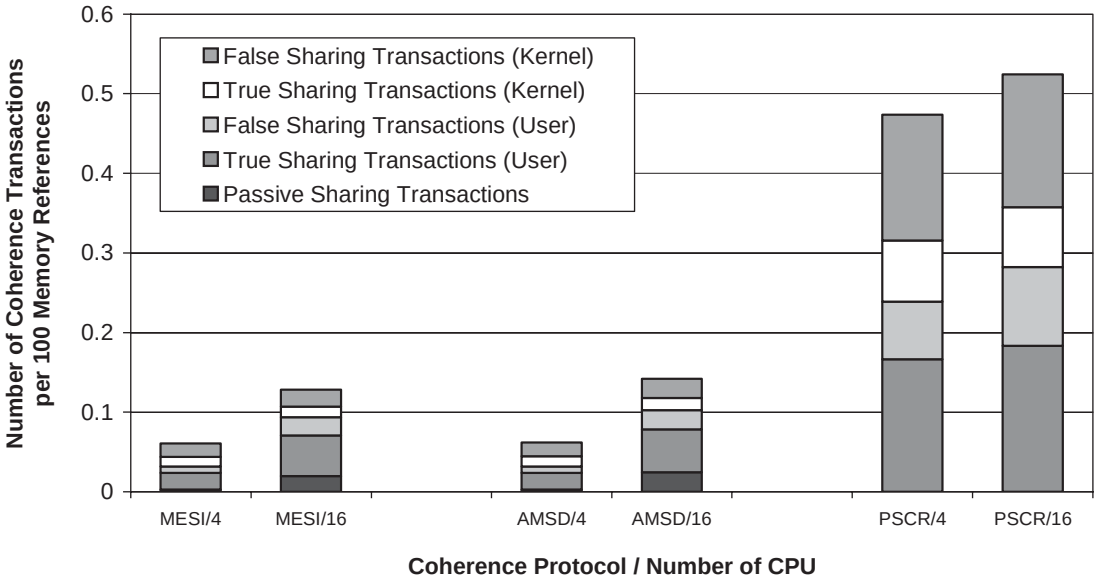

Fig. 8. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4, 16). Data assume, an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. As we switch to a 16-processor configuration, there is an increment in the sharing overhead in all of its components. This increment is more evident in the WI class protocols, also because there is more passive sharing overhead.

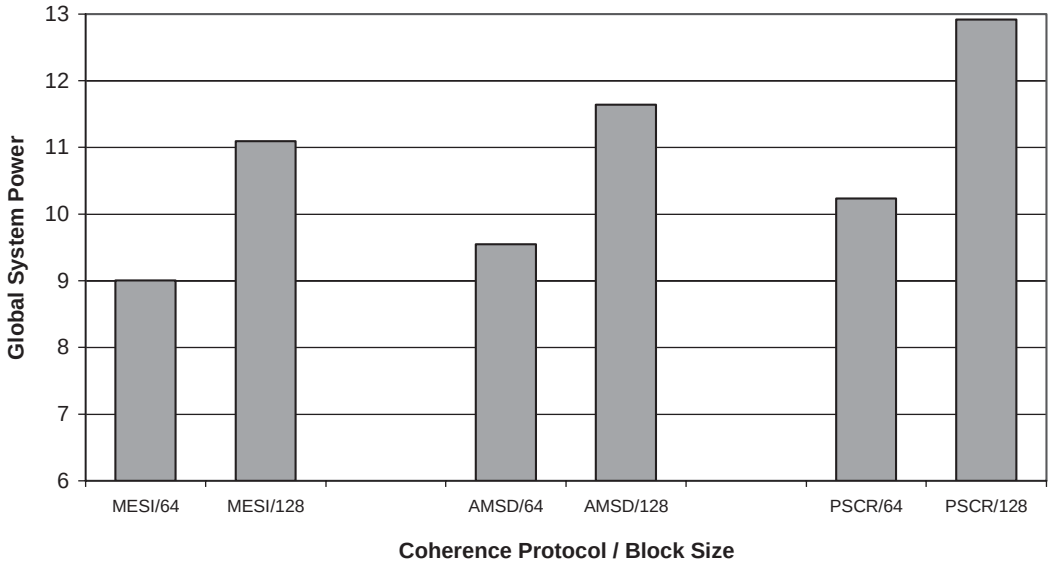

Fig. 9. Gobal System Power versus block size (64, 128 bytes) and coherence protocol (AMSD, MESI, PSCR). Data assume, an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches.

dation Misses and Transactions (in the user part) increase especially for WI protocols (Figs. 7 and 8), thus speeding-up the performance of the WU solution (PSCR).

