Simulation study of memory performance of SMP multiprocessors running a TPC-W workload

P. Foglia, R. Giorgi and C.A. Prete

Abstract: The infrastructure to support electronic commerce is one of the areas where more processing power is needed. A multiprocessor system can offer advantages for running electronic commerce applications. The memory performance of an electronic commerce server, i.e. a system running electronic commerce applications, is evaluated in the case of shared-bus multiprocessor architecture. The software architecture of this server is based on a three-tier model and the workloads have been setup as specified by the TPC-W benchmark. The hardware configurations are: a single SMP running tiers two and three, and two SMPs each one running a single tier. The influence of memory subsystem on performance and scalability is analysed and several solutions aimed at reducing the latency of memory considered. After initial experiments, which validate the methodology, choices for cache, scheduling algorithm, and coherence protocol are explored to enhance performance and scalability. As in previous studies on shared-bus multiprocessors, it was found that the memory performance is highly influenced by cache parameters. While scaling the machine, the coherence overhead weighs more and more on the memory performance. False sharing in the kernel is among the main causes of this overhead. Unlike previous studies, passive sharing i.e. the useless sharing of the private data of the migrating processes, is shown to be an important factor that influences performance. This is especially true when multiprocessors with a higher number of processors are considered: an increase in the number of processors produces real benefits only if advanced techniques for reducing the coherence overhead are properly adopted. Scheduling techniques limiting process migration may reduce passive sharing, while restructuring techniques of the kernel data may reduce false sharing misses. However, even when process migration is reduced through cache-affinity techniques, standard coherence protocols like MESI protocol don't allow the best performance. Coherence protocols such as PSCR and AMSD produce performance benefits. PSCR, in particular, eliminates coherence overhead due to passive sharing and minimises the number of coherence misses. The adoption of PSCR and cache-affinity scheduling allows the multiprocessor scalability to be extended to 20 processors for a 128-bit shared bus and current values of main-memory-to-processor speed gap.

1 Introduction

The infrastructure to support electronic commerce [1–3], is one of the areas where more processing power is currently needed. A typical e-commerce application is based on three-tiered architecture [4–6]. On tier one, the user machine runs a client program, typically a web browser and/or Java applets; the client sends its requests to the server and receives the results for the user [5]. Tier two includes a web server that satisfies application specific requests; it takes care of task management and delivers standard services, such as transaction management and activity log. Tier three contains data and their managers, typically DBMS systems, to furnish credit-card information, catalogue information, shipping information, and user information. The elements of tiers two and three can be merged onto a single platform, or they can be distributed on several computers (clustered solution [7, 8]). The single-computer solution has the advantage of lower cost and simplified management. The distributed solution has flexibility, scalability, and fault-tolerance. In both cases the systems can be based on multiprocessor architecture [9].

We consider servers based on shared-bus shared-memory multiprocessor systems. In this case, design issues are scalability and speedup, which may be limited by memory latency and bus traffic. The use of cache memories can reduce both. Unfortunately multiple cache memories introduce the coherence problem [10–12]. The coherence protocol may have a great influence on performance. As a matter of fact, to guarantee cache coherence the protocol needs a certain number of bus transactions known as coherence overhead that add up to the basic bus traffic, which is also present in cache-based uniprocessors. Thus a design issue is to minimise coherence overhead. A commonly adopted solution to the coherence is the use of MESI protocol [13]. Consequences on memory performance may also come from the scheduling algorithm, which plays an essential role in these systems to obtain load balancing. This also generates process migration and a scarcely considered form of sharing, namely passive sharing, where a private data block of a process can become resident in multiple caches; coherence has to be enforced even on those data, thus generating useless

© IEE, 2004
IEE Proceedings online no. 20040349
doi: 10.1049/ip-cdt:20040349
Paper first received 8th July and in revised form 31st October 2003

P. Foglia and C.A. Prete are with Dipartimento di Ingegneria dell'Informazione, Facolta' di Ingegneria, Universita' di Pisa, Via Diotisalvi, 2 – 56126 PISA (Italy)
R. Giorgi is with Dipartimento di Ingegneria dell'Informazione, Facolta' di Ingegneria, Universita' di Siena, Via Roma, 56 – 53100 SIENA (Italy)

coherence-related overhead, which may limit system performance [14, 15]. This sharing is not to be confused with other types of avoidable sharing like false sharing [16].

The aim of this paper is to analyse the influence of the memory subsystem on the performance and scalability of an e-commerce server based on a shared-bus shared-memory multiprocessor architecture. In our evaluation the workloads have been setup as it is specified in the TPC-W benchmark [17]. TPC-W simulates the activities of a business-oriented transactional web server. That workload exercises the breadth of system components associated with such environments. The components we used in our implementation are: the Apache daemon [18], which handles HTTP requests; several UNIX utilities, which both access file system and interface the various programs running on the system; and an SQL server, namely PostgreSQL [19], which handles database queries. We considered the cases when the workload is running on a single multiprocessor and when it is distributed on two multiprocessors (one for each server tier). The performance evaluation methodology relies on enhanced trace-driven simulation by means of the trace factory environment [20, 21]. We considered the major bottlenecks of the memory subsystem of this architecture. Starting from a reference case we explored different architectural choices as for cache, number of processors, scheduling algorithm, and coherence protocol.

We found, similar to other works [22, 23], that large caches and cache-affinity improve the performance, but already with a 2 Mbyte cache size the performance increase is limited and it is mostly influenced by the coherence overhead. This situation is more evident as the number of processors is increased.

The results we obtained show that in these systems MESI is not an optimal choice as coherence protocol, mainly because of its nonselective invalidation mechanism. The use of a selective invalidation line in PSCR [14] can recover the efficiency of a write-update policy to maintain coherence. This paper shows how this kind of protocol is effective for some test-beds used in e-commerce server through a TPC-W benchmark.

2 Memory subsystem issues and their implications on system performance

From the user's point of view, a multiprocessor system could be employed to speed-up e-commerce workloads with the goal of serving more requests per time unit. At the same time, the system designer tries to achieve a higher scalability to offer a wide range of processing power using the same architecture.

The memory subsystem plays an essential role in achieving these goals [24–27]. To enhance the performance of the memory subsystem we need to minimise the memory latency, also in current generations of microprocessor which incorporate techniques to aggressively exploit instruction-level parallelism [28].

A typical solution, with diminishing returns [29], to reduce bus traffic is to include large cache memories. Caches contribute to hide memory latency and to reduce traffic on the bus [30–32], but they cause the coherence problem [10–12]. Two or more processors may store a copy of the same memory block in their private cache. When one of them performs a write operation on a location within that block, a coherence protocol is required to guarantee that each subsequent read operation by any processor may get the updated value.

The protocol activity requires a number of bus transactions (coherence overhead) to keep the copies coherent, which add up to the basic bus traffic as present in cache-based uniprocessors. In shared-memory systems, another important factor of the bus traffic is due to misses generated as a consequence of coherence handling (invalidaton misses).

The coherence overhead depends on the kind of data sharing exhibited by the program. Three different types of data sharing can be observed: true sharing [16], which occurs when the same cached data item is referenced by processes running on different processors, false sharing [16], which occurs when several processes running on different processors reference different data items belonging to the same memory block, and passive [15] or process-migration [33] 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. While true sharing is unavoidable, the other two forms of sharing are useless. The relevant overhead they produce can be reduced [16, 34–38] and possibly avoided [14].

True sharing is generated by the process communication and access to kernel data. Its overhead depends on the communication mechanism, the parallelism of the application, and the programming style. A typical example of true sharing is the use of data structures belonging to critical sections, where processes read and modify data in an exclusive manner [37, 39, 40]. False sharing is due to the mismatch between the program data size and the size of cache block. It is influenced by the variable-allocation strategies used by the compiler and the access patterns to these variables performed by the processes [16]. Passive sharing is not directly connected with the programming style or compiler strategies, but it is due to private data that appear as shared data to the coherence scheme when a process, which has been pre-empted on a certain processor, is rescheduled to a different processor (process migration). Migration of processes depends on the number of ready processes, and the scheduling policy of the operating system.

Process migration is an important issue to the global performance [24, 26, 41–43] and particularly in our system since workloads like e-commerce applications generate a number of processes that we want to assign dynamically to available processors to achieve load balancing without requiring programmer's efforts. However, when a process resumes its execution on a new processor, a miss peak is generated (context-switch misses) to reload its working set. Moreover, a coherence-transaction peak may be also generated because of passive sharing. To limit passive sharing overhead it is possible to invalidate all private data belonging to a process on context-switches, but doing so eliminates the opportunity for a given process to reutilise the working set previously loaded on a certain cache when, after a migration, that process is scheduled again on the same processor in the next scheduling time slices. In such a condition, that process generates again a load-miss burst that reduces performance. We evaluated the impact of the invalidation on context switches in our initial works, finding the worst results for every protocol, and concluding that the best strategy to eliminate passive sharing overhead is to invalidate private data only when they are accessed on a different processor [15].

Using cache-affinity scheduling [22] is a typical solution to reduce migration related problems [23]. Some specialised coherence protocols as well can alleviate passive sharing. Besides the commonly used MESI we therefore considered further protocols which intervene both on the migration effects and the coherence overhead: AMSD [34, 37] and PSCR [14], which we describe.

A *de-facto* standard for maintaining coherence is the MESI protocol [13]. MESI, a write invalidate protocol, is a widely employed solution and it is implemented with different flavours on most high-performance microprocessors [44, 45] (like AMD K5, K6, Athlon-MP, PowerPC series, Power4, SUN UltraSparc II, III-Cu, MIPS R10k, Intel Pentium, Pentium Pro, Pentium-II, Pentium-III, Pentium 4, IA-64/Itanium, Xeon-MP). Each implementation differs in some details and some implementations only use three states (MIPS R20k), while others use five (MOESI Athlon-64 [46]) or even seven states (Power4, [47]). Although most industrial solutions are based on MESI, significant contributions from the academics continue to explore cache coherence solutions [48–50].

Besides classical MESI protocol states the implementation of MESI that we considered [45] uses the following bus transactions: *read-block* (to fetch a block); *read-and-invalidate-block* (to fetch a block and invalidate all the copies in other caches); *invalidate* (to invalidate all the copies in other caches); and *update-block* (to write back dirty copies when they have to be destroyed for replacement). The drawback of the invalidation transaction used to obtain coherency is the possible need to reload an invalidated copy if it is used again by a remote processor, 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.

AMSD is designed for migratory sharing, which is a kind of true sharing that is characterised by the exclusive use of data by a certain processor for a long time interval. Typically this happens when the control over these data migrates from one process to another running on different processors. Migratory sharing patterns were also identified in kernel data structures [43]. 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 (the four basic states of MESI are augmented with three new states to detect migratory sharing copy.) Although designed for migratory sharing, AMSD may have some beneficial effects also on passive sharing too. As in the case of MESI, AMSD coherence overhead is due to invalidate transactions and invalidation misses.

