Speeding-up Multiprocessors Running DSS Workloads through Coherence Protocols

Pierfrancesco Foglia, Roberto Giorgi, Cosimo Antonio Prete

Dipartimento di Ingegneria dell'Informazione, Facolta' di Ingegneria, Universita' di Pisa,
Via Diotisalvi, 2 – 56126 PISA (Italy)
foglia@iet.unipi.it, {prete,giorgi}@unici.it

Abstract

In this work, we analyze how a DSS (Decision Support System) workload can be accelerated in the case of a shared-bus shared-memory multiprocessor, by adding simple support to the classical MESI solution for the coherence protocol. The DSS workload has been set-up utilizing the TPC-D benchmark on the PostgreSQL DBMS. Analysis has been performed via trace driven simulation and the operating system effects are also considered in our evaluation.

We analyzed a basic four-processor and a high-end sixteen-processor machine, implementing MESI and two coherence protocols which deal with migration of processes and data: PSCR and AMSD. Results show that, even in the four processor case, for a DSS workload the use of a write-update protocol with a selective invalidation strategy for private data improves performance (and scalability) with respect to a classical MESI based solution, because of the access pattern to shared data and the lower bus utilization due to the absence of invalidation miss when we eliminate the contribution of passive sharing. In the 16 processor case, and especially in situation when the scheduler cannot apply the affinity requirements, the gain becomes more important: the advantage of a write-update protocol with a selective invalidation strategy for private data, in term of execution time, could be quantified in a 20% relatively to the other evaluated protocols. This advantage is about 50% in the case of high cache-to-cache transfer latency.

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

1 Introduction

An ever-increasing number of multiprocessor server systems shipped today run commercial workloads [50]. These workloads include databases applications such as online transaction processing (OLTP) and decision support system (DSS), file servers and application servers [34]. Nevertheless, technical workloads were widely used to drive the design of current multiprocessor systems [34], [10], [50] and different studies have shown that commercial workloads exhibit different behavior from technical ones [20] [25].

The simpler design for a multiprocessor system is a shared-bus shared-memory architecture [51]. In shared-bus systems, 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]. Cache memories introduce the coherency problem and the need for adopting adequate coherence protocols [37], [38]. The main coherence protocol classes are Write-Update (WU) and

Write-Invalidate (WI) [37]. 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. 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, together with the accesses pattern to application data, determines the best protocol choice for a given workload [55], [56], [19]. Different optimizations to minimize coherence overhead have been proposed [12], [37], [42], [18], also acting at compile time and architectural level (as the adoption of adequate coherence protocols [39], [18]).

When the performance achieved by shared-bus shared-memory multiprocessor is not sufficient, and this is typical for DBMS applications, an SMP (Symmetrical Multi-Processing, which includes shared-bus shared-memory architectures) or a NUMA (Non Uniform Memory Access) approach can be utilized [51], [22]. In the 1st case, a crossbar switch interconnects the processing elements, in the 2nd an Interconnection Network. Such solutions increase the communication bandwidth among elements, thus allowing more CPU to be added to the system, but at the cost of more expensive and complex communication network. In both the design, the basic building bock (node) may be a single processor system or, better, a shared-bus shared-memory multiprocessor (example are the HP V-Class and the SGI Origin families of multiprocessors [52]). In this way, by adding high performance nodes, we can achieve the desired level of performance with only a little number of elements, simplifying the crossbar or IC network design, and lowering the price of the whole system. Unfortunately, due to limited bus bandwidth, only a small number of CPU (max 4 for the Pentium family of CPU [32]) may be included in the single node.

The aim of this paper is to analyze the scalability of shared-bus shared-memory multiprocessors running commercial workload and DSS applications in particular, and to investigate solutions, as concern the memory subsystem, which can increase the processing power of such architectures. In this way, we can meet the performance requirement of commercial applications with a single shared-bus shared-memory machine, or we can adopt more performing nodes in an SMP or NUMA design, allowing simpler and cheaper design of switch or IC networks.

Decision Support Systems are based on DBMS; they are utilized to extract management information and analyze huge historical data: a typical DSS activity is performed via a query to look for ways to increase revenues, such as an SQL query to quantify the amount of revenue increase that would have resulted from eliminating a company discount in a given percentage in a given year [53]. DSS queries are long running, typically scanning large amounts of data [50]. In particular, such queries are, in most part, read-only queries. Consequently, as already observed in [45], [52], coherence activity and transactions involve essentially DBMS metadata (i.e. data structures that are needed by the DBMS to work rather than to store the data) and particularly data structure needed to implement software locks [45], [52]. The type of access pattern to those data is one-producer/many-consumers. Such pattern may advantage solution based on the Write Update family of coherence protocols, especially if there is a high number of processes (the ‘consumers’) that concurrently read shared data [19]. These considerations can suggest the adoption of other coherence protocols, with respect to the usually utilized MESI WI protocol [36].

In our evaluation, the DSS workload of the system is generated by running all the TPC-D queries [44] (through the PostgreSQL [49] 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 [17], 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], [41]. 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 [18]. As *passive sharing* overhead dramatically decreases the performance of pure WU protocols [18], [47], we considered in our analysis an hybrid WU protocol, PSCR [18], which adopts a selective invalidation strategy for private data and an hybrid WI protocol, AMSD [8], [33], which deals with migration of data and, then, of processes.

Our results show that, in the case of a 4-processor configuration, performance of DSS workload is moderately influenced by cache parameters and the influence of coherence protocol is minimal. In 16 processors configurations, the performance differences due to the adoption of different architectural solutions are significant. In high-end architectures, MESI is not the best choice and 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 of about 20%, and adapt better to workload. 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. In Section 3, we report the results of studies related to the analysis of workloads similar to ours, differentiating our contribution. In Section 4, we present our experimental setup and methodology. In Section 5, we 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* [40], [42], which occurs when the same cached data item is referenced by different processes concurrently running on different processors; ii) *false sharing* [40], [42], 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.

The main coherence protocol classes are Write-Update (WU) and Write-Invalidate (WI) [37]. 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. An optimal selection for the coherence protocol can be made by considering the traffic induced by the two

approaches in case of different sharing and its effects on performances [55]. The coherence overhead induced by a WU protocol is due to all the operations needed to update the remote copies. Whereas, a WI protocol invalidates remote copies and processors generate a miss on the access to the invalidated copy (invalidation miss). The access patterns to shared data determine the coherence overhead. In particular, fine-grain sharing denotes high contention for shared-data; sequential sharing is characterized by long sequences of writes to the same memory item performed by the same processor. A WI protocol is adequate in the case of sequential sharing, whilst, in general, a WU protocol performs better than WI for program characterized by fine-grain sharing [56], [19], [16] (in alternative, a metric based on the write-run and external rereads model can be used to estimate the best protocol choice [2]). In our evaluation, we considered MESI protocol as baseline, because it is used in most of the high performance processors today (like AMD K5 and K6, PowerPC series, SUN UltraSparc II, SGI R10000, Intel Pentium, Pentium Pro series (Pro, II, III), Pentium 4, and IA-64/Itanium). Then, based on the previous considerations on the access pattern of DSS workloads, we included a selective invalidation protocol based on a WU policy for manage shared data, which limits most effects of process migration (PSCR [18]) and a WI protocol specifically designed to treat the data migration (AMSD [8], [33]).

Our implementation of MESI uses the classical MESI protocol states [36] 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 [32]. 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*.

PSCR (Passive Shared Copy Removal) 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 [18]. This technique reduces coherence overhead (especially passive sharing overhead), otherwise dominant in a pure WU protocol [30]. Invalidate transactions are eliminated and coherence overhead is due to *Write Transactions* and *Invalidation Misses* caused by local invalidation (*Passive Sharing Misses*).