5.4.2. *Analyzing the effects of a larger block size in the 'High-End' system*

We started our analysis from the 64-bytes block size for references comparison with other studies. When switching from 64 to 128 bytes, PSCR has further advantages in respect of the other two considered protocols (Fig. 9). This is due to the following two reasons. Firstly, we observe a reduction of Capacity+Conflict miss component (Fig. 10), a small reduction of coherence traffic (Fig. 12), and Invalidation Miss Rate (Fig. 11). Secondly, in the case of 64-byte block, the system is in saturation [1] [16] for all configura-

[1] We recall that a shared-bus shared-memory SMP system is in saturation [16] when the GSP does not increase at least of a given quantity (0.5), when we add one processor to the machine. In that case, we could easily recognize from the graph of GSP versus number-of-processors, that the system is exiting from the quasi-linearity zone.

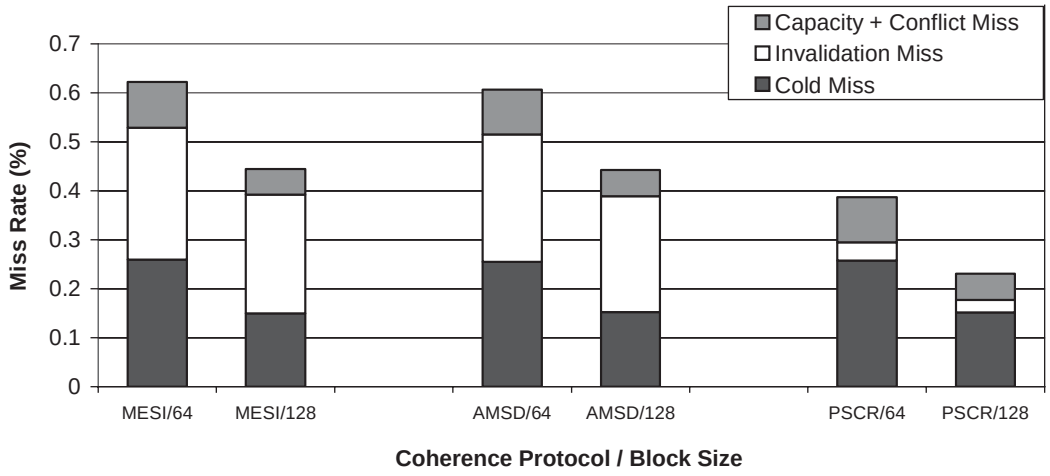

Fig. 10. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. There is a decrease of Cold, Capacity+Conflict miss components, and a little decrease of invalidation miss component.

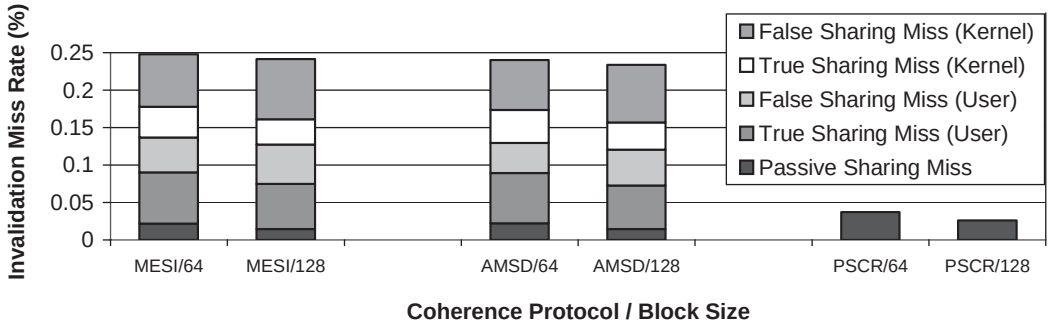

Fig. 11. Breakdown of Invalidation miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. Passive Sharing Misses decrease when increasing block size because the invalidation unit is larger