PSCR (passive shared copy removal) adopts a selective invalidation scheme for the private data, and it uses the *write-update* scheme for the shared data [14]. A cached copy belonging to a process private area is invalidated locally as soon as another processor fetches the same block. The scheme, based on *write-update* RST protocol [10], requires five states (one more than MESI) and an additional bus line compared with MESI. This technique eliminates passive sharing overhead without the drawbacks of private data invalidation on context switch. Invalidate transactions are eliminated and coherence overhead is due to write transactions. It has been observed that the major contributing kernel data structures are those that mostly store the per-process private state. Namely, they are the kernel stack, components of the user structure, and the process table [43]. Since they may be considered private data, if these data appear as shared to the system it is because the owner process migrates among CPUs. Thus it would be possible to extend the selective invalidation mechanism of PSCR to this kind of kernel data too.

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

3 E-commerce server setup and workload definition

We considered a typical e-commerce system based on the three-tiered model as introduced by Edwards *et al.* [5] (Fig. 1): tier one is constituted by the e-commerce clients (typically web browsers), which access the server over the internet; tier two is constituted by the web server, the transaction management process, and application processes (which also provide accounting and auditing); and tier three is constituted by data and their managers.

The activity of e-commerce systems typically involves data scan (to access a product list, product features, credit card information, shipping information), update (to update customer status and activity status) and transactions (buying products, making payments, for example). These activities involve the interaction between tiers according to the following model: the user (i.e. the client, tier one) sends its requests by means of a web-browser. A process (a daemon, which constitutes part of the tier two) waits for the user request and sends the request to a child application process. Then the daemon waits for new requests while the child process handles the incoming request. This activity may require accessing HTML pages and/or invoking service processes at tier three.

The processing load of tiers two and three can be distributed on several computational nodes by using different solutions. A single node can be used or, as the processing load increases, more nodes connected by a high-speed network (cluster) can be used. Load distribution can be obtained by allocating different software components on the available processors or by distributing processing requests among the nodes where the whole software architecture is replicated by using clustering techniques and load balancing [7, 8, 51]. In both cases, a node may be a multiprocessor system.

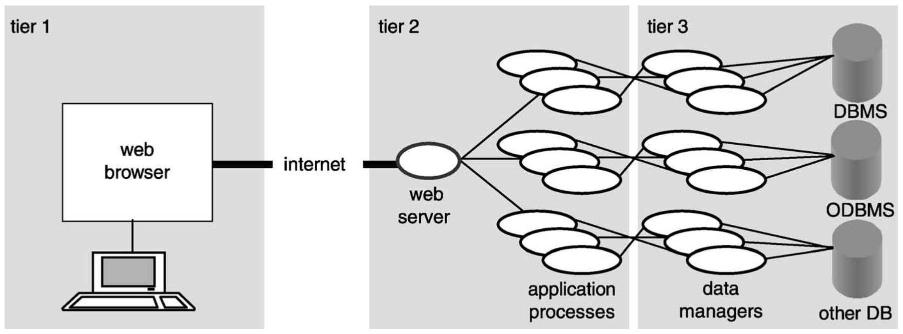

Fig. 1 *Typical architecture of e-commerce system*
Tier one consists of client program, typically a web-browser
Tier two consists of web server, transaction processing monitor that enables requests and spawns child application processes
In tier three service processes furnish standard services (database or other legacy applications)

As for workloads, we implemented a software architecture based on the following freeware components. The system front-end (part of tier two) includes an Apache HTTP server [18, 52]. Client requests involving database activities are forwarded to the database management system (DBMS) PostgreSQL [19]. PostgreSQL is constituted of a front-end (also part of tier two), which intercepts requests, and a backend (part of tier three), which executes queries.

We configured the Apache server so that it spawns a minimum of eight idle processes, a maximum of 40 idle processes. The number of requests that a child can process before dying is limited to 100. PostgreSQL is a shareware DBMS, initially developed at University of California, Berkeley. It utilises shared memory to cache frequently accessed data, indices, and locking structures [27].

We considered a general case of a workload not depending on a specific e-commerce system. To this end we followed the TPC-W benchmark [17], which specifies how to simulate the activities of a business-oriented transactional web server and exercises the breadth of system components.

The application portrayed by the benchmark is a retail store with customer browse-and-order scenario. The customer visits the storefront in the company web site to look at products, find information, place an order, or request the status of an existing order. The visitor activity is mostly devoted to browse the site. Some percentage of all visits results in submitting a new order. The activity of a site client is described through 14 possible web interactions specified by the benchmark ('home', when connecting to the server, 'shopping cart', 'customer registration', 'buy request', 'buy confirm', 'order inquiry', 'order display', 'search request', 'search result', 'new products', 'best sellers', 'product detail', and two administrative web interactions). Each web interaction describes both the web-page content and the values to submit in the case of forms. These values are generally the inputs of queries. A static diagram specifies the sequence of web interactions. Among web objects we can have JPEG and GIF images of various dimensions. The TPC-W database system is organised by means of the following tables: 'ITEM', 'country', 'author', 'customer', 'orders', 'order_line', 'cc_xacts', 'address'.

The e-commerce activity of TPC-W is performed by a number of entities, denominated emulated browsers (EBs), which dynamically produce typical client activities. Each activity generates a number of web interactions and consequently the exchange of a number of web objects. The number and the type of these exchanges refer to the benchmark implementation. In our experiments 20 EB clients run on several workstations connected to the simulated server via a LAN. In the benchmark, the number of clients and the number of entries of ITEM table define the dimension and the initial population of the DB. This population varies during the execution of the benchmark. In our case the number of entries in the ITEM table is about 100 K. This corresponds to a dimension of 80 Mbytes for the ITEM table and a total dimension for the DB of 200 Mbytes.

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`, `gzip`, and `rm`) have been considered in our workload setup. These utilities are important because they model the 'glue' activity of the middleware software. Their implementation comes from the standard GNU code available in Linux distributions [53]. These utilities do not have shared data and thus they increase the effects of process migration, and they may interfere with shared data and code footprint of other applications.

We defined three experiments to consider three possible scenarios of multiprocessor use for e-commerce applications. In the first experiment all the server components are allocated on a single multiprocessor. In the other two, we allocated the web-server subsystem and the DBMS subsystem on separated single multiprocessors.

4 Methodology

Our methodology is based on enhanced trace-driven simulation [21, 54, 55], and on the simulation of the three kernel activities that mostly affect performance: system calls, process scheduling, and virtual-to-physical memory address translation. We used the Trace Factory environment [20]. This methodology is aimed at producing a process trace, i.e. a sequence of user references, system-call positions and references (from a Linux kernel [56]), and synchronisation events in the case of multiprocess programs. (Our traces also contain particular events that we called 'synchronisation events'. This implies that, for example, spin-loops do not dilute the source trace, but they are just filtered out and substituted by synchronisation events [20], which are crucial during the simulation of the whole multiprogrammed workload [57].) This is done for each process belonging to the workload by means of a tracing tool and *ad hoc* instrumentation code. Then the environment models the execution of workloads by combining multiple process traces and the references of system calls, and by simulating process scheduling and virtual-to-physical memory address translation. As for virtual memory and process scheduling, we only simulated their effects on the processes. Finally, Trace Factory furnishes the references to a memory-hierarchy simulator [21] on demand. All these aspects are important to achieve accuracy in simulation, especially in multiprogrammed and OS intensive workloads running on modern ILP processors [50, 58], as in the case of our analysis. Based on the Mauer classification [58], our multiprocessor simulator utilises traces that include OS code and events, so it appears to be a static system simulator and then a functional-first simulator. However, the events are scheduled based on both functional and timing components (which are tightly coupled in our environment), so it can be better classified as a timing and function simulator, where some are decoupled based on system timing.

On the simulated target system, process scheduling is modelled by dynamically assigning a ready process to a processor. The process scheduling is driven by time-slice for single process applications, while it is driven by time-slice and synchronisation events for multiprocess applications. Clearly, I/O activities are taken into account through system calls, which could be blocking: in such a case the scheduler is taking adequate actions to schedule a new process. Virtual-to-physical memory address translation is modelled by mapping sequential virtual pages into nonsequential physical pages.

Using this methodology, the TPC-W benchmark specification, and the freeware components, we generated three workloads (EC-server, DB and Web-server). We traced the execution of the workload programs handling 100 web interactions in a specific time interval corresponding to 130 millions of references. Our methodology, to stop the execution, is similar to the one presented in [50]. To have comparable results, all the workloads are setup with 26 ready processes. The EC-server workload consists of 13

Table 1: Statistics of source traces for some UNIX utilities in the case of 32-byte block size

Application	Distinct blocks	Code (%)	Data (%)	Data write (%)
AWK	9876	76.23	23.77	8.83
CP	5432	77.21	22.79	8.88
GZIP	7123	82.32	17.68	2.77
RM	2655	86.18	13.82	2.11
LS-AR	5860	80.23	19.77	5.79

processes spawned by the Apache daemon, eight by PostgreSQL, and five UNIX utilities. Table 1 (for the single process applications) and Table 2 (for the multiprocess ones) contain some statistics of the traces used to generate the workloads for a 32-byte block size. The DB workload consists of 21 active processes from PostgreSQL application while five processes are UNIX utilities. The Web-server workload is constituted of 26 Apache processes. Table 3 summarises the statistics of the resulting workloads.

The simulator included in Trace Factory characterises a shared-bus multiprocessor in terms of CPU, cache, and bus parameters. The CPU parameters are the number of clock cycles for a read/write CPU operation. The simulated processors are MIPS R10000 [59], based on out-of-order paradigm (issue width is four, instruction window size is 64, and other internal parameters have standard values, in agreement with [60]); the page size is 4 Kbyte. The cache parameters are cache size, block size, and associativity. The caches are nonblocking ones and up to eight outstanding misses are allowed [61]. The cache block replacement policy is LRU (least recently used).

Each processor uses a write buffer and implements a relaxed model of memory consistency, in particular the processor consistency model [31, 62, 63] with a two-word store buffer. Cache controllers are fully simulated, both from the timing and functional behaviours [64]. Finally the bus parameters are the number of CPU clock cycles for each kind of transaction: write, invalidation, update-block, memory-to-cache read-block, cache-to-cache read-block, memory-to-cache read-and-invalidate-block, and cache-to-cache read-and-invalidate-block. The bus supports transaction splitting and has a 128-bit fixed width.