AMSD [8], [33] 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. Although 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 it restarts on a new processor (context-switch misses); ii) useless coherence transactions due to passive sharing. These situations may become frequent due to 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 on DSS systems

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 [34], [50] motivated many studies that consider benchmarks like TPC-series (including DSS, OLTP, WEB-server benchmarks) representative of commercial workloads [45], [3], [4], [31], [28], [27], [25].

Trancoso et al. study the memory access patterns of a TPC-D-based DSS workload [45]. 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 128K-bytes to 8M-bytes, the main results are that both large cache blocks and data prefetching 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 database (OLTP benchmark), TPC-D (DSS queries), and the Altavista search engine. They found that memory is accounting for 75% of stall time. 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 8M-bytes outer level cache, they observed that true sharing misses limit performance.

Cao et el. examine a TPC-D workload executing on a Pentium-Pro 4-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 [43]) and a DSS workload (query 6 of TPC-D [44]) [31]. Their study is based on trace-driven simulation, where traces are collected on a 4-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 4-processor system and an ILP configuration with 4-way issue, 64-entry instruction window, 4 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 [48]. 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, constituted by four-processor SMP nodes connected using a Scalable Coherent Interface based coherent interconnects. Coherence protocol is directory-based. Each node is based on a Pentium Pro processor with a 512K-byte, 4-way set associative cache. Results show that TPC-D miss rate is much lower and performance is less sensitive to L2 miss latency than in the OLTP (TPC-B) experiments. 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 database workloads running on simultaneous multithreading processors [13] - an architectural technique to hide memory and functional units latencies. The study is based on trace-driven of a four processor AlphaServer4100 and Oracle 7 DBMS as in [3]. They consider both an OLTP workload (modeled after the TPC-B [43] benchmark) and a DSS workload (query 6 of the TPC-D [44] benchmark). Results show that while DBMS workloads have larger memory footprints, there is a substantial data reuse in a small working set.

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 our 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.

4 Methodology and Workload

The methodology that we used is based on trace-driven simulation [35], [29], [46] 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 - 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 by combining multiple source traces and simulating system calls (which could also involve I/O activity), process scheduling and virtual-to-physical address translation. Finally, Trace Factory produces the references (*target* trace) furnished as input to a memory-hierarchy simulator [29]. 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]. 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 [17].

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	139324	73.05	16.89	10.06	7224	1.58	0.43
TPC-D_{12}	12	93657	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	179862	74.59	16.26	9.15	7806	1.76	0.53
DSS_{18}	18	124268	74.00	16.11	8.89	6242	1.63	0.48

The workload considered in our evaluation includes DB activity reproduced by means of an SQL server, namely PostgreSQL [49], which handles the TPC-D [44] 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.

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 [44]. It simulates an application for a wholesale supplier that manages, sells, and distributes a product worldwide. Following TPC-D specifications, we populated the database 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.

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 (TPC-D_{18} in Table 2). 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 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.

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. Our results show that solutions that allow us to achieve a higher scalability are possible for this kind of machine – compared to standard solutions, - and consequently a greater performance at a reasonable cost.

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 4 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 512K bytes and 2M bytes, while for block size we used 64 bytes and 128 bytes. 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 8 outstanding misses are allowed [26]. The base case study timings and parameter values for the simulator are summarized in Table 3.

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

5.2 Performance Metrics

To investigate the effects on the performance due to memory subsystem operations, we analyzed the causes influencing memory latency and the total execution time. The memory latency, 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 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.

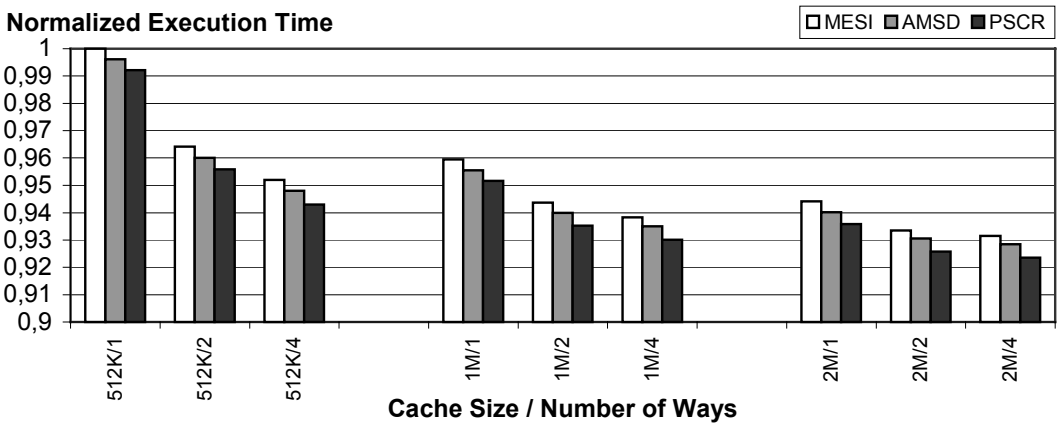

Figure 1. Normalized Execution Time versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 4 processors, 64-byte block size and a cache affinity scheduler. Execution Times are normalized with respect to the MESI – 512K, direct access architecture. PSCR presents the lowest Execution Time, whilst MESI the highest.

In our analysis, we differentiated between Cold Misses and 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 Miss is 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 an extension 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 [42], [12], [11]. Passive sharing misses are invalidation misses generated by the useless coherence maintaining of private data [18]. Clearly, these private data could appear as shared

to the coherence protocol, because of the process migration. Similarly, the coherence transactions are classified as true, false, and passive sharing transactions either in the case they are invalidation or write transactions [14].

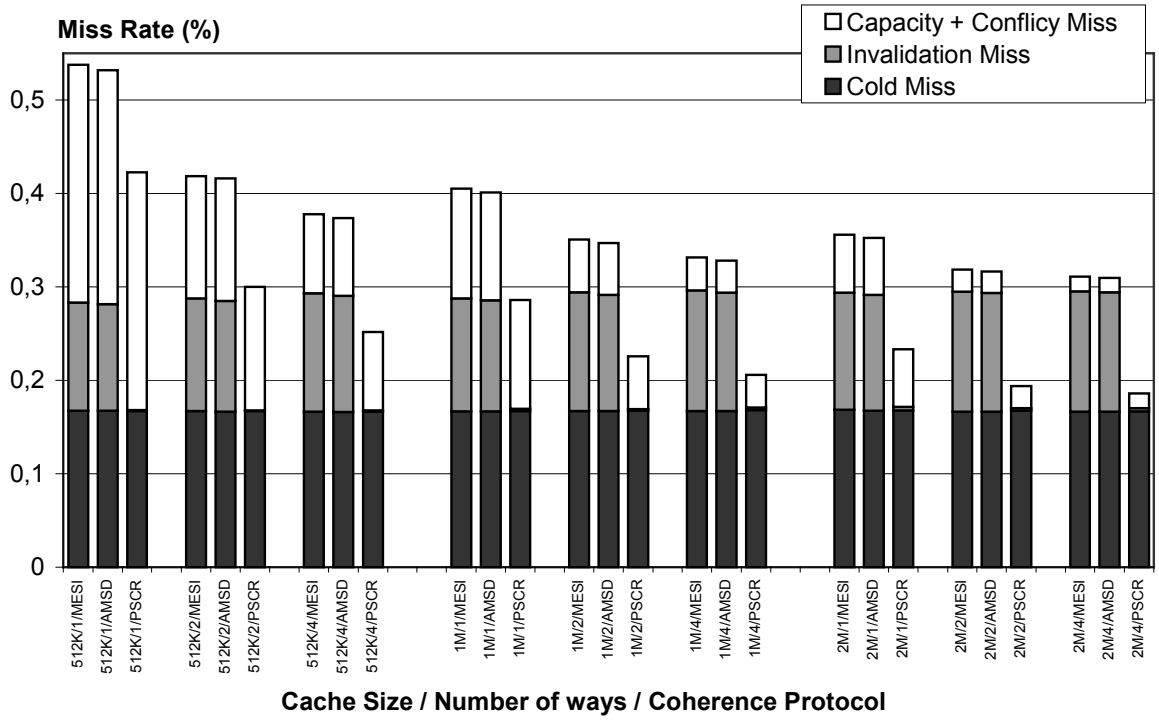

Figure 2. Breakdown of miss rate versus cache size (512 K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assumes 4 processors, affinity scheduling and 64-byte block.