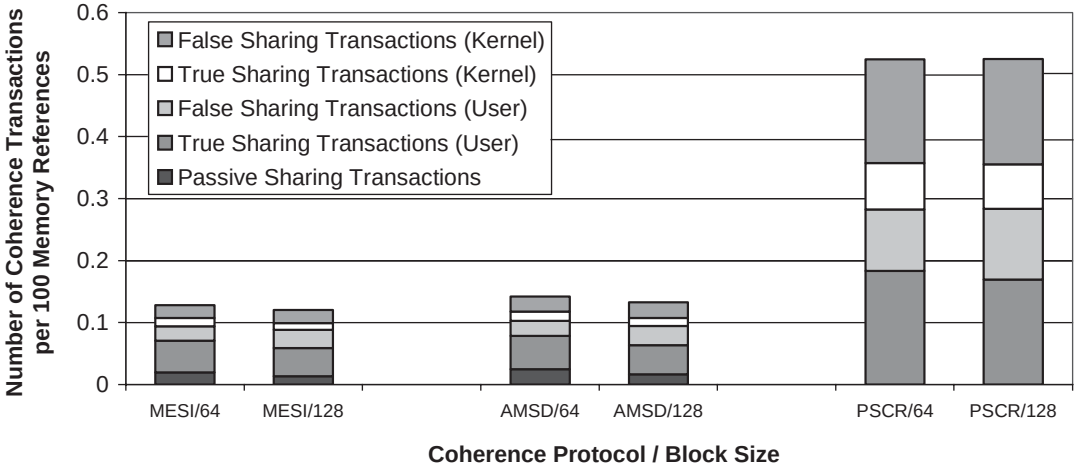

Fig. 12. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and block size (64, 128 byte). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches.

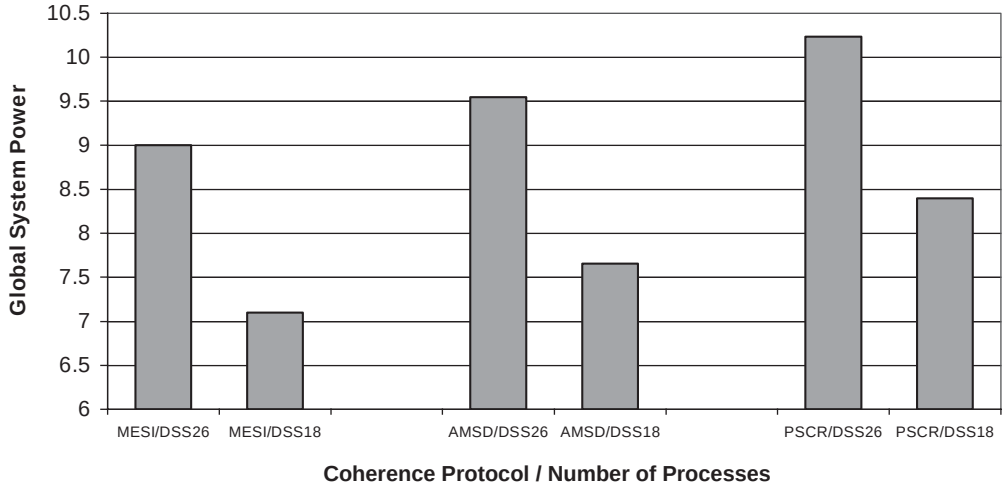

Fig. 13. Global System power versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26-DSS18). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches.

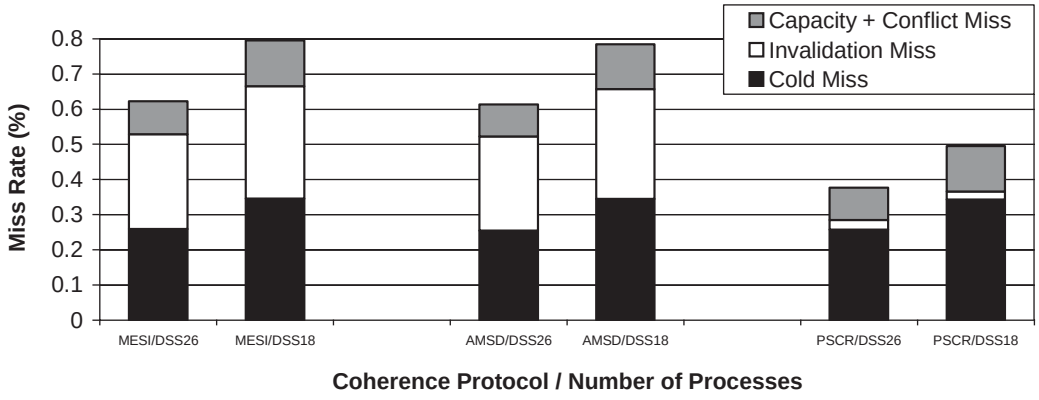

Fig. 14. Breakdown of Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26-DSS18). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. The workload DSS18 exhibits the higher miss rate, due to an increased number of Cold, Capacity, and Conflict Miss. This is a consequence of process migration, that affinity fails to mitigate.