The bus timings relative to our experiments are reported in Table 4. The reported values are calculated by taking into consideration memory latencies. These values are chosen in line with current high-performance processor's typical latencies, *cf.* Ultrasparc III-Cu [65], Power4 [66]; Scheduler

Table 2: Statistics of multiprocess application source traces (Apache and PostgreSQL) in the case of 32-byte block size

Workload	Number of processes	Distinct blocks	Code (%)	Data (%) Access	Data (%) Write	Shared blocks	Shared data (%) Access	Shared data (%) Write
Apache	13	34311	73.84	26.16	6.99	1105	1.84	0.6
PostgreSQL (DB)	21	259023	72.82	27.18	10.12	10515	2.27	0.67
PostgreSQL (EC-server)	8	24141	71.94	28.06	9.89	5838	2.70	0.79

Table 3: Statistics of workloads in the case of 32-byte block size

Workload	Number of processes	Distinct blocks	Code (%)	Data (%) Access	Data (%) Write	Shared blocks	Shared data (%) Access	Shared data (%) Write
EC-server	26	112183	75.49	24.51	7.39	6101	1.68	0.54
DB	26	330279	74.59	25.41	9.15	10516	1.77	0.53
Web-server	26	72550	73.94	26.06	6.92	1108	1.54	0.49

Table 4: Numerical values of timing parameters for multiprocessor simulator (timings in clock cycles)

Class	Parameter	Timing (byte) 32	64	128	256
CPU	Read/Write operation	2	2	2	2
Bus	Invalidate transaction	5	5	5	5
	Write transaction	5	5	5	5
	Memory-to-cache read-block transaction	68	72	80	96
	Memory-to-cache read-and-invalidate-block transaction	68	72	80	96
	Cache-to-cache read-block transaction	12	16	24	40
	Cache-to-cache read-and-invalidate-block transaction	12	16	24	40
	Update-block transaction	8	12	20	36

supports two scheduling policies: a random policy and cache-affinity [23]; Scheduler time-slice is equivalent to about 200,000 references.

A great number of statistics are obtained by the simulator. These include the miss rate, number of write transactions per 100 memory operations, bus utilisation ratio, and global system power (GSP). The most significant metric for our evaluation of the machine is the GSP, which includes the combined effects of processor architecture and memory hierarchy. We briefly recall the definition of the global system power metric [10, 14, 20, 21, 67]: $GSP = \Sigma U_{cpu}$, where $U_{cpu} = (T_{cpu} - T_{delay})/T_{cpu}$. T_{cpu} is the time needed to execute the workload, and T_{delay} is the total CPU delay time due to memory latency. GSP provides a measurement of the processing power of a multiprocessor system, which is more realistic in such cases as ours when we have a workload made of applications running on the system in a continuous and cyclical way (and not applications where we want to measure execution time from beginning to end).

We also measured the scalability of the system by using this definition: we say that a multiprocessor system is scalable up to N processors, if N is the number of processors that causes the GSP to drop by more than 0.5 when switching from an N- to an $(N+1)$-processor machine. This definition is equivalent to the definition of 'critical point' in [14].

The simulator also classifies the coherence overhead by analysing the access patterns to shared data (true, false [16], and passive sharing [15, 33]. In particular, it classifies coherence transactions (write or invalidate) and misses due to a previous invalidate transaction (invalidation misses). An operation issued to access a cache block generates an invalidation-miss if the missing copy is still present in cache, but has been invalidated because of an invalidate transaction. The type of access pattern to the cache block determines the type of the coherence transaction or invalidation-miss. The classification [68] is based on an existing algorithm [69], extended to the case of passive sharing, finite-size caches, and process migration.

5 Simulation results

We analyse the memory subsystem of a shared-bus multiprocessor employed to run e-commerce applications. In particular, we wish to show how the performance and the scalability of the whole system can be improved by using a careful design of the memory subsystem.

First consider the EC-server workload running on a single multiprocessor. Afterwards, we will consider the case in which the two server tiers are each running on a different node. For the single multiprocessor case, we analysed 4-, 8-, 12-, 16-processor configurations for the shared-bus multiprocessor. We varied some common cache parameters (e.g. cache size, block size) and considered solutions for the scheduling policy and the coherence protocol that influence the performance of our system more closely, once the machine is pushed to higher performance levels.

In our performance analysis we use the global system power (GSP) and a measure of the scalability of the multiprocessor. These two metrics are discussed in detail in Section 4. We use miss rate and coherence transaction graphs to explain the performance results and to identify the main sources of memory overhead, according to the methodology already used in [31].

5.1 Analysis of performance limits of 4- and 8-processor configurations

We consider two configurations with a relatively low number of processors as we vary cache parameters (cache size and block size), and we highlight the main causes of performance limits. The results with different associativity are not shown, since they are in line with general theory of caches.

5.1.1 Four-processor configuration:

We started our analysis with a commonly used four-processor configuration. Previous studies also considered four-processor machines [24–27]. The caches have a 32-byte block size, while capacity has been varied between 128 Kbytes and 2 Mbytes, and for the associativity we considered two ways. The system adopts MESI protocol, discussed in Section 2. The scheduler relies on a random scheduling policy. As a result of this experiment we found that: GPS increases with cache size and associativity, ranging from 2.6 to 3.3; the scalability, calculated as in Section 4, is four processors in the 128 Kbyte case and nine processors in the 2 Mbyte case; the invalidation misses becomes more and more important when increasing cache size, ranging from a 5% of the total misses in the 128 Kbyte case to 20% of the total misses in the 2 Mbyte case.

The first two results reported are due to the reduction of bus traffic, which is decreasing from 72 to 35%, as the cache size and associativity increase. In this case, the main part of traffic is due to classical misses (sum of cold, conflict, and capacity misses [31], reported in Fig. 2 as 'other misses') and coherence traffic, constituted of invalidation misses (first bar in each couple of bars in Fig. 2) and invalidate transactions (Fig. 3). (*Update* transactions are only a negligible part of the bus traffic: very low for large cache sizes and lower than 10% of *read-block* transactions even in the worst case considered here, which is for smaller caches.) The reduction of bus traffic is due to the lower miss rate and in particular to the lower 'other-miss' rate. As the 'other misses' are reduced through the increase of cache size and associativity, the coherence traffic weighs more and more on the total traffic (third result reported), since the invalidation misses not only do not decrease but, on the contrary, slightly increase (Fig. 2, and more in detail Fig. 3). This effect is more evident as we increase the block size and discussed in detail in Section 5.1.3. At the same time, the coherence transactions increase (see Fig. 5).

The main conclusion from this first experiment is that with such a configuration of the machine, intervening on the cache capacity and associativity can increase the GSP and scalability, as already shown in the literature, but other factors like passive sharing already start to appear. On the

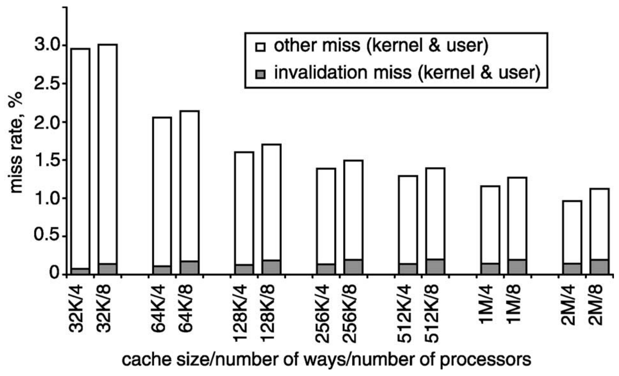

Fig. 2 *Breakdown of miss rate against cache size in case of 4 and 8 processors, random scheduling policy, 32-byte block size, and two-way set associative cache*

Miss rate decreases when increasing cache size and associativity (values not shown), although invalidation misses slightly increase. All miss components increase when switching from 4- to 8-processor configurations

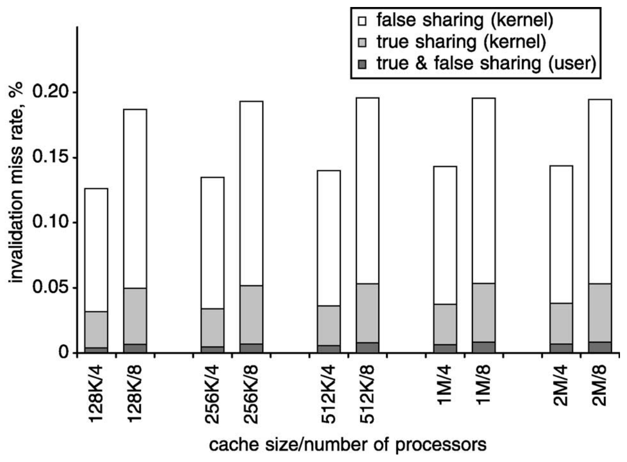

Fig. 3 *Breakdown of invalidation miss rate against cache size in case of 4 and 8 processors, random scheduling policy, 32-byte block size for two-way set associative cache*

Each component of invalidation misses increases when switching to 8-processor configuration

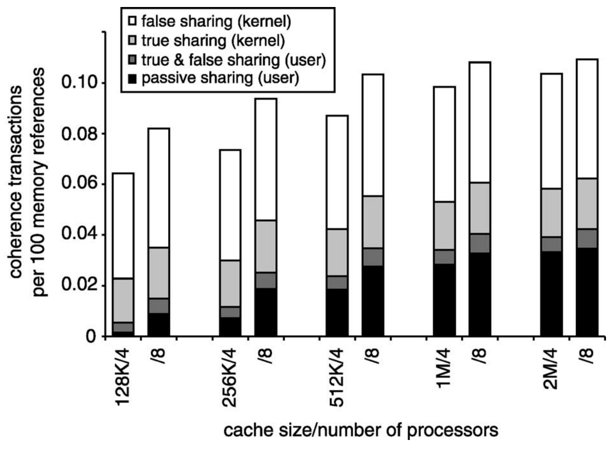

Fig. 5 *Number of coherence transactions (invalidate transactions) against cache size in case of 4 and 8 processors, random scheduling policy, 32-byte block size and two-way set associative cache*

Each component of overhead increases in 8-processor configuration. Passive sharing component increases more significantly

other hand, we have to observe that the 'other misses' cannot be cleared completely: to increase the performance and scalability further we need to reduce the coherence overhead.

5.1.2 Eight-processor configuration:

We found that the previous configuration would be scalable up to nine processors, so we wish to analyse the consequences of doubling the number of processors. Figure 4 shows that the machine performance does not double as we switch to eight-processors. The reasons are: the miss-rate increases in comparison with the four-processor case; the number of coherence transactions particularly increases in terms of false and passive sharing (Fig. 5). In this configuration the higher number of processors produces a greater process migration and an increase of data sharing. These are the main causes of both the observed effects.