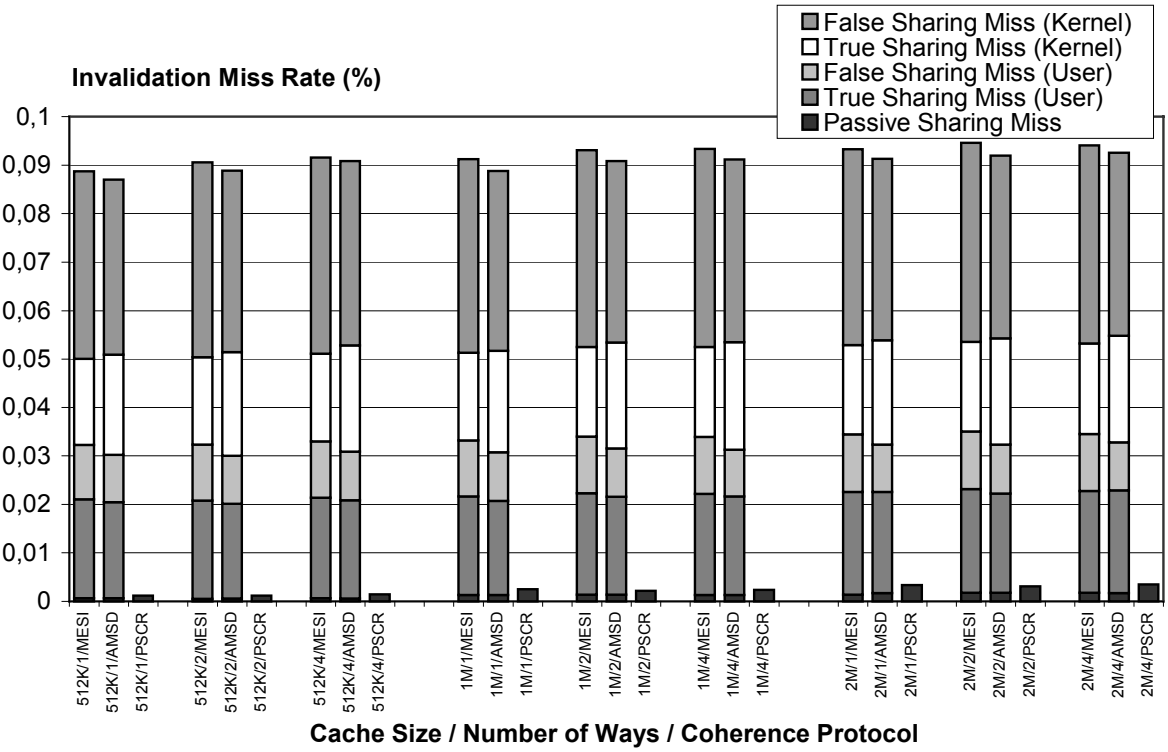

Figure 3. Breakdown of miss rate versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4) and coherence protocol (AMSD, MESI, PSCR). Data assume 4 processors, affinity scheduling and 64-byte block.

5.3 Analysis of the Reference System

We started our analysis from a 4-processor machine similar to cases used in literature [45], [3], [4], [31], running the DSS_{26} workload (Table 2).

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

Execution Time (Figure 1) is affected by the cache architecture. It decreases with larger cache sizes and/or more associativity. Anyway, this variation is limited to an 8% between the less (512K byte, one way) and most performing (2M 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 reason for this performance improvement (Figure 2) by decomposing the miss rate in term of traditional (cold, conflict and capacity) and invalidation misses. In Figure 3, we show the contribution of each kind of sharing to the Invalidation Miss Rate and, in Figure 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 (Figure 2). For cache sizes larger than 1M 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 2M bytes already capture the main working set. In fact, for cache sizes larger than 2M bytes, Cold Misses remain constant, invalidation misses increase and only Capacity+Conflict misses may decrease, but their contribution to miss rate is already minimal. The solution based on PSCR presents the lowest miss rate, and negligible invalidation misses.

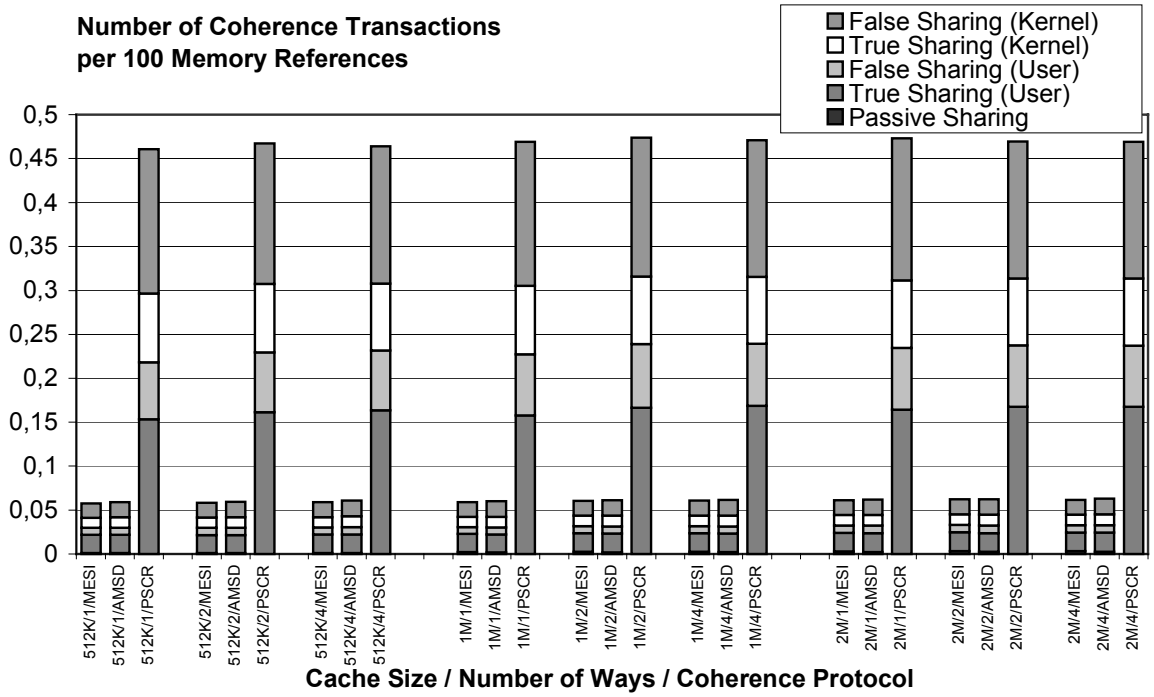

Figure 4. Number of coherence transactions versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR). Coherence transactions are invalidate transactions in MESI, and AMSD, write transactions in PSCR. Data assume 4 processors, 64-byte block size, and an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

Our analysis of coherence overhead (Figures 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 Figure 1 that PSCR is able to achieve the best performance compared with the other protocols. 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 (Figure 2).