tions of Fig. 9. In the case of 128-byte blocks, an architecture based on PSCR is not saturated, and thus we can use configurations with a higher number of processors efficiently. This is shown by the higher GSP for PSCR (almost 13 for a 16-processor system) compared to other solutions. When switching from 64 to 128 bytes, the increase in the GSP is 27% percent for PSCR, and only a 20% for the other protocols (Fig. 12). We observe that a block size larger than 128 bytes produces diminishing returns, because the increased cost of read-block transaction is not compensated by the reduction of the number of misses. Similar result has been obtained for a CC-NUMA machine running a DSS workload [28]. In that work, the number of "Effective Processors" (a metric similar to our GSP) for a 16-processor CC-NUMA system was almost the same that we obtained for a cheaper shared-bus shared-memory system.

5.4.3. Analyzing the effects of variations in the number of processes of the workload

We considered another scenario, where the number of processes in the workload may vary and thus the scheduler could fail in applying affinity. The affinity scheduling could fail when the number of ready processes is limited. We defined a new workload (DSS_{18}, Table 2) having characteristics similar to DSS_{26} workload that used in the previous experiments, but constituted of only 18 processes. The machine under study is still the 16-processor one. In such a condition, the scheduler can only choose between (at most) two ready processes.

The measured miss rate and number of coherence transactions (Figs. 14–16) shows an interesting behavior. The miss rate, and in particular Cold, Conflict+Capacity miss rate, increases in respect to the DSS_{26} workload. This is conse-

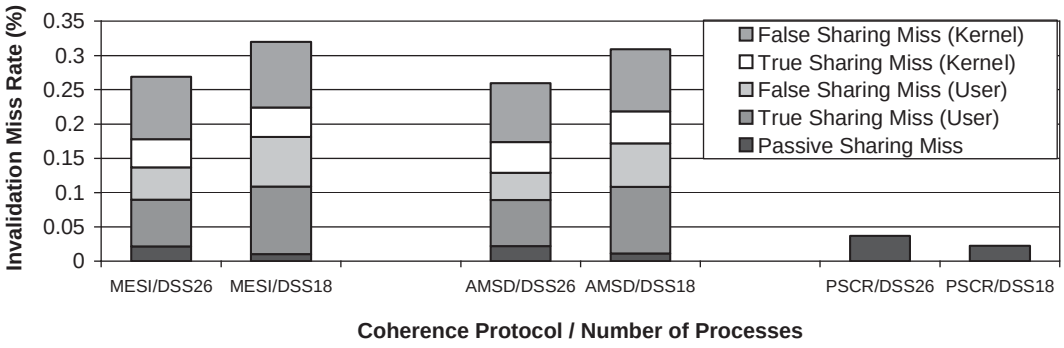

Fig. 15. Breakdown of Invalidation Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26-DSS18). Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches. Passive sharing misses decrease, while true and false sharing misses increase. This is consequence of the failure of affinity scheduling: in the DSS18 execution. Processes migrate more than in DSS26 execution, thus generating more reuse of shared data (and more invalidation misses on shared data) but lower reuse of private data (and lower number of passive sharing misses).

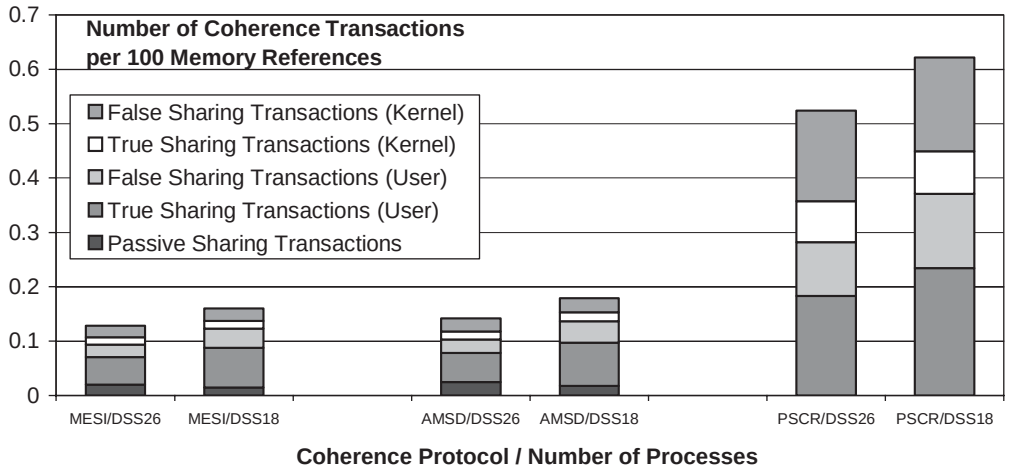

Fig. 16. Breakdown of Coherence Transactions versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26-DSS18). Data assume affinity, 64-byte block, 1 M-byte two-way set associative caches.