The miss-rate increase is due both to the increased invalidation miss-rate (Fig. 3), in turn due to the higher parallelism of the system (causing a higher probability that a shared block is used by a higher number of processors,) and to the higher 'other miss' rate caused by the higher number of context-switch misses (the misses generated when reloading the working set of a newly scheduled process) related to the higher number of migrating processes. The invalidation miss-rate increase (Fig. 5) is mostly due to the kernel activity, and in particular to the false sharing.

The increase in the number of coherence transactions (Fig. 5) is essentially due to the increased passive sharing. As in the four-processor case, passive sharing becomes more significant with larger caches. Thus the larger caches adopted in current systems enhance the passive sharing overhead.

This experiment also shows the importance of misses and coherence overhead as we scale up the number of processors. In the eight-processor case we need larger caches than in the four-processor case to exploit the machine effectively. For example, to reach at least about the same average processor utilisation that we have for a 128 Kbyte cache in the case of four processors, we need a 2 Mbyte cache in the case of eight processors (Fig. 6).

5.1.3 Effect of block size on misses and coherence overhead:

Both misses and coherence overhead depend on the cache block size. As the block size increases from 32 to 256 bytes the main results are: a higher

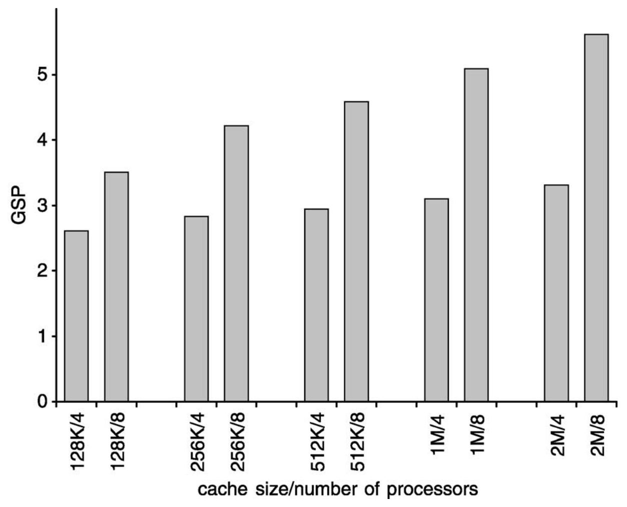

Fig. 4 *Global system power against cache size in the case of 4 and 8 processors, random scheduling policy, 32-byte block size and two-way set associative cache*

GSP increase is higher for 8-processor case as cache size increases due to higher bus utilisation in 8-processor case and longer bus latency

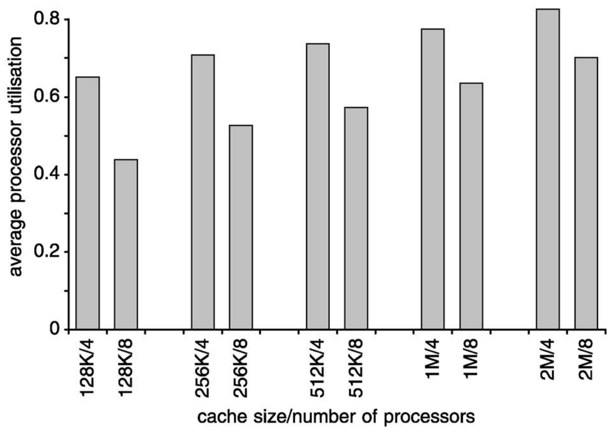

Fig. 6 *Average processor utilisation against cache size in case of 4 and 8 processors, random scheduling policy, 32-byte block size, and two-way set associative cache*

System efficiency decreases as machine doubles number of processors from 4 to 8. Having larger caches is more relevant in 8-processor case

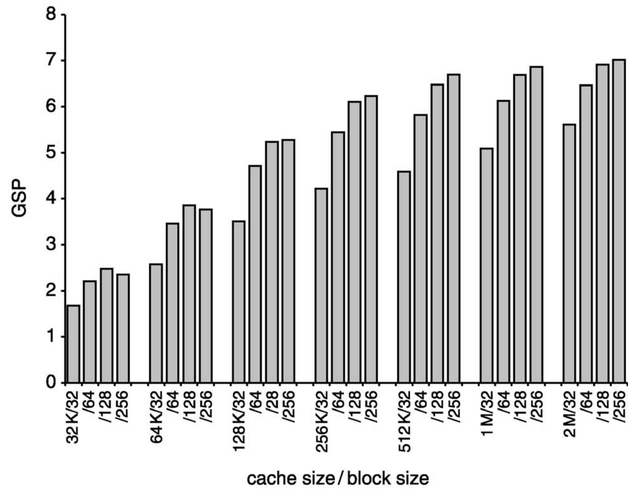

Fig. 7 *Global system power against cache size and block size in case of 8 processors, random scheduling policy, and two-way set associative cache*

GSP increases with block size. Increase is more noticeable when switching from 32-byte block size to 64-byte and 128-byte

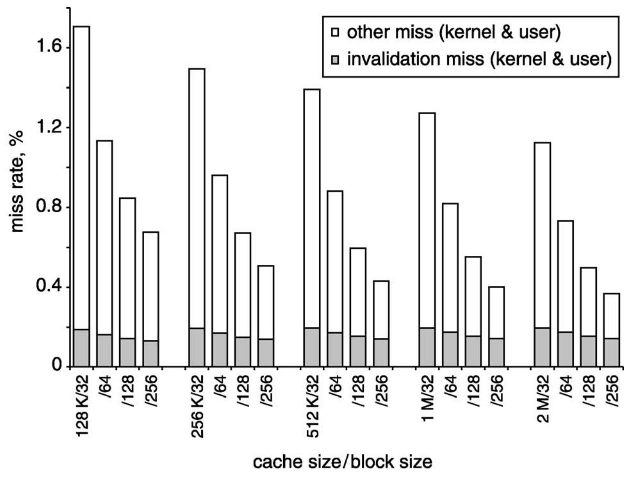

Fig. 9 *Breakdown of miss rate against cache size and block size, in case of 8 processors, random scheduling policy, and two-way set associative cache*

'Other miss' rate decreases as block size increases

GSP (in the case of two-way caches and 2 Mbyte cache sizes, GSP increases from 5.6 to 7, Fig. 7), a lower bus utilisation (in the same case, bus utilisation decreases from 71 to 43%, Fig. 8), and the number invalidation of misses becomes comparable to the number of 'other misses' (Fig. 9). This would allow us to increase architecture scalability up to 18 processors, corresponding to a 14.5 GSP value. The first two results are due to the decrease of miss rate. For a 2 Mbyte cache size, for example, the decrease is from 1.1 to 0.39% (Fig. 9). This is a consequence of the reduction of the 'other miss' component (Fig. 9), and of the invalidation miss component (Fig. 10). The decrease of invalidation misses (Fig. 10) is due to a decrease of false and true sharing misses in the kernel area. Figure 7 shows how the increase of GSP is less effective for block sizes above 128 bytes. Actually both higher transaction cost and coherence overhead have to be taken into account. The third result is caused by the different behaviour of the 'other misses' and invalidation misses, as the block size is increased. The 'other misses' are influenced by the spatial locality of the single process, while invalidation misses depend on the access pattern of processes on different processors.

The main result of this experiment is that by increasing the block size we can improve the performance of the system both in terms of GSP and scalability (although the use of very large block size is not recommended, as we explain subsequently). In the best configuration for the block size, the 'other misses' are about the same number of the invalidation misses. Considering that the 'other misses' cannot be completely eliminated, the only way to further reduce the misses is to act on the invalidation misses.

A drawback of increasing the block size too much is that system performance becomes too tied to the program locality. Considering that program locality may vary, it is not convenient to use much larger block sizes. Therefore an optimal choice for the block size would be 128 bytes for our system. Another drawback is that the number of false sharing misses could increase with block size. False sharing misses are generated by the superposition of two main effects: on the one hand, false sharing misses decrease when

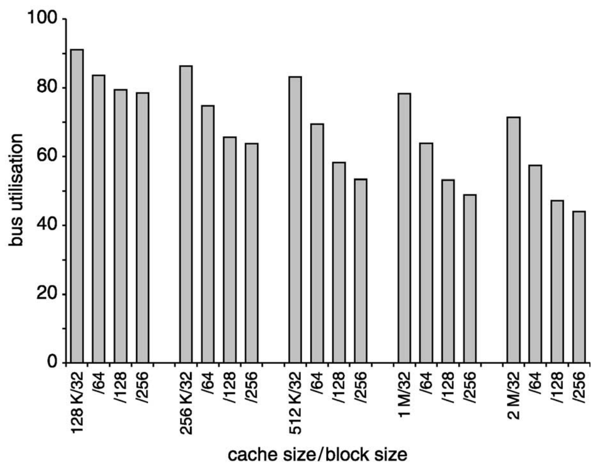

Fig. 8 *Bus utilisation against cache size and block size in case of 8 processors, random scheduling policy, and two-way set associative cache*

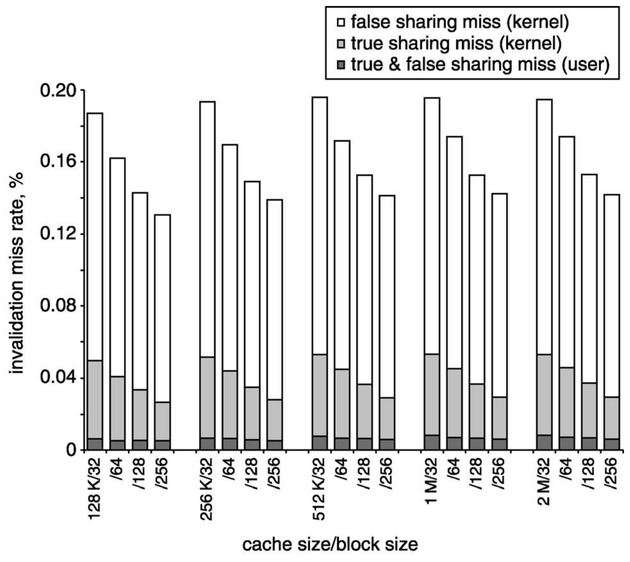

Fig. 10 *Breakdown of invalidation miss rate against cache sizes and block sizes in case of 8 processors, random scheduling policy, and two-way set associative cache*

Every component decreases as cache block size is increased switching from smaller to medium-size blocks. This is due to the lower miss probability, as a general effect of caches. On the other hand, false sharing itself always increases with block size. The actual number of false sharing misses is mainly the combined effect of these two factors. Also, the locality of accesses performed by each process and the probability of migration may have to be accounted for. All these factors make the behaviour of false sharing misses, in a system with migration, not completely predictable.