Our conclusions, for the 4-processor configuration, agree with previous studies as for the analysis of miss rate and the effects of coherence maintaining [3], [4], [45].

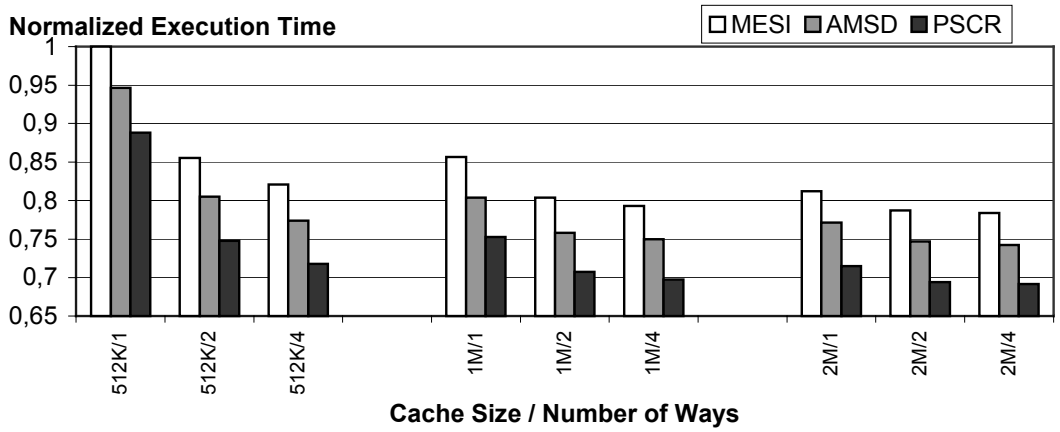

Figure 5. Normalized Execution Time versus cache size (512K, 1M, 2M bytes), number of ways (1, 2, 4), and coherence protocol (AMSD, MESI, PSCR). Data assume 16 processors, 64-byte block size, and an affinity scheduler. Execution Times are normalized with respect to the Execution Time of the MESI, 512K, direct access configuration.

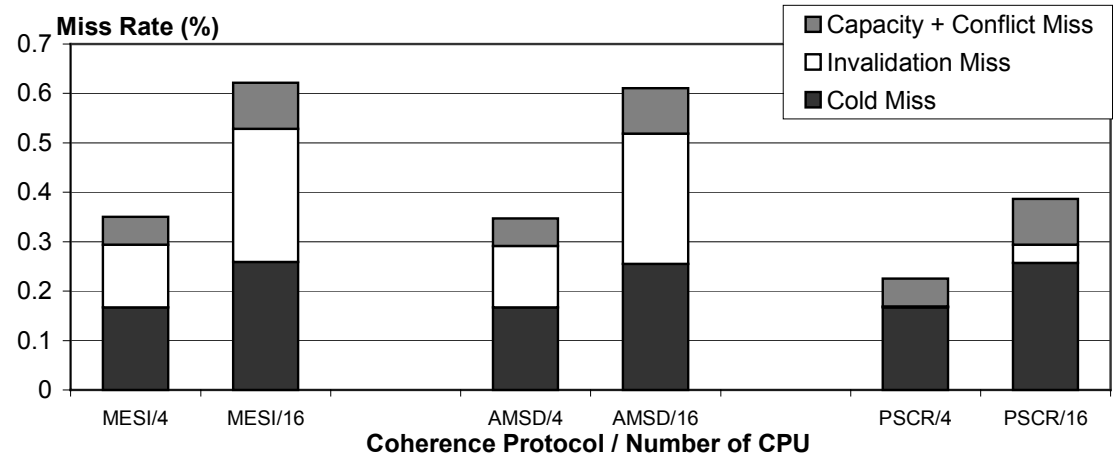

Figure 6. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume an affinity scheduler, 64-byte block, 1M-cache size two-way set associative. The higher number of processor causes more coherence misses (false plus true sharing) and more capacity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

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 and adapt the coherence protocol to the access pattern of the workload. This architecture has not been analyzed in literature, and still represents a relatively economic solution to enhance the performance.

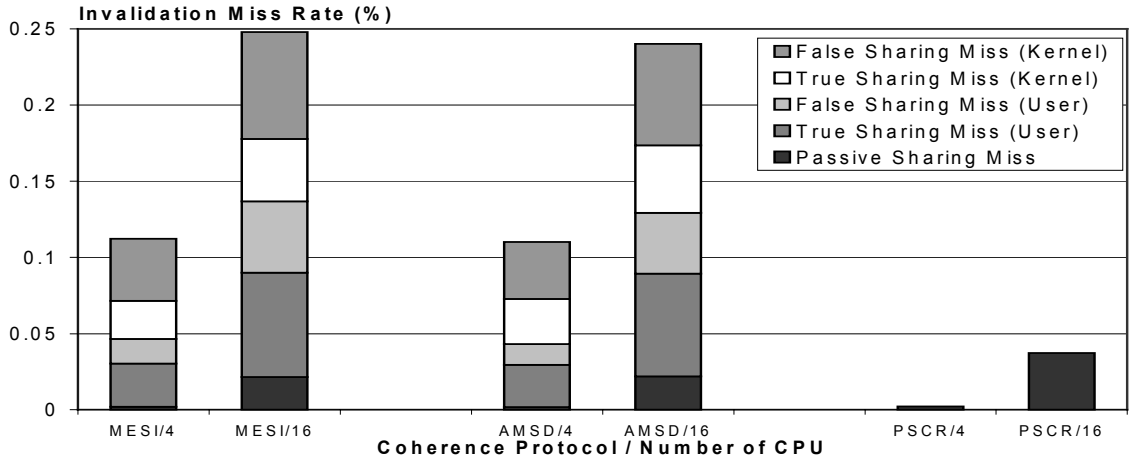

Figure 7. Breakdown of Invalidation Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4,16). Data assume, an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. Having more processors causes more coherence misses (false and true sharing). Passive sharing 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 performed on a write operation on shared data, thus less frequently than in PSCR.

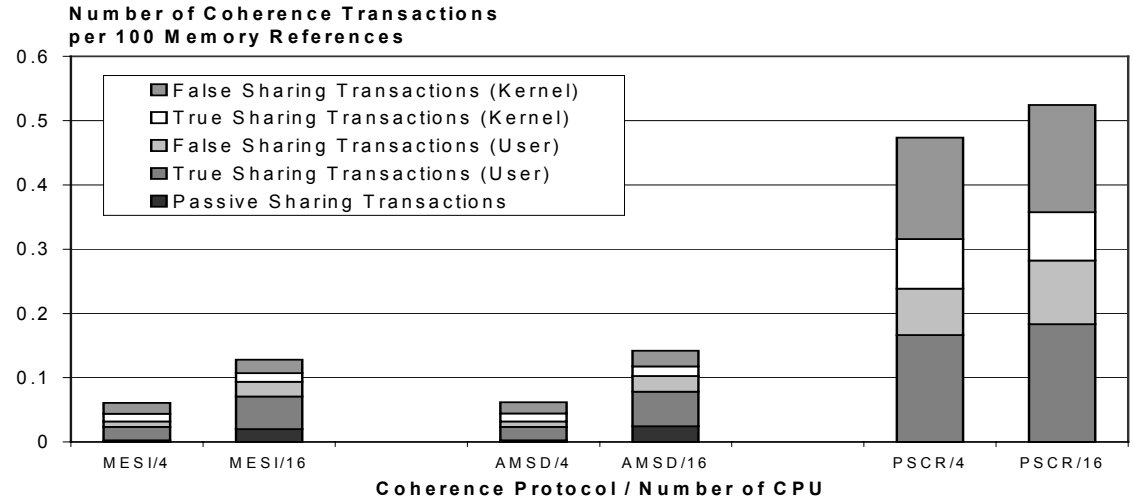

Figure 8. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and number of processors (4, 16). Data assume, an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. There is an increment in the sharing overhead in all of its components. This increment is more evident in the WI class protocols, also because there is more passive sharing overhead.