quence of process migration, and it is determined by the failure of the affinity requirement: as the number of processes is almost equal to the number of processors, it is not always possible for the system to reschedule a process on the processor where it last executed. In such cases, PSCR can reduce greatly the associated overhead and it achieves the best performance. In the case of DSS_{18} workload, PSCR reaches a GSP that is almost equal to the GSP reached by MESI protocol in the case of DSS_{26} workload (Fig. 13). When moving from the 26- to the 18-process condition, the reduction of GSP is only 15% for the PSCR protocol, while it is 21% for the other protocols.

The main conclusion here is that PSCR maintains its advantage also in different load conditions, while the other protocols are more penalized by critical scheduling conditions.

5.4.4. Analyzing the effects of variations in the cache-to-cache transfer latency

In our implementation of the coherence protocols, we assumed that data, missing in a local cache but present in a remote cache, will be supplied by a remote cache. This is useful because the transfer between caches is usually faster than the transfer between cache and memory (see Table 3). This is compatible with the early design of bus-based systems, and it derives from the assumption that cache could supply data more quickly than memory. This is especially true in the case of small-scale SMP architectures [9]. However, this advantage is not necessary present in modern high-end SMP architecture [9], such as the Sun series based on the Wildfire [21] or the Fireplan bus [7], in which cache-to-cache latency is similar to the latency to access the main memory. As previous studies of commercial workloads have highlighted that a large portion of misses can be served by cache-to-cache transfers [25,32], we analyzed the effect on GSP of different cache-to-cache latencies. Fig. 17 shows GSP for three protocols, when cache-to-cache transfer latency is varied. The cases analyzed in previous analysis assumed a cache-to-cache latency to read-block latency ratio of $\frac{1}{16}$. The reason for this choice is a comparability with previous studies: if we had chosen higher cache-to-cache laten-

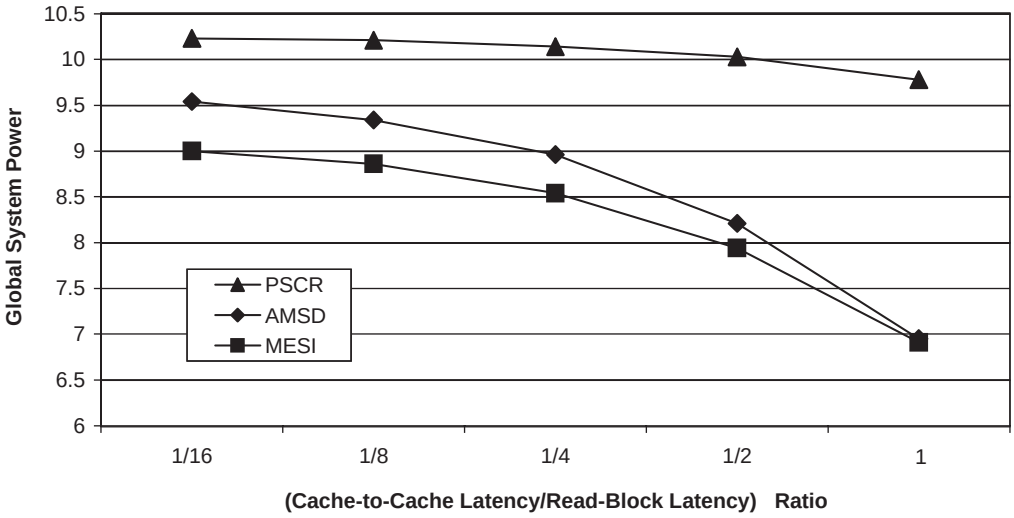

Fig. 17. Global System Power versus coherence protocol (AMSD, MESI, PSCR) and the ratio between cache-to-cache transfer latency and read-block latency. Data assume an affinity scheduler, 64-byte block, 1 M-byte two-way set associative caches.

cies from the beginning, the result of our experiments would have been more advantageous for the proposed protocol solutions.