Some researchers had found an increase of false sharing misses with block size [16, 35, 70] for other applications. However, this is only true starting from a certain cache block size [70] that depends on the application. In our experiments we found that this critical block size is 1024 bytes. This means that a decrease of false sharing misses could be generally observed, as in the case of the block size range we used. A similar result can be found in other recent papers [68, 71].

5.1.4 *Summary of performance limits of four and eight-processor configurations:*

The performance of the machine we analysed depends heavily on the number of cold, conflict, and capacity misses and coherence overhead (due to invalidation misses and coherence transactions). In the four-processor case the use of larger caches can effectively maximise performance. In the case of more processors larger caches can help but, as we show, it is more cost-effective to use other solutions that intervene directly on the major causes (process migration and kernel false sharing) that limit performance and scalability.

The reduction of miss rate is a good result, since it allows us to obtain better performance when the miss cost is high or when we are executing a program that exhibits a number of misses that is intrinsically high. The miss cost may rise as well because of an increase of processor speed, while the other system parameters do not vary. We can reduce classical misses (the 'other misses') by using hardware techniques [30–32] (in particular modifying cache associativity, cache size, and cache block size,) or by using program restructuring techniques [72–74]. Other studies [43] also suggested that instruction cache misses could be reduced by laying out basic blocks that conflict in the cache. The adoption of a scheduling algorithm based on cache-affinity [22, 23] can reduce the 'other miss' component due to process migration (context switch misses).

Coherence overhead weighs more and more as we add processors. The component of coherence overhead due to false sharing can be decreased either by using special coherence protocols, which adopt selective block invalidation [38], by properly allocating the involved shared data structures [35], or by means of data restructuring through profiling information [16, 35]. Kernel data restructuring could be easily accomplished since the kernel is a completely known part of the system at design time. As we wish to push the performance to higher levels, we need to act more effectively on it: we suggest intervening by using an appropriate coherence protocol.

Why is MESI protocol not effective to act on these causes? The main reason is its invalidation mechanism, which produces subsequent invalidation misses that are more dangerous to the performance as we increase the number of processors. For example, by doubling the cache size (from 1 to 2 Mbytes, 128 byte block size, eight processors, Fig. 9), we save only 10% of misses. In the same condition, invalidation misses remain about 35% of the total misses. In this situation it is reasonable to change the

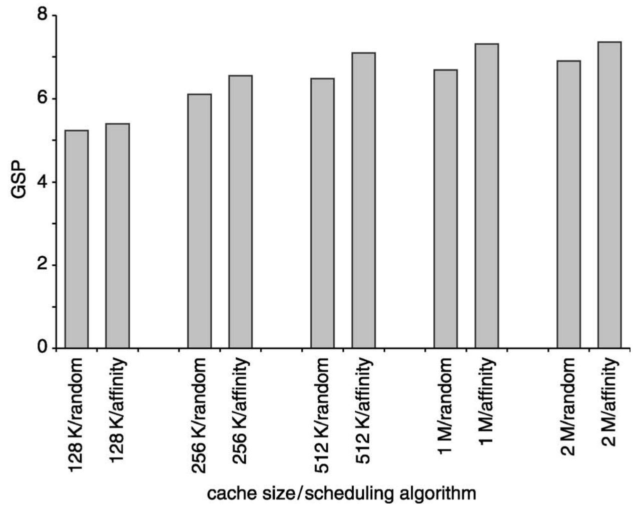

Fig. 11 *Global system power against cache size and scheduling policy (random, affinity), in case of 8 processors, 128-byte block size, and two-way set associative cache*

coherence protocol to reduce the coherence related overhead and the effects of process migration.

5.2 *Reducing process migration and coherence overhead effects*

As process migration is also influenced by the scheduling policy, we evaluate a situation where a scheduling that reduces process migration is used. One of the most-used solutions is cache-affinity scheduling policy. Then we evaluate the benefits of introducing special coherence protocols that act on the process migration.

5.2.1 *Cache-affinity:*

The aim of the following experiments is to analyse the system when the kernel adopts a scheduling policy based on cache-affinity [22, 23]. From previous analysis we deduce that we can continue our study using a baseline configuration with eight processors, 128 byte block size and a two-way set associative cache. The main results in comparison with the case of random scheduling policy are: GSP increases (Fig. 11) and bus utilisation becomes lower allowing for a higher scalability, the percentage of invalidation misses on total misses becomes higher (Fig. 12), and the coherence transactions

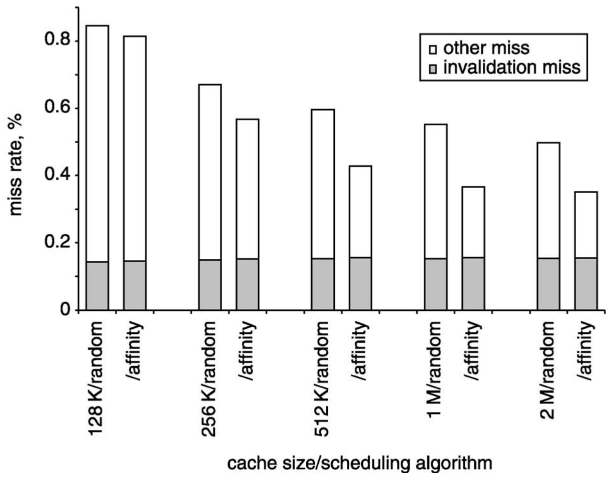

Fig. 12 *Breakdown of miss rate against cache size and scheduling policy (random, affinity), in case of 8 processors, 128-byte block size, and two-way set associative cache*

Miss reduction due to cache-affinity technique is evident

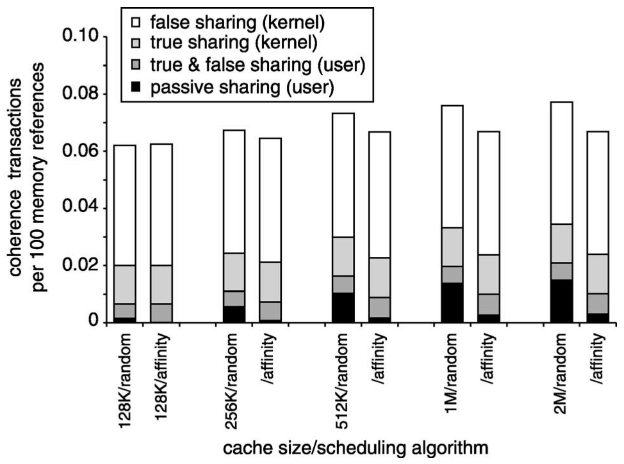

Fig. 13 *Number of coherence transactions (invalidate transactions) against cache size and scheduling policy (random, affinity), in case of 8 processors, 128-byte block size, and two-way set associative cache*

Cache-affinity scheduling reduces mainly passive sharing component, while other components remain constant

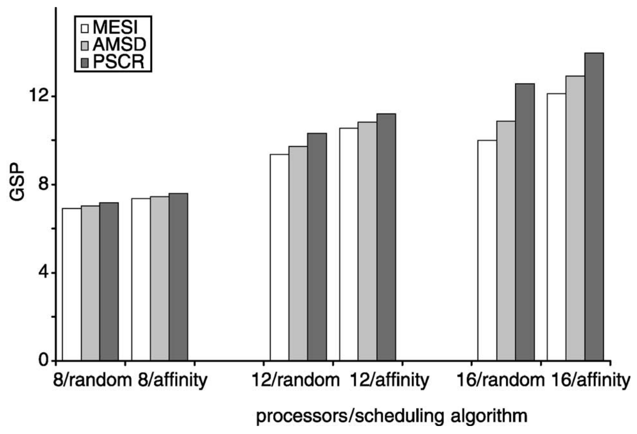

Fig. 14 *Global system power against number of processors and scheduling algorithm (random, affinity)*

Cache is 2-way set associative with 128 byte block size, and 2 Mbyte size

are lower (Fig. 13). This situation would allow us to extend the number of system processors up to 14, with a related increase of GSP equal to 11.3 (again using the definition given in Section 4). The first two results come from the 'other misses' reduction. The third is due to the reduction of passive sharing transactions. As for invalidation misses (not reported in Figure), there is no substantial difference compared with the base scheduling policy case. This means that cache-affinity acts not only on context-switch miss, as expected in the literature [23], but also on the passive sharing transactions, which effect has not been noticed before. Thus affinity is useful to reduce the effects of process migration. It is also clear that to further improve the performance (GSP and scalability) we need to concentrate on the reduction of invalidation misses (which are a 50% of the total misses for a 2 Mbyte cache), since the other misses are now almost exclusively due to intrinsic misses. Another reason to focus on coherence related misses is that they could not be reduced with indefinitely larger caches [43]. This can be observed from the strong flattening of the slope between the cases of 1 Mbyte and 2 Mbyte caches (cache-affinity case) in Fig. 12.

Since we wish to reduce the invalidation misses we can use an adequate coherence protocol. It is also clear that an adequate coherence protocol is important since cache-affinity cannot solve all the problems generated by process migration. Moreover, the load condition can vary significantly and there are cases in which the number of ready-to-run process, compared with the number of available processors, is critical and process migration cannot be avoided [14, 75]. If we rely only on cache-affinity, the machine performance would be too tied to the number of ready process.

5.2.2 Coherence protocol: We wish to show how to reduce coherence overhead (both as invalidation misses and coherence transactions) by using an appropriate coherence protocol. We evaluated 8-, 12-, and 16-processor configurations with a two-way set associative cache, and 128 byte block size. We carried out tests with both random and cache-affinity scheduling algorithms.