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 4-processor case and sensitivity to cache size and associativity

In Figure 5, we show the Execution Time 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 30% difference between the most and less performing configuration.

We clarify the relationship between execution Time and process migration, by showing the Miss Rate (Figure 6) and the Number of Coherence Transactions (Figure 8). In detail, we show the breakdown of the most varying components of Miss Rate (Invalidation Miss, Figure 7) and breakdown of coherence transactions (Figure 8). In the following, for the sake of clearness, we assume a 1-Mbyte-cache size, a 64-byte block size, and 2-ways.

The Execution Time 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 lower Execution Time because of its lower Miss Rate (Figure 6, and in particular Invalidation Miss Rate, Figure 7). The small variation in the Number of Coherence Transactions (Figure 8) does not weigh much on the performance. PSCR gains strongly from the reduction of Miss Rate, while it is not penalized too much by the high Number of Coherence Transactions (Figure 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 (Figure 6). In particular PSCR does not have invalidation misses due to true and false sharing, neither in the user nor in the kernel mode (Figure 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 4-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 speed-up 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 [45], [52] 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 structure needed to implement software locks [45], [52]. 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 Invalidation Misses and Transactions (in the user part) increase especially for WI protocols (Figure 7 and 8), thus speeding-up the performance of the WU solution (PSCR). However, we cannot utilize a pure WU protocol, as passive sharing dramatically decreases performances [47].

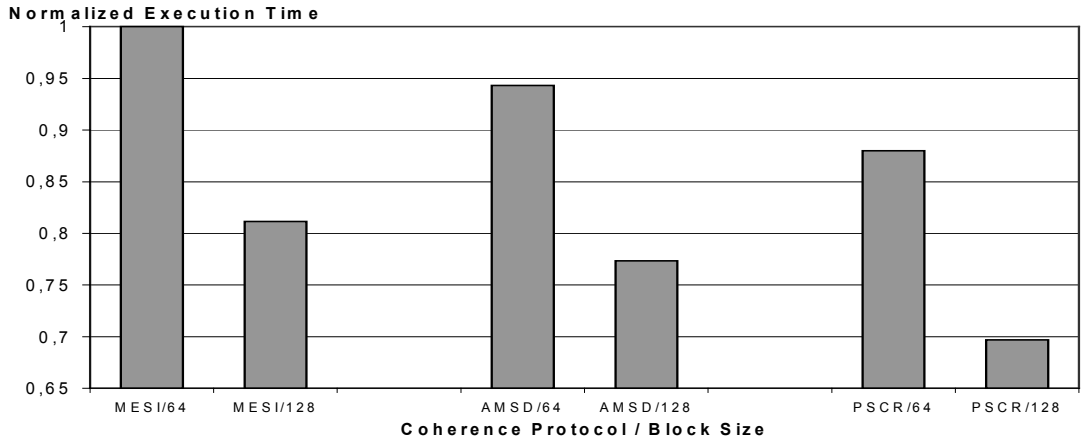

Figure 9. Normalized Execution Time versus block size size (64, 128 bytes) and coherence protocol (AMSD, MESI, PSCR). Data assume, an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. Execution Times are normalized with respect to the execution time of the MESI, 64 bytes block configuration.

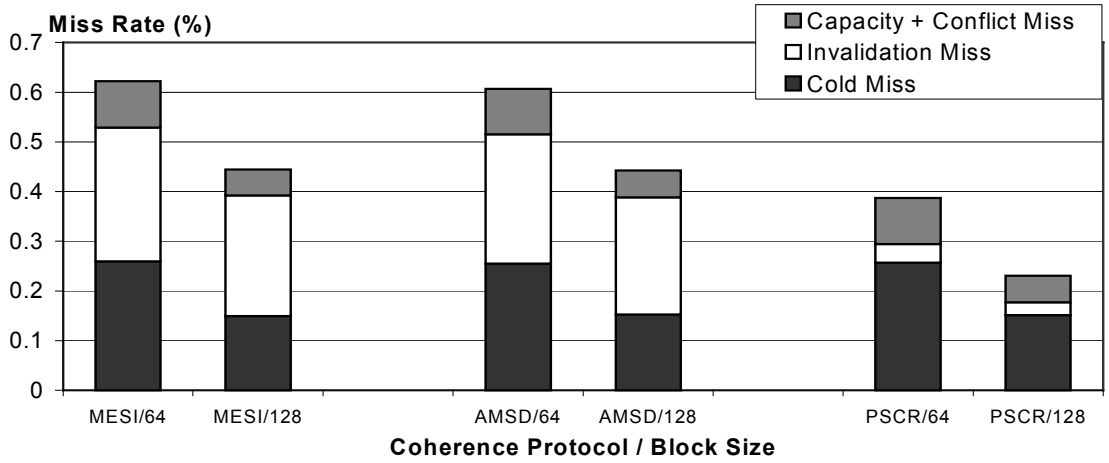

Figure 10. Breakdown of miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. There is a decrease of Cold, Capacity+Conflict miss components, and a little decrease of invalidation miss component.

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 (Figure 9). This is due to the following two reasons. First, we observe a reduction of Capacity+Conflict miss component (Figure 10), a small reduction of coherence traffic (Figure 12), and Invalidation Miss Rate (Figure 11). Secondly, in the case of 64-byte block, the system is in saturation[1] [18] for all configurations of Figure 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. When switching from 64 to 128 bytes, the decrease of the Execution Time is 27% percent for PSCR, and only a 20% for the other protocols. We observe that

[1] We recall that a shared-bus shared-memory SMP system is in saturation [18] when the performance does not increase at least of a given quantity, when we add one processor to the machine.

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" for a 16-processor CC-NUMA system was almost the same that we obtained for our cheaper shared-bus shared-memory system (figure not showed).

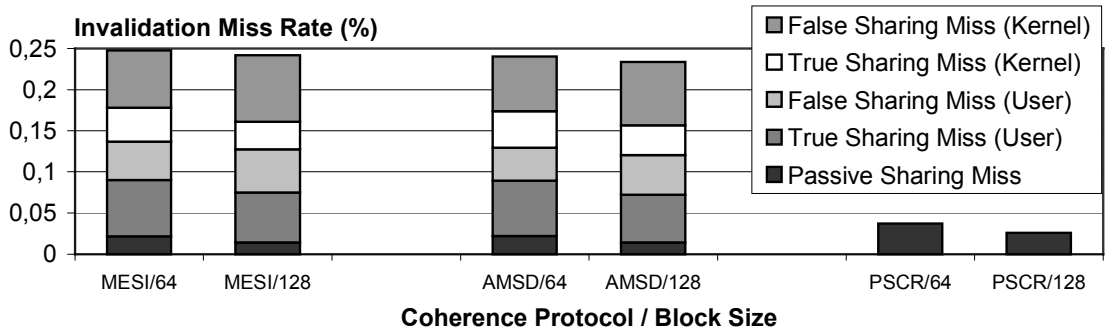

Figure 11. Breakdown of Invalidation miss rate versus coherence protocol (AMSD, MESI, PSCR) and block size (64 byte, 128 byte). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. Passive Sharing Misses decrease when increasing block size because the invalidation unit is larger.