As can be seen, when we increase that latency, GSP decreases. However, the architecture based on PSCR shows less sensitivity to this parameter. This is consequence of the lower number of cache-to-cache transfers needed in PSCR with respect to the other protocols. In fact, as a benefit of the WU strategy, shared data are never invalidated in PSCR. This also generates several updated copies in several caches: a migrating process does not need to reload them from remote cache or memory. When cache-to-cache latency is increased up to read-block latency, the advantage of PSCR is about 50% in comparison with the other two coherence protocols.

6. Conclusions

We evaluated the memory performance of a SMP multiprocessor running a DSS workload, by considering several different choices that could improve the overall performance of the system like number of (efficient) processors, cache affinity, cache parameters, coherence protocols (in particular MESI—a pure WI protocol, widely used in high-performance multiprocessors,—AMSD—a WI protocol designed to reduce effects of data migrations—and PSCR—a coherence protocol using an hybrid strategy, that is WU for shared data and WI for private data, designed to reduce the effect of process migration).

The DSS workload was setup using the PostgreSQL DBMS executing queries of the TPC-D benchmark and typical Unix shell commands and daemons. We considered kernel effects that are more relevant to our analysis like process scheduling, virtual memory mapping, user/kernel code interactions.

Our conclusions, for the four-processor case, agree with previous studies as for the analysis of miss rate and the effects of coherence maintaining. Our analysis outlines also: (i) cache sizes larger than 2 M bytes already capture the working set of such workload; (ii) the kernel effects account for about 50% of the coherence overhead. Previous studies that considered DSS workloads were mostly limited to four-processor systems, did not consider the effects of process migration, and did not correlate the amount of sharing to the performance of the system.

Differently from previous works, our analysis shows quantitatively the various types of sharing (in particular passive sharing), shows the effects of process migration on the memory subsystem, and highlights the influence of architectural parameters in case of a relatively high number of processors. This analysis allows us also to point out solutions that increase performance (without changing the processor architecture), like coherence protocols (especially in the case of current-system cache-to-cache transfer latency).

Our analysis of a “high-end” machine considered a 16-processor SMP. We analyzed variations of classical cache parameters, variations in the workload pressure on the scheduler due to a different number of processes, and variations of the cache-to-cache latency. We found that in the high-end systems some factors, that were less noticed in the four-processor case, become more evident.

MESI protocol is not the best choice in high-end SMP architectures: AMSD improves the performance of a DSS system of about 10% compared to MESI, PSCR improves the performance of about 20% compared to MESI when we use a low cache-to-cache latency.

DSS workloads running on SMP architectures generate a variable load. The affinity scheduler may fail to deliver the affinity requirements. The use of adequate protocols can

give an advantage to SMP-based DSS systems, in terms of a reduced load sensitivity and a performance boost of about 50% in case of high (current-system) cache-to cache latencies.

References

[1] A. Agarwal, Analysis of Cache Performance for Operating Systems and Multiprogramming, Kluwer Academic Publishers, Norwell, MA, 1989.

[2] J.K. Archibald, J.L. Baer, Cache coherence protocols: evaluation using a multiprocessor simulation model, ACM Trans. Comput. Systems 4 (April 1986) 273–298.

[3] L.A. Barroso, K. Gharachorloo, E. Bugnion, Memory system characterization of commercial workloads, in: Proceedings of 25th International Symposium on Computer Architecture, Barcelona, Spain, June 1998, pp. 3–14.

[4] Q. Cao, J. Torrellas, P. Trancoso, J.L. Larriba-Pey, B. Knighten, Y. Won, Detailed characterization of a quad Pentium Pro server running TPC-D, Proceedings of the International Conference on Computer Design, October 1999, pp. 108–115.

[5] R. Chandra, S. Devine, B. Verghese, A. Gupta, M. Rosenblum, Scheduling and page migration for multiprocessor compute servers, Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, October 1994, pp. 12–24.

[6] J. Chapin, S. Herrod, M. Rosenblum, A. Gupta Memory system performance of UNIX on CC-NUMA multiprocessors, Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1995, pp. 1–13.

[7] A. Charlesworth, The sun fireplan system interconnect, Proceedings of the Conference on High Performance Networking and Computing (SC01), November 2001, http://www.sc2001.org/papers/pap.pap150.pdf.

[8] A.L. Cox, R.J. Fowler, Adaptive cache coherency for detecting migratory shared data, in: Proceedings of the 20th International Symposium on Computer Architecture, San Diego, CA, May 1993, pp. 98–108.

[9] D.E. Culler, J.P. Singh, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[10] Z. Cvetanovic, D. Bhandarkar, Characterization of alpha AXP performance using TP and SPEC workloads, Proceedings of the 21st International Symposium on Computer Architecture, April 1994, pp. 60–70.