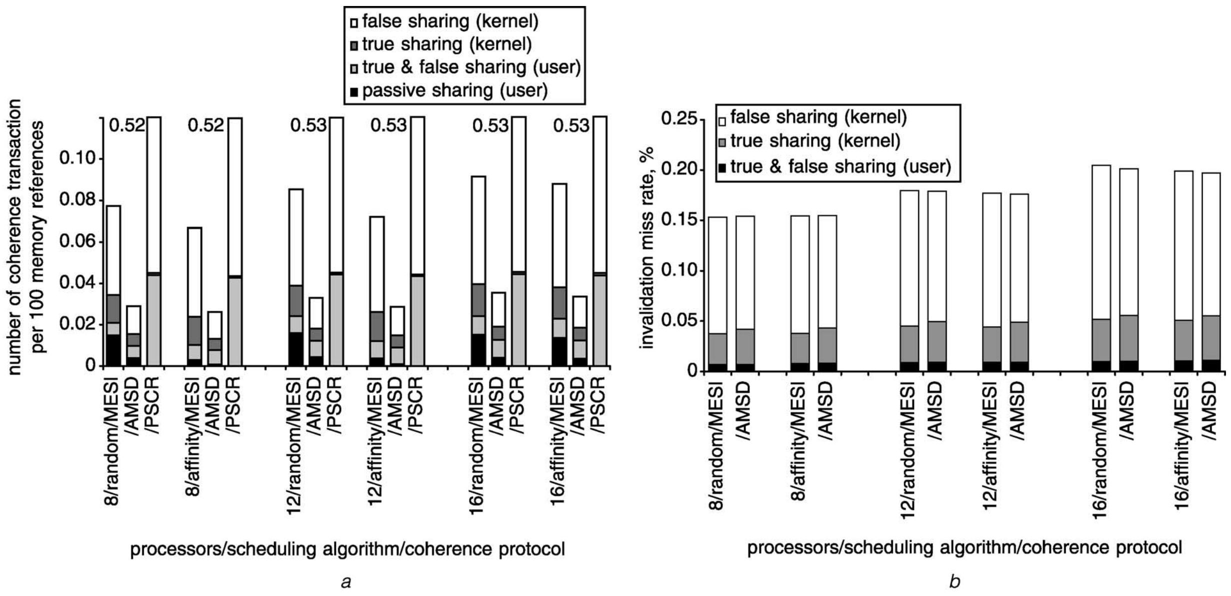

Fig. 15 *a Coherence transactions against number of processor and scheduling algorithm (random, affinity). Cache has 128-byte block size, 2 Mbyte size and is 2-way set associative. b Invalidation miss rate against number of processor and scheduling algorithm (random, affinity). Cache has 128 byte block size, 2 Mbyte size and is 2-way set associative*

Among possible choices for the coherence protocols, we considered PSCR [14] and AMSD [34, 37], as they could reduce the effects of process migration on coherence overhead. The first-mentioned is based on write-update technique, the last on a write-invalidate technique. We described these protocols in Section 2. Here, the main results are: GSP and scalability are improved with PSCR and AMSD in comparison to MESI protocol, and PSCR outperforms both AMSD and MESI in all configurations. As the number of processors increases (Fig. 14), the performance difference among protocols becomes more evident. The choice of MESI protocol, in particular, appears as the least performing.

AMSD has beneficial effects on the reduction of passive sharing, although it does not eliminate it completely. The benefits on passive sharing are mainly due to a decrease of coherence transactions (Fig. 15). The reduction of the number of coherence transaction is due to the behaviour of AMSD on shared copies. When AMSD detects a block that has to be treated exclusively for a long interval, it invalidates the copy locally during the handling of a remote miss, avoiding a necessary consequent bus transaction in this way.

PSCR is based on the update of effectively shared copies, which avoids invalidation misses. Using the write-update technique the number of coherence transactions is higher compared with other protocols (Fig. 15), but they cost less. In fact, the cost of the coherence overhead is limited by the lower cost of the coherence maintaining write transactions (Table 4). On the other hand, the reduction of total number of misses (Fig. 16) produces a more consistent bus utilisation decrease (Fig. 17) than the other protocols do. Another advantage of the use of write transaction, with benefits for these results, is due to the fact that the transaction can be performed asynchronously, without a direct processor delay. Finally, the write transaction cost is independent from the block size. More generally, in nontechnical workloads it has been noticed that there is a scarce reuse of data and there are large working sets [42]. This will give further advantage to the solutions based on write-update techniques, like PSCR.

Now analyse the scalability offered by the various protocols. It has previously been observed that the system is in saturation when the GSP does not increase by a minimal quantity as the number of processors is increased. In our experiments we calculated that this minimal quantity

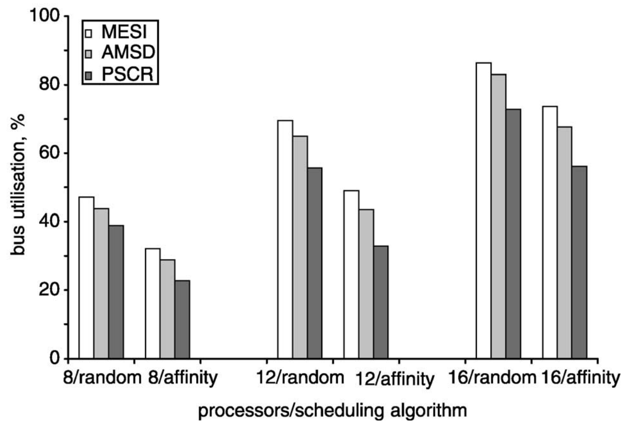

Fig. 17 *Bus utilisation against number of processor and scheduling algorithm (random, affinity)*

Cache has 128 byte block size, 2 Mbyte size and is 2-way set associative

is equal to a GSP of 0.5 for each added processor. As a rule of thumb, this corresponds to a GSP increase of two when switching among different configurations in Fig. 14. Thus, as shown in Fig. 14, we can state that MESI (in the case of random scheduling policy) is already near the saturation threshold for a 12-processor configuration. AMSD performs slightly better since the saturation is reached for a number of processors between 12 and 16, for both scheduling policies. In the configurations shown, PSCR is never in saturation. We also observe that, when using such protocols, the average processor utilisation varies more consistently for configurations having a higher number of processors (Fig. 18). Therefore adding more processors is more effective once a more performing protocol is chosen. This affects performance/price ratio advantageously. For instance, GSP performance for PSCR in a 16-processor configuration is twice that for MESI in an eight-processor configuration (Fig. 14).

Summarising the results of experiments carried out so far: in configurations with a lower number of processors, the choice of a different protocol is less critical. When the performance is pushed to the limits (and consequently the system works near saturation) the designer should take advantage of more optimisation techniques, like smart coherence protocols. The combination of all analysed techniques (adequate block size, cache-affinity, and PSCR) allows system scalability to be pushed up to 20

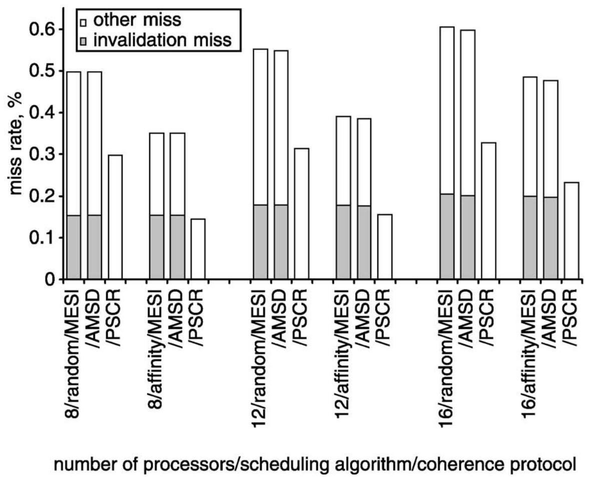

Fig. 16 *Miss rate against number of processor and scheduling algorithm (random, affinity)*

Cache has 128 byte block size, 2 Mbyte size, and is 2-way set associative

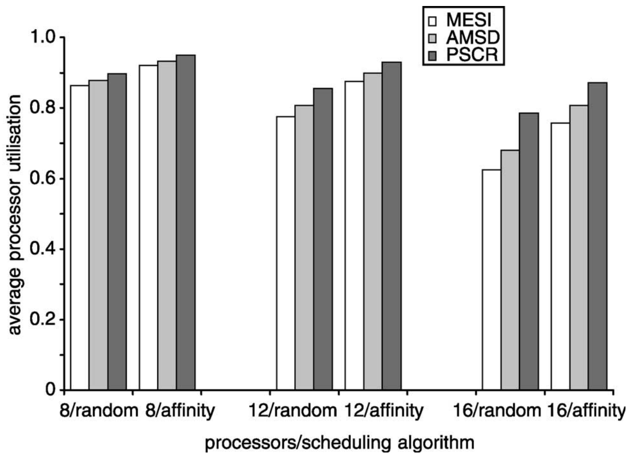

Fig. 18 *Average processor utilisation against number of processors and scheduling algorithm (random, affinity)*

Cache is 2-way set associative with 128 byte block size, and 2 Mbyte size

Table 5: Scalability that can be reached on our shared-bus multiprocessor by combining several solutions

	Section	5.1.1	5.1.2	5.1.3	5.2.1	5.2.2
System parameter	Number of processors	4	8	8	8	16
	Cache capacity (bytes)	2M	2M	2M	2M	2M
	Cache block size (bytes)	32	32	256	128	128
	Cache associativity	2	2	2	2	2
	Scheduling policy	random	random	random	cache affinity	cache affinity
	Coherence protocol	MESI	MESI	MESI	MESI	PSCR
Performance	GSP	3.3	5.6	7	7.4	14
	Bus utilisation	38%	71%	43%	32%	55%
Scalability	Max number of processors	9	9	18	14	20
	Corresponding (Estimated) GSP	~6	~6	~14	~11	~16

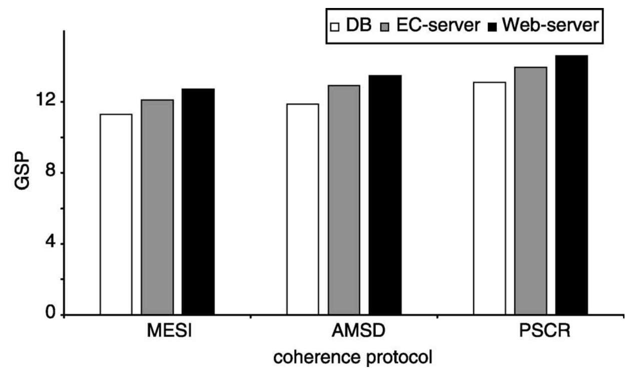

Fig. 19 *Global system power of systems when workload type is changed for possible e-commerce server cases*

Graph refers to case of 16 processors, cache-affinity scheduling, 2 Mbyte cache size, 2-way set associative cache, 128-byte block size

processors with a corresponding GSP of about 16. Further improvement could be obtained if the selective invalidation mechanism were applied to those kernel data structures [43] that are used in an exclusive manner by a process. In Table 5 we report a summary of the configuration that we tested and how the solutions considered have been found effective in increasing the scalability of a machine (Fig. 19).

5.3 Performance analysis for distributed workloads

We analyse another possible solution to implement an e-commerce system. Two distinct single multiprocessors are used to execute separately the Web-server workload (tier two) and the DBMS workload (tier three). The purpose is to evaluate the effectiveness of the bus-based SMP architecture discussed previously also in such distributed environment. We also compare directly the results obtained for the single-multiprocessor configuration with this new case study.

We generated two different workloads, still based on the TPC-W benchmark, namely: DB and Web-server, whose detailed statistics are given in Table 3. The two servers are connected via a high-speed LAN. By putting only the database activity on a single machine, the DBMS server has the opportunity to spawn more processes, thus increasing the quantity of processed data in the time unit and consequently the amount of the accessed shared data (mostly database metadata) by the DBMS.