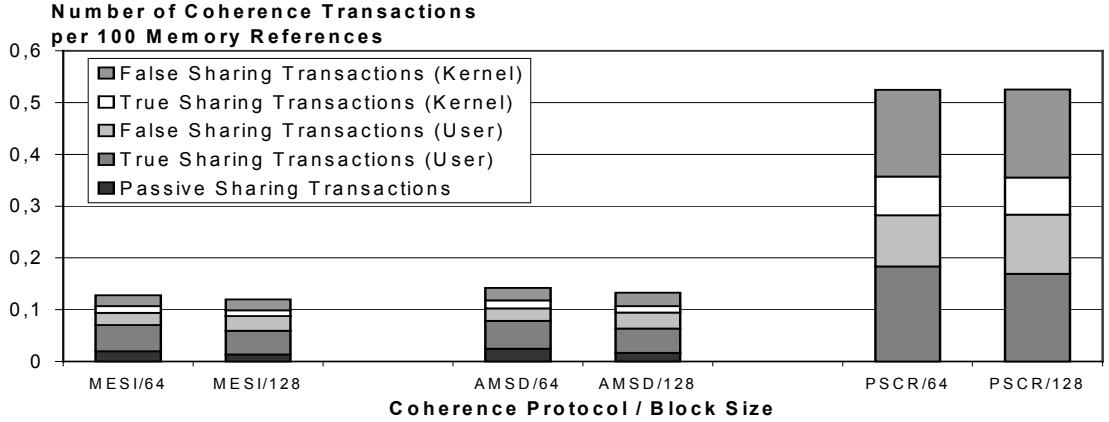

Figure 12. Number of coherence transactions versus coherence protocol (AMSD, MESI, PSCR) and block size (64, 128 byte). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches.

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 (Figures 14, 15, 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 consequence 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 (Figure 13).

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.

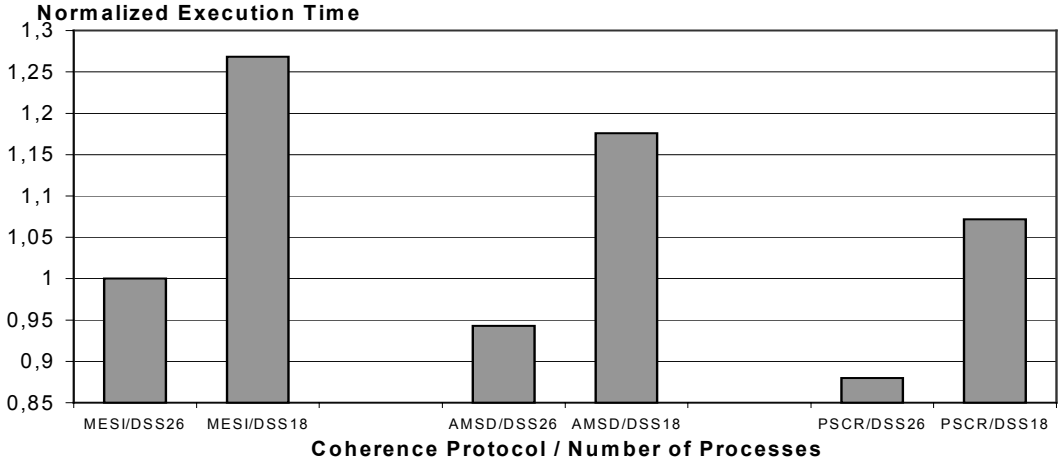

Figure 13. Normalized Execution Time versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26 – DSS18). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches.

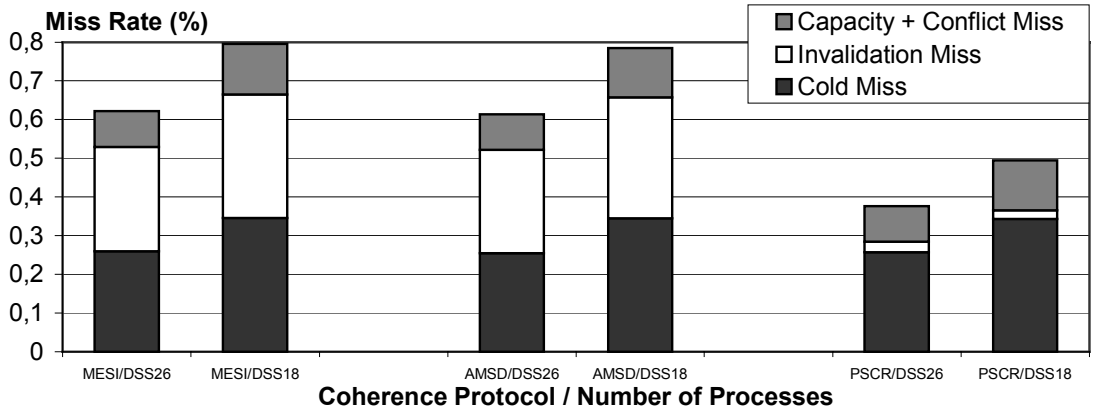

Figure 14. Breakdown of Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26 – DSS18). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. The workload DSS18 exhibits the higher miss rate, due to an increased number of Cold, Capacity, and Conflict Miss. This is a consequence of process migration, that affinity fails to mitigate.

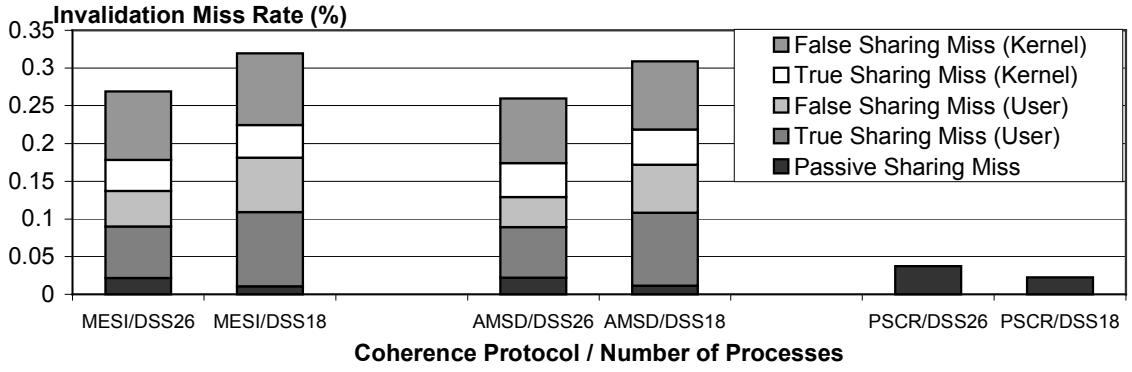

Figure 15. Breakdown of Invalidation Miss Rate versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26 – DSS18). Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. Passive sharing misses decrease, while true and false sharing misses increase. This is consequence 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).