[11] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, P. Stenström, The detection and elimination of useless miss in multiprocessor, in: Proceedings of the 20th International Symposium on Computer Architecture, San Diego, CA, May 1993, pp. 88–97.

[12] S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, R.L. Stamm, D.M. Tullsen, Simultaneous multithreading: a platform for next-generation processors, IEEE Micro. 17 (5) (October 1997) 12–19.

[13] S.J. Eggers, T.E. Jeremiassen, Eliminating false sharing, Proceedings of the 1991 International Conference on Parallel Processing, August 1991, pp. I:377–381.

[14] P. Foglia, An algorithm for the classification of coherence related overhead in shared-bus shared-memory multiprocessors, IEEE TCCA Newsletter (January 2001) 40–46.

[15] K. Gharachorloo, A. Gupta, 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, CA, April 1991, pp. 245–357.

[16] R. Giorgi, C.A. Prete, PSCR: a coherence protocol for eliminating passive sharing in shared-bus shared-memory multiprocessors, IEEE Trans. Parallel Distributed Systems 10 (7) (July 1999) 742–763.

[17] R. Giorgi, C. Prete, G. Prina, L. Ricciardi, A hybrid approach to trace generation for performance evaluation of shared-bus multiprocessors, in: Proceedings of the 22nd EuroMicro International Conference, Prague, September 1996, pp. 207–241.

[18] R. Giorgi, C. Prete, G. Prina, L. Ricciardi, Trace factory: a workload generation environment for trace-driven simulation of shared-bus multiprocessor, IEEE Concurrency 5 (4) (October–December 1997) 54–68.

[19] S.R. Goldschmidt, J.L. Hennessy, The accuracy of trace-driven simulations of multiprocessors, Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1993, pp. 146–157.

[20] A.M. Grizzaffi Maynard, C.M. Donnelly, B.R. Olszewski, Contrasting characteristics and cache performance of technical and multi-user commercial workloads, Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, October 1994, pp. 158–170.

[21] E. Hagersten, M. Koster, WildFire: a scalable path for SMPs, Proceedings of the Fifth Symposium on High-Performance Computer Architecture, January 1999, pp. 172–181.

[22] J. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Third ed., Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[23] R.L. Hyde, B.D. Fleisch, An analysis of degenerate sharing and false coherence, J. Parallel Distributed Comput. 34 (2) (May 1996) 183–195.

[24] T.E. Jeremiassen, S.J. Eggers, Reducing false sharing on shared memory multiprocessors through compile time data transformations, ACM SIGPLAN Notices 30 (8) (August 1995) 179–188.

[25] K. Keeton, D. Patterson, Y. He, R. Raphael, W. Baker, Performance characterization of a quad Pentium Pro SMP using OLTP workloads, Proceedings of the 25th International Symposium on Computer Architecture, June 1998, pp. 15–26.

[26] D. Kroft, Lockup-free instruction fetch/prefetch cache organization, Proceedings of the Eighth International Symposium on Computer Architecture, June 1981, pp. 81–87.

[27] J.L. Lo, L.A. Barroso, S.J. Eggers, K.G. Gharachorloo, H.M. Levy, S.S. Pareck, An analysis of database workload performance on simultaneous multithreaded processors, in: Proceedings of the 25th Annual International Symposium on Computer Architecture, Barcelona, Spain, June 1998, pp. 39–50.

[28] T. Lovett, R. Clapp, StiNG: a CC-NUMA computer system for the commercial marketplace, Proceedings of the 23rd International Symposium on Computer Architecture, May 1996, pp. 308–317.

[29] M. Poess, C. Floyd, New TPC benchmarks for decision support and web commerce, ACM SIGMOD Rec. 29 (4) (December 2000) 64–71.

[30] C.A. Prete, G. Prina, R. Giorgi, L. Ricciardi, Some considerations about passive sharing in shared-memory multiprocessors, IEEE TCCA Newsletter, (March 1997) 34–40.

[31] C.A. Prete, G. Prina, L. Ricciardi, A trace driven simulator for performance evaluation of cache-based multiprocessor system, IEEE Trans. Parallel Distributed Systems 6 (9) (September 1995) 915–929.

[32] P. Ranganathan, K. Gharachorloo, S.V. Adve, 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, San Jose, CA, 1998, pp. 307–318.

[33] T. Shanley, Mindshare Inc., Pentium Pro and Pentium II System Architecture, Second ed., Addison-Wesley, Reading, MA, 1999.