On the contrary, the Web-server can better utilise its kernel shared data, without interference from DB-server kernel activity. The Web-server activity doesn't involve user shared data; this is characteristic of the Apache

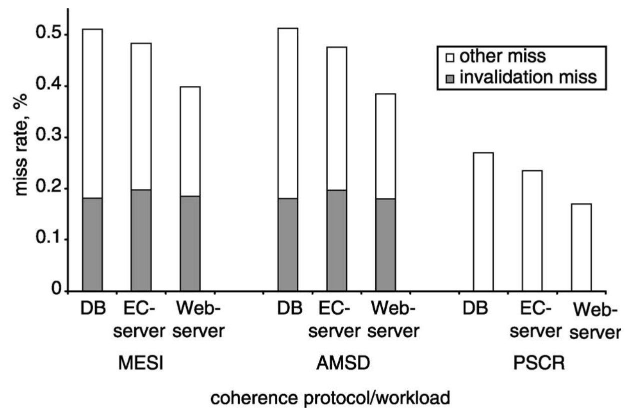

Fig. 20 *Miss rate of systems when workload type is changed for possible e-commerce server cases*

Graph refers to case of 16 processors, cache-affinity scheduling, 2 Mbyte cache size, 2-way set associative cache, 128-byte block size. DB workload exhibits higher miss rate, Web-server the lower

implementation of the web server, where each process manages a preconfigured number of user request without communicating with other user processes.

Further details on these two workloads are discussed in Sections 3 and 4.

The results related to the configuration with cache affinity scheduling and a standard cache configuration (2 Mbyte

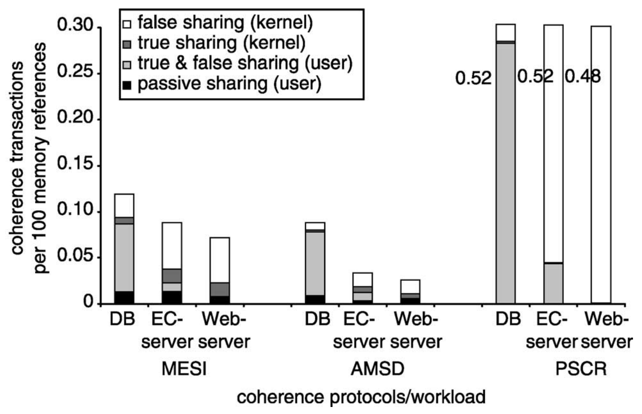

Fig. 21 *Number of coherence transactions per 100 memory references of systems when workload type is changed for possible e-commerce server cases*

Graph refers to case of 16 processors, cache-affinity scheduling, 2 Mbyte cache size, 2-way set associative cache, 128-byte block size. In Web-server workload there are no user shared data, but there are user invalidations due to passive sharing effects

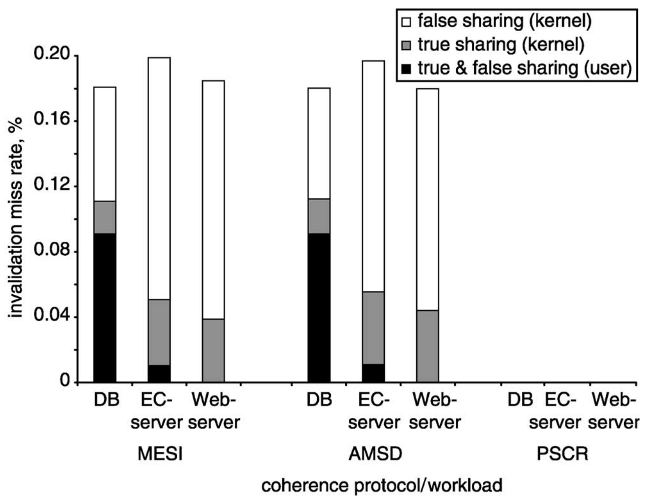

Fig. 22 *Breakdown of invalidation miss rate of systems when workload type is changed for possible e-commerce server cases.*

Graph refers to case of 16 processors, cache-affinity scheduling, 2 Mbyte cache size, 2-way set associative cache, 128-byte block size. User invalidation miss are significant in DB workload, due to database activity while absent in Web-server workload

cache size, two-way set associative cache, 128 byte block size) are as follows

- In the case of DB workload, we observed a GSP decrease compared with EC-server case (Fig. 19). This is essentially due to higher miss rate generated by classical misses (‘other miss’ in Fig. 20) and the increased coherence overhead in the user part (Figs. 21 and 22).
- In the case of Web-server, we have a higher GSP due to a lower miss rate (Fig. 20). This is due to the higher locality of the HTTP processes, as remarked also in [24], and the lack of user shared data. Nevertheless, we have some passive sharing overhead (Figs. 21 and 22). Besides, we have an increase of kernel activity, which generate more invalidation misses and coherence transactions with respect to the DB workload.
- Globally, the considerations that we have drawn for EC-server workload, in spite of varying cache configuration, scheduling policy and coherence protocol, are still valid. In Fig. 19, the GSP obtained with different protocols is compared for the three different workloads considered. The relative performance of the three protocols examined remains the same as in the single-multiprocessor case study.

In conclusion, such analysis shows that when distributing the workload the most critical part to achieve an efficient implementation of an SMP e-commerce system remains the design of the DBMS server. In this case an efficient management of conventional misses and user coherence overhead is more critical to achieve high (memory) performance. In particular, a designer should focus on optimisations of the software component (DBMS) and the hardware component (coherence protocol) to manage the user part of the workload. On the other hand, an efficient web server should include a careful management of kernel resources. From a designer point of view, as in the EC-server case study, we recommend adequate kernel data layout to improve memory performance.

6 Related work

In this Section we consider the most significant evaluations of multiprocessors related to our work and their main results. We present a summary of results for various workloads and platforms. As platforms we considered a combination of multiprocessor architecture and operating system.

Several current categories of commercial applications, like Web-server and database applications, motivated a more realistic evaluation framework for shared memory multiprocessing research [9, 50, 57]. Other studies started considering benchmarks like TPC-series (including OLTP, DSS, Web-server benchmarks), representative of commercial workloads, to evaluate the performance of multiprocessor servers [24–27, 76, 77]. In the following discussion we first consider works that evaluated workloads similar to ours, and then works related to the general framework of designing multiprocessor and operating systems to optimise the execution of complex workload.

Cain *et al.* [76] implemented TPC-W as a collection of Java servlets and presented an architectural study detailing the memory system and branch predictor behaviour of that workload. They used a six-processor IBM RS/6000 S80 SMP machine, running AIX 4.33 operating system. They also evaluated the effectiveness of a coarse-grained multithreaded processor, simulated by means of SimOS, at increasing system throughput. However, their evaluation uses no more than six processors. As shown in our work as well, they found that the false sharing in the user part is almost absent.

Karlsson *et al.* [77] present a detailed characterisation of the memory system behaviour of two benchmarks based on Java (EC-perf and SPECjbb) using both commercial server hardware and full system simulation. Their results show that memory footprint and primary working set of these workloads are small compared with other commercial workload, and that a large fraction of the working set are shared between processor. They suggest also the adoption, in one case, of a shared data cache, typical of single-chip multiprocessors. However, they consider in the evaluation only middle-tier activity (they exclude all the DB activity), so their results are not directly comparable with ours, and are the consequence of an implementation heavily based on Java Virtual Machine.

The paper by Alameldeen *et al.* [50] also evaluates several workloads (OLTP, WEB, Apache, FLASH-like parallel workload) and highlights the influence of several coherency protocols on the global performance. They show that coherency may have a very different impact on performance, depending on the workload.

Ranganathan *et al.* consider both an OLTP workload (modelled after TPC-B [78]) and a DSS workload (query 6 of TPC-D [60, 79]). 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 [59]. The latter processor, used in our evaluation, makes us reasonably safe that this processor architecture is sound for investigation in the memory subsystem.

Trancoso *et al.* consider TPC-D-based DSS queries running on Postgres95 on a simulated four-processor CC-NUMA multiprocessor [27]. They study the memory access patterns of this workload for a memory-resident database. They find that the memory use of queries differs largely depending on the different types of data structures used in database indexing. Coherence-related misses are mainly due to indices and metadata structures in index queries. The simulation encompassed several cache block (4 to 128 bytes) and capacity (128 Kbytes to 8 Mbytes) sizes. The authors conclude that queries can benefit from large cache blocks and data prefetching.

Barroso *et al.* evaluated an Alpha 21164-based SMP memory system by using hardware counter measurements and SimOS simulation [24]. The machine was running Digital-UNIX operating system and Oracle-7 DMBS. The authors consider a TPC-B database (OLTP benchmark), TPC-D (DSS queries) and Altavista search engine. They consider two basic configurations: one with four and the other with eight processors. Results show that the memory hierarchy accounts for 75% of stall time. The number of processes per processor, which is usually kept high to hide I/O latencies, significantly influences cache behaviour as well. For TPC-B workload, they report a high coherency-related miss rate. When switching among 2-, 4-, 6-, and 8-processor configurations they found that coherency miss stalls increase linearly with the number of processors. Beyond an 8 Mbytes outer-level cache they observed that true sharing misses limit performance.

Keeton *et al.* characterise the performance of Quad Pentium Pro SMP server running commercial applications [26]. They found that the most critical issue in the case of database workload (namely TPC-C) is the processor traffic to memory. They used an Informix DBMS on Windows NT-SP3 operating system (allowing cache-affinity scheduling). The methodology of evaluation relies on hardware counters. They found almost the same characteristics (high context switch rates, substantial portion of execution time in operating system) found by others as for commercial workloads [24, 42]. They conclude that larger caches are necessary especially in the case of multiprocessor systems, even if they increase the coherency traffic. As a matter of fact, larger caches produce lower bus utilisation and lower memory latencies. The findings of that work are in agreement with ours, in the case of four-processor configuration. Anyway, we also analysed other aspects that become critical in different architectural configurations (e.g. with more processors). Other differences are due to our specific database workload: while in the authors' work there are many read and write operations, in our case, that is for a database connected to a web server, we mostly have read patterns. Therefore as discussed in Section 5.3, we find different cache performance in the case of DB-only workload.

Cao *et al.* examine a TPC-D workload in execution on a Pentium-Pro four-processor machine with Windows NT and MS SQL Server [25]. Their methodology is based on hardware counters. Major sources of processor stalls are instruction fetch bottleneck and data miss in outer-level caches. They found lower miss rates for data caches in comparison with other studies on TPC-C [24, 26]. This is due to the smaller working set of TPC-D compared with TPC-C.