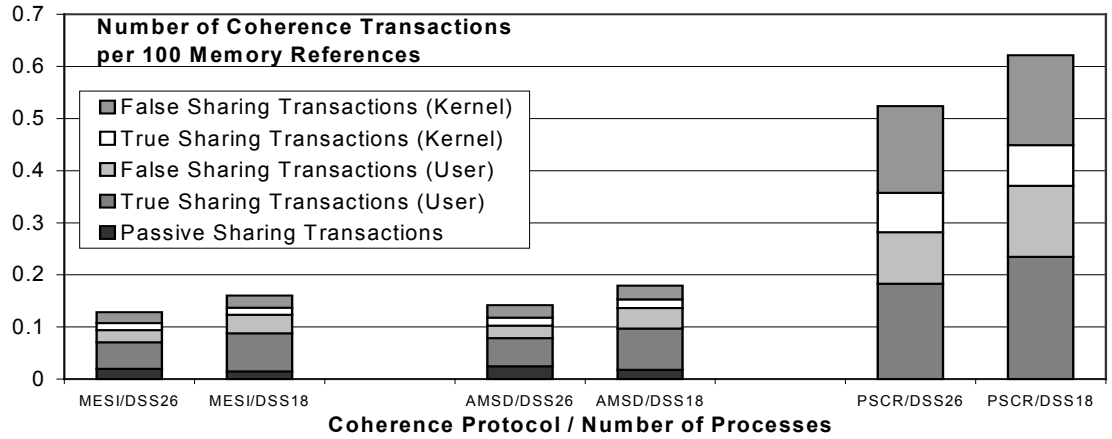

Figure 16. Breakdown of Coherence Transactions versus coherence protocol (AMSD, MESI, PSCR) and workload (DSS26 – DSS18). Data assume affinity, 64-byte block, 1M-byte 2-way set associative caches.

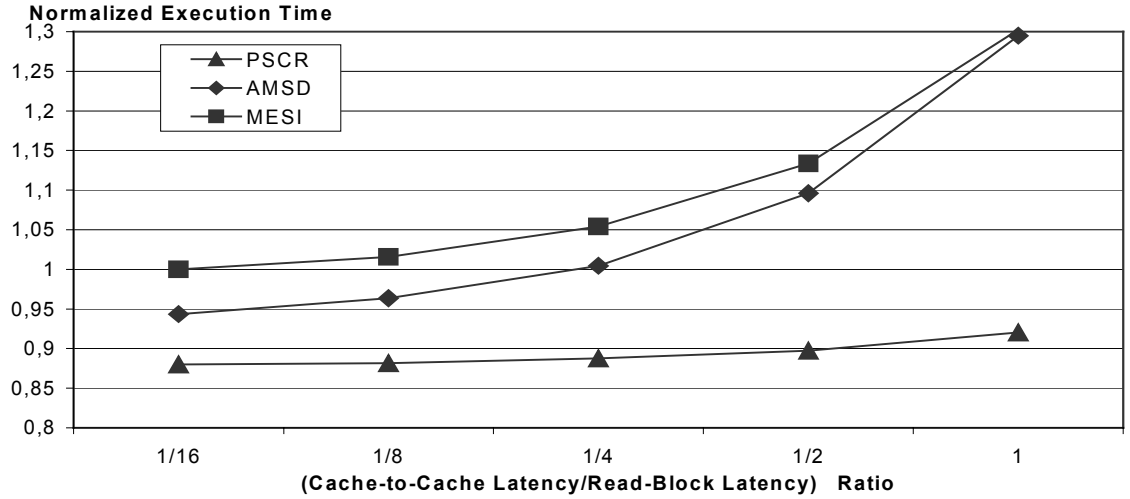

Figure 17. Normalized Execution Time versus coherence protocol (AMSD, MESI, PSCR) and the ratio between cache-to-cache transfer latency and read-block latency. Data assume an affinity scheduler, 64-byte block, 1M-byte 2-way set associative caches. Execution Times are normalized with respect to the execution time of the MESI, 1/16 configuration.

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

In our implementation of the coherence protocols, we assume that, when a data, which is missing in a local cache, is present in a remote cache, that cache supplies the data, 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 transfer [25], [31], we analyze the effect on the Execution Time of different cache-to-cache latencies. Figure 17 shows the Execution Time for the three protocols, when cache-to-cache transfer latency is varied. The cases analyzed in previous analysis assumed a ratio of 1/16 for the ratio between cache-to-cache latency and read-block latency.

Obviously, when we increase the latency, Execution Time increases. 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% with respect to the other two coherence protocols.

6 Conclusions

We evaluated the memory performance of a shared-bus shared-memory multiprocessor running a DSS workload, by considering several different choices that could improve the overall performance of the system. We considered different architectures based on the following coherence protocols: 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 4-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 2M bytes already capture the working set of such workload; ii) the kernel effects account for 50% of the coherence overhead. Previous studies that considered DSS workloads were mostly limited to 4-processor systems, did not consider the effects of process migration, and did not correlate the amount of sharing to the performance of the system.

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 4-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 PSCR allows us to build systems, whose performance is less influenced by the load condition. The use of adequate protocols can deliver to DSS system an advantage, in case of SMP systems, with high cache-to-cache latencies (up to 50% of improvement in our case).