[34] P. Stenström, M. Brorsson, L. Sandberg, An adaptive cache coherence protocol optimized for migratory sharing, Proceedings of the 20th Annual International Symposium on Computer Architecture., May 1993.

[35] P. Stenström, E. Hagersten, D.J. Li, M. Martonosi, M. Venugopal, Trends in shared memory multiprocessing, IEEE Comput. 30 (12) (December 1997) 44–50.
[36] C.B. Stunkel, B. Janssens, W.K. Fuchs, Address tracing for parallel machines, IEEE Comput. 24 (1) (January 1991) 31–45.
[37] P. Sweazey, A.J. Smith, A class of compatible cache consistency protocols and their support by the IEEE futurebus, Proceedings of the 13th International Symposium on Computer Architecture, June 1986, pp. 414–423.
[38] M. Tomasevic, V. Milutinovic, The cache coherence problem in shared-memory multiprocessors—hardware solutions, IEEE Computer Society Press, Los Alamitos, CA, April 1993.
[39] M. Tomasevic, V. Milutinovic, Hardware approaches to cache coherence in shared-memory multiprocessors, IEEE Micro 14 (5) (October 1994) 52–59;
M. Tomasevic, V. Milutinovic, IEEE Micro 14 (6) (December 1994) 61–66.
[40] M. Tomasevic, V. Milutinovic, The word-invalidate cache coherence protocol, Microprocess. microsystems 20 (March 1996) 3–16.
[41] J. Torrellas, M.S. Lam, J.L. Hennessy, Share data placement optimizations to reduce multiprocessor cache miss rates, in: Proceedings of 1990 International Conference on Parallel Processing, vol. 2: Software, Urbana-Champaign, IL, August 1990, pp. 266–270.
[42] J. Torrellas, A. Gupta, J. Hennessy, Characterizing the caching and synchronization performance of a multiprocessor operating system, Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating System, September 1992, pp. 162–174.
[43] J. Torrellas, M.S. Lam, J.L. Hennessy, False sharing and spatial locality in multiprocessor caches, IEEE Trans. Comput. 43 (6) (June 1994) 651–663.
[44] Transaction Processing Performance Council, "TPC Benchmark B (Online Transaction Processing) Standard Specification", June 1994, http://www.tpc.org.
[45] Transaction Processing Performance Council, "TPC Benchmark D (Decision Support) Standard Specification", December 1995, http://www.tpc.org.
[46] P. Trancoso, J.L. Larriba-Pey, Z. Zhang, J. Torrellas, The memory performance of DSS commercial workloads in shared-memory multiprocessors, in: Proceedings of the Third International Symposium on High Performance Computer Architecture, Los Alamitos, CA, February 1997, pp. 250–260.
[47] R.A. Uhlig, T.N. Mudge, Trace-driven memory simulation: a survey, ACM Comput. Surveys. (June 1997) 128–170.
[48] S.C. Woo, M. Ohara, E. Torrie, J.P. Shingh, A. Gupta, The SPLASH-2 Programs: characterization and methodological considerations, In: Proceedings of 22nd Annual International Symposium on Computer Architecture (May 1994) 24–36.
[49] K.C. Yeager, The MIPS R10000 superscalar microprocessor, IEEE Micro 16 (4) (August 1996) 42–50.
[50] R. Yu, L. Bhuyan, R. Iyer, Comparing the memory system performance of DSS workloads on the HP V-class and SGI origin 2000, in: Proceedings of the International Parallel and Distributed Processing Symposium, Fort Lauderdale, FL, April 2002.
[51] A. Yu, J. Chen, The POSTGRES95 User Manual, Computer Science Division Department of EECS, University of California at Berkeley, July 1995.

Pierfrancesco Foglia received the M.S. degree and the Ph.D. degree in Computer Engineering both from the University of Pisa. At present, he is Assistant Professor at the Department of Information Engineering, University of Pisa, Italy. His research interests include computer architecture, coherence protocols, network protocol, high-performance server and infrastructure for E-Commerce. He has been project manager in the project involving the University of Pisa and Siemens ICN for the development of a management system for a telecommunication network of GSM devices. He took part in the ESPRIT SPP project, where he validates a multiprocessor architecture for a geographic system. He is member of the 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 (USA). He received his M.S. in electronic engineering, summa cum laude, and his Ph.D. 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.

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, CA). 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 Ph.D. from the University of Pisa in 1989. He is a member of IEEE, IEEE Computer Society and ACM.