References

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

[2] S. J. Eggers, "Simplicity Versus Accuracy in a Model of Cache Coherency Overhead". IEEE Transactions on Computers, Vol. 40, No. 8, pp. 893-906, August 1991.
[3] L.A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads". Proc. of 25th Int.l Symp. on Computer Architecture, Barcelona, Spain, pp. 3-14, June 1998.
[4] Q. Cao, J. Torrellas, P. Trancoso, J. L. Larriba-Pey, B. Knighten, and Y. Won, "Detailed characterization of a quad Pentium Pro server running TPC-D". Proc. of the Int.l Conf. on Computer Design, pp.108-115, Oct. 1999.
[5] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum, "Scheduling and Page Migration for Multiprocessor Compute Servers". Proc. of the 6th ASPLOS, pp. 12-24, Oct. 1994.
[6] J. Chapin, S. Herrod, M. Rosenblum, and A. Gupta "Memory System Performance of UNIX on CC-NUMA Multiprocessors". *ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 1-13, May 1995.
[7] A. Charlesworth, "The Sun Fireplan System Interconnect". Proc. of the Conf. on High Performance Networking and Computing (SC01), November 2001, http://www.sc2001.org/papers/pap.pap150.pdf.
[8] A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data". *Proc. 20th* Int.l Symp. *on Computer Architecture*, San Diego, California, pp. 98-108, May 1993.
[9] D. E. Culler and J. P. Singh, Parallel Computer Architecture: a Hardware/Software Approach. Morgan Kaufmann Publishers, San Francisco, CA, 1998.
[10] Z. Cvetanovic and D. Bhandarkar, "Characterization of Alpha AXP Performance Using TP and SPEC Workloads". *Proc. 21st Int.l Symp. on Computer Architecture*, pp. 60-70, Apr. 1994.
[11] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenström, "The Detection and Elimination of Useless Miss in Multiprocessor". Proc. of 20th Int.l Symp. on Computer Architecture, San Diego, CA, pp. 88-97, May 1993.
[12] S. J. Eggers, T. E. Jeremiassen, "Eliminating False Sharing". Proc. 1991 Int.l Conf. on Parallel Processing, August 1991, pp. I:377-381.
[13] S.J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen, "Simultaneous Multithreading: A Platform for Next-Generation Processors". IEEE Micro, pp. 12-19, vol. 17, no. 5, Oct. 1997.
[14] P. Foglia, "An Algorithm for the Classification of Coherence Related Overhead in Shared-Bus Shared-Memory Multiprocessors". IEEE TCCA Newsletter, pp. 40-46, Jan. 2001.
[15] K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors". Proc. of the Fourth ASPLOS, Santa Clara, California, pp. 245-357, Apr. 1991.
[16] S. J. Eggers, R. H. Katz, "Evaluating the performance of four snooping cache coherency protocols ". Proc. of 16th Annual International Symposium on Computer Architecture, pp. 2-15, Jerusalem, Israel, 1989.
[17] R. Giorgi, C. Prete, G. Prina, and L. Ricciardi, "Trace Factory: a Workload Generation Environment for Trace-Driven Simulation of Shared-Bus Multiprocessor". IEEE Concurrency, vol. 5, no. 4, pp. 54-68, October – Dec. 1997.
[18] R. Giorgi and C.A. Prete, "PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors". IEEE Trans. on Parallel and Distributed Systems, pp. 742-763, vol. 10, no. 7, July 1999.
[19] S. J. Eggers, R. H. Katz, "A Characterization of Sharing in Parallel Programs and Its Application to Coherency Protocol Evaluation". In. Proc. of 15th Annual Int. Symp. on Computer Architecture, pp. 373-382 Honolulu, Hl, May 1988.
[20] A. M. Grizzaffi Maynard, C. M. Donnelly, and B. R. Olszewski, "Contrasting characteristics and cache performance of technical and multi-user commercial workloads". Proc. of the 6th ASPLOS, pp. 158-170, Oct. 1994.
[21] E. Hagersten and M. Koster, "WildFire: A Scalable Path for SMPs". Proc. of the 5th Symp. on High-Performance Computer Architecture, pp. 172-181, Jan. 1999.
[22] J. Hennessy and D.A. Patterson, Computer Architecture: a Quantitative Approach, 3rd edition. Morgan Kaufmann Publishers, San Francisco, CA, 2002.
[23] 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.
[24] T. E. Jeremiassen, S. J. Eggers, "Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations". ACM SIGPLAN Notices, 30 (8), pp. 179-188, Aug. 1995.
[25] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker, "Performance characterization of a quad Pentium Pro SMP using OLTP workloads". Proc. of the 25th Int.l Symp. on Computer Architecture, pp. 15-26, June 1998.
[26] D. Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization". *Proc. of the 8th Int.l Symp. on Computer Architecture*, pp. 81-87, June 1981.
[27] Jack L. Lo, Luiz A. Barroso, Susan J. Eggers, Kourosh G. Gharachorloo, Henry M. Levy, Sujay S. Pareck, "An analysis of Database Workload Performance on Simultaneous Multithreaded Processors". Proc. of the 25th Annual Int.l Symp. on Computer Architecture, pp. 39-50, Barcelona, Spain, June 1998.
[28] T. Lovett, R. Clapp, "StiNG: A CC-NUMA Computer System for the Commercial MarketPlace". Proc. of the 23rd Int.l Symp. on Computer Architecture, pp. 308-317, May 1996.
[29] C.A. Prete, G. Prina, and L. Ricciardi, "A Trace Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor System". IEEE Transactions on Parallel and Distributed Systems, vol. 6 (9), pp. 915-929, Sept. 1995.
[30] 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.
[31] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. Barroso, "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors". Proc. of the 8th ASPLOS, pp. 307-318, San Jose, CA, 1998.
[32] T. Shanley and Mindshare Inc. Pentium Pro and Pentium II System Architecture. Addison Wesley, Reading, MA, 1999.
[33] P. Stenström, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing". Proc. of the 20th Annual Int.l Symp. on Computer Architecture., May 1993.
[34] P. Stenström, E. Hagersten, D.J. Li, M. Martonosi, and M. Venugopal, "Trends in Shared Memory Multiprocessing ". IEEE Computer, vol. 30, no. 12, pp. 44-50, Dec. 1997.
[35] B. Stunkel, B. Janssens, K. Fuchs, "Address Tracing for Parallel Machines". *IEEE Computer*, vol. 24 (1), pp. 31-45, Jan.1991.

[36] P. Sweazey and A. J. Smith, “A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus”. Proc. of the *13th Int.l Symp. on Computer Architecture*, pp. 414-423, June 1986.
[37] M. Tomasevic and V. Milutinovic, “The Cache Coherence Problem in Shared-Memory Multiprocessors –Hardware Solutions”. IEEE Computer Society Press, Los Alamitos, CA, Apr. 1993.
[38] 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.
[39] M. Tomasevic and V. Milutinovic, “The Word-Invalidate Cache Coherence Protocol”. Microprocessors and Microsystems, pp. 3-16, vol. 20, Mar. 1996.
[40] J. Torrellas, M. S. Lam, and J. L. Hennessy, “Share Data Placement Optimizations to Reduce Multiprocessor Cache Miss Rates”. *Proc. of 1990 Int.l Conf. on Parallel Processing,* Urbana, IL, pp. 266-270, Aug. 1990.
[41] J. Torrellas, A. Gupta, and J. Hennessy, “Characterizing the caching and synchronization performance of a multiprocessor operating system”. *Proc. 5th ASPLOS*, pp. 162-174, Sept. 1992.
[42] 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.
[43] Transaction Processing Performance Council, “TPC Benchmark B (Online Transaction Processing) Standard Specification”. June 1994. http://www.tpc.org.
[44] Transaction Processing Performance Council, “TPC Benchmark D (Decision Support) Standard Specification”. December 1995. http://www.tpc.org.
[45] P. Trancoso, J. L. Larriba-Pey, Z. Zhang, and J. Torrellas, “The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors”. Proc. of the 3rd Int.l Symp. on High Performance Computer Architecture, pp. 250-260, Los Alamitos, CA, Feb. 1997.
[46] R. Uhlig, T. Mudge, “Trace-Driven Memory Simulation: a survey”. *ACM Computing Surveys*, pp. 128-170, June 1997.
[47] C. A. Prete, G. Prina, R. Giorgi, L. Ricciardi, “Some Consideration about Passive Sharing in Shared-Memory Multiprocessors”, IEEE TCCA Newsletter, March 1997.
[48] K. C. Yeager, “The MIPS R10000 Superscalar Microprocessor”. IEEE Micro, vol. 16, n. 4, pp. 42-50, Aug. 1996.
[49] A. Yu and J. Chen, “The POSTGRES95 User Manual”. Computer Science Div., Dept. of EECS, UCB, July 1995.
[50] K. Keeton, R. Clapp, A. Nanda, “Evaluating Servers with Commercial Workloads”. IEEE Computer, Vol. 36, N. 2, pp. 29-32, Feb. 2003.
[51] A. S. Tanenbaum, “Structured Computer Organization, 4ed.”. Prentice-Hall, Inc, 2001.
[52] R. Yu, L. Bhuyan, R. Iyer, “Comparing the Memory System Performance of DSS Workloads on the HP V-Class and SGI Origin 2000”. In Proc. of the Int. Parallel and Distributed Processing Symposium, Fort Lauderdale, FL, April 2002.
[53] Ballinger C., “Relevance of the TPC-D Benchmark Queries:The Questions You Ask Every Day", NCR parallel System, http://www.tpc.org/information/other/articles/TPCDart_0197.asp.
[54] A. Ramírez, J.L. Larriba, C. Navarro, X. Serrano, J. Torrellas and M. Valero. "Code reordering of decission support systems for optimized instruction fetch". IEEE International Conference on Parallel Processing. Japan, Sept. 1999.
[55] P. Foglia, R. Giorgi, C.A. Prete: Analysis of Sharing Overhead in Shared Memory Multiprocessors. 31st IEEE Hawaii Int. Conf. on Systems, Vol. 7, pp-776-778, Kohala Coast, HL, January 1998.
[56] S. J. Eggers, R. H. Katz, “The Effect of Sharing on the Cache and Bus Performance of Parallel Programs”. Proc. 3rd ASPLOS, pp. 257-270, Boston, MA, April 1989.