Most of the conclusions of these commercial-workload evaluations [24, 26, 27, 76] have analogies with our evaluation, especially in the case of four-processor configuration. In particular, large caches, more associativity, and larger blocks help in the case of large working set. The major drawback of large caches is the increased coherence overhead. In our case we consider also passive sharing, configurations with more processors and with different solutions for the cache parameters, coherence protocol, and scheduling policies (in particular cache affinity). Another important contribution of our work concerns the evaluation of a bus-based shared-memory multiprocessor in the context of three-tier software architecture. We consider database and Web-server workloads, as well as the combination of these two.

In the evaluation of multiprocessor architectures, a popular benchmark suite used in the past decade is SPLASH-2 [80] from Stanford University. The suite comprises technical (scientific, engineering, and graphics) applications and is targeted for the evaluation of multiprocessor systems like DSM and CC-NUMA.

Some other studies compare the behaviour of technical and commercial workloads [26, 42], finding several differences. Technical workloads are highly code-optimised, 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 benchmark ones, as noticed in [26, 42, 81]. Moreover, technical workloads are usually single-user, while commercial workloads are usually multiuser and have significant amount of kernel activity [42]. Keeton *et al.* also notice that commercial applications exhibit high context-switch rate compared with technical ones [26].

Griffazzi–Maynard *et al.* examine the characteristic differences between technical and commercial workloads and illustrate how those differences affect cache performance [42]. The reference machine is a uniprocessor system based on IBM RS-6000, running AIX operating system. They use trace-driven simulation to explore cache design alternatives that perform well in commercial workloads. They found that large instruction caches help because of the large code working sets of nontechnical applications; in the case of database application, the user and kernel portion can exhibit similar behaviour and in particular they can experience misses similarly high; process switching should be considered in the design of outer-level caches. Some of these results are in common with ours, since we both use a commercial workload, in particular, the behaviour of the applications as the block and cache size increases, as discussed subsequently. In our case, the focus is on TPC-W in execution on a multiprocessor-based system. As for other commercial workloads, the typical scenario is a single complex application that is shared by several users (or clients). Anyway, most of their results are not directly applicable. For instance, classical misses change because the number of processes visiting a certain processor decreases, and data sharing produce significant overhead not present in the uniprocessor case.

Woo *et al.* evaluate a 32-processor CC-NUMA for characterising SPLASH-2 benchmark [80]. Kernel activity for this workload is not significant and the evaluation focuses on the user part of execution time. The authors find that performance is highly influenced by coherence overhead: misses are mainly due to coherence misses rather than classical misses. In particular, they show that false sharing overhead and interconnection network traffic increases, while total misses decrease with the block size (in the range of 32 to 256 bytes).

Other studies consider the influence of operating system on the overall evaluation of a multiprocessor system. Chapin *et al.* characterise the performance of the memory system of a CC-NUMA machine: the Stanford DASH with 32 processors organised as eight clusters of four processors each one, running a precommercial release of SGI IRIX 5.2 (based on UNIX SVR4) [82]. The authors analyse the effects of possible operating system and architectural changes. Their methodology relies on a nonintrusive cache miss monitor. They consider a fairly OS-intensive technical workload, namely Combo32, which features characteristics of a software development/engineering environment. We found analogies with this work regarding the importance of OS effects, the high influence of coherency misses, and the limited usefulness of large caches in a multiprocessor system. However, most of their results are not directly applicable to our case.

Chandra *et al.* examine the effects of OS scheduling and page migration policies on the performance of the Stanford DASH distributed memory cache-coherent multiprocessor server [41]. The machine has 16 processors organised as four clusters of four processors each one, which uses a modified version of SGI IRIX (based in UNIX SVR3). They consider different workloads consisting of multiple parallel applications that are combined in several ways. They found that cache-affinity with page migration is effective in the case of their architecture and for a sequential multi-programmed workload. They observe that in other studies on bus-based multiprocessors these techniques are not so effective. Our results on cache-affinity scheduling agree with this work. In particular we found that affinity scheduling is not so effective when the number of processes is near to the number of processors.

Torrellas *et al.* characterise the cache performance of a commercial operating system (System-V UNIX) running on a four-processor multiprocessor (SGI PowerStation 4D/340). The authors consider the following three workloads: a parallel compile of 56 files; a parallel numeric program running concurrently with the parallel compile and a five-screen edit sessions; an Oracle database. The methodology relies on a hardware monitor that records the cache misses of this fixed-configuration commercial machine. Their results show that the operating system can slow down software development and commercial workloads by 17–21%. They identify the causes for this slowdown and propose solutions to reduce their effects. We found analogies between this work's results and ours in the case of four-processor configuration. Anyway, we also analyse other aspects that become critical in different architectural configurations (e.g. with more processors). Other differences are due to our specific workload.

7 Conclusions

This paper has analysed the memory subsystem of a shared-bus multiprocessor employed to run e-commerce applications. In particular, we have shown how the performance and the scalability of the whole system can be improved (even doubled compared with MESI based solutions) by resolving specific problems of the memory subsystem, like passive sharing. Our workload has been set up according to the specification from TPC-W benchmark. We considered software components like an HTTP server (Apache), PostgreSQL database management system, and typical UNIX shell commands. The analysis has been carried out through trace-driven simulation and by considering not only user references, but also the most influencing kernel activities.

In the four-processor case, our conclusions agree with the literature: intervening on the cache capacity and associativity can increase GSP and scalability. As we scale up the number of processors the importance of misses and coherence overhead becomes clearer. In particular, coherence overhead and process migration weighs more and more as we add processors. In the eight-processor case, we need larger caches than in the four-processor case to exploit the machine effectively. By increasing the block size we can improve the performance of the system both in terms of GSP and scalability. However, increasing the block size too much makes system performance too tied to the program locality. Also the adoption of a scheduling algorithm based on cache-affinity improves the performance. Cache-affinity is useful to reduce the effects of process migration: it acts on context-switch misses, as expected in the literature, but also on the passive sharing transactions. However, cache-affinity cannot eliminate process migration. In fact, the load condition can vary significantly in e-commerce workloads and there are situations in which the number of processes, compared with the number of available processors, is critical and process migration cannot be avoided. In such a case the machine performance significantly decreases.

To limit the overhead due to process migration we suggest the adoption of appropriate coherence protocols, like AMSD and PSCR, different from usual MESI protocol. As the number of processors increases the performance difference among protocols becomes more evident. In particular the choice of MESI protocol appears the least reliable. AMSD has beneficial effects on passive sharing although it does not eliminate it completely. PSCR eliminates passive sharing and avoids invalidation misses. By using the write-update technique, the number of coherence transactions is higher compared with other protocols. On the other hand, the reduction of total number of misses produces a more consistent bus utilisation decrease than with the other protocols. The combination of all analysed techniques (adequate block size, cache-affinity, and coherence protocols) allows us to push system scalability up to 20 processors with a corresponding GSP of about 16. Further improvement could be obtained if the selective invalidation mechanism of PSCR were used for the kernel data structure associated to a process in an exclusive manner.

The considerations that we have drawn, while varying cache configuration, scheduling policy, and coherence protocol, are still valid when we distribute the whole workload on two multiprocessor systems, one running the second tier and another running the third tier of the e-commerce workload. In particular, for the Web-server (second tier of the e-commerce workload), the higher locality of the HTTP processes speeds up the execution compared with the other solutions, and the main focus for system developers must be on the design of the kernel. The DB workload (third tier of the e-commerce workload), instead, is the less performing part of the system. The focus here must be on the DBMS software and on coherence strategy to deal with user shared data, rather than on the kernel.

8 Acknowledgments

The work described has been carried out under the financial support of the Italian Ministero dell'Università e Ricerca Scientifica e Tecnologica (MURST). We are particularly grateful to Gianpaolo Prina and Luigi Ricciardi for a contribution to the multiprocessor simulator design and the useful discussions about the strategy to limit passive sharing effects. We thank the anonymous referees for their precious comments, which helped improve the quality of our paper.

9 References

1 Andreoli, J.M., Pacull, F., and Pareschi, R.: 'XPECT: a framework for electronic commerce', *IEEE Internet Comput.*, 1997, **1**, (4), pp. 40–48
2 Milutinovic, V.: 'Infrastructure for electronic business on the internet' (Kluwer, Norwell, MA, 2001)
3 Walker, D.W.: 'Free-market computing and the global economic infrastructure', *IEEE Parallel Distrib. Technol.*, 1996, **4**, (3), pp. 60–62

4 Brandau, B., Confrey, T., D'Silva, A., Matheus, C.J., and Weihmayer, R.: 'Reinventing GTE with information technology', *IEEE Comput.*, 1999, **32**, (3), pp. 50–58
5 Edwards, J.: '3-tier client/server at work' (Wiley, New York, NY, 1999)
6 Lewis, T.: 'The legacy maturity model', *IEEE Comput.*, 1998, **31**, (11), pp. 125–128
7 Buyya, R.: 'High performance clustered computing' (Prentice Hall PTR, NJ, USA, 1999)
8 Short, R., Gamache, R., Vert, J., and Massa, M.: 'Windows NT clusters for availability and scalability'. Proc. 42nd IEEE Int. Computer Conf., San Jose, CA, February 1997, pp. 8–13
9 Stenström, P., Hagersten, E., Li, D.J., Martonosi, M., and Venugopal, M.: 'Trends in shared-memory multiprocessing', *IEEE Comput.*, 1997, **30**, (12), pp. 44–50
10 Prete, C.A.: 'RST cache memory design for a tightly coupled multiprocessor system', *IEEE Micro*, 1991, **11**, (2), pp. 16–19, 40–52
11 Tomasevic, M., and Milutinovic, V.: 'The cache coherence problem in shared-memory multiprocessors – hardware solutions' (IEEE Computer Society, Los Alamitos, CA, 1993)
12 Tomasevic, M., and Milutinovic, V.: 'Hardware approaches to cache coherence in shared-memory multiprocessors', *IEEE Micro*, 1994, **14**, (5), pp. 52–59
13 Sweazey, P., and Smith, A.J.: 'A class of compatible cache consistency protocols and their support by the IEEE futurebus'. Proc. 13th Int Symp. on Computer Architecture, Tokyo, Japan, June 1986, pp. 414–423
14 Giorgi, R., and Prete, C.A.: 'PSCR: a coherence protocol for eliminating passive sharing in shared-bus shared-memory multiprocessors', *IEEE Trans. Parallel Distrib. Syst.*, 1999, **10**, (7), pp. 742–763
15 Prete, C.A., Prina, G., Giorgi, R., and Ricciardi, L.: 'Some considerations about passive sharing in shared-memory multiprocessors', *IEEE TCCA Newsletter*, March 1997, pp. 34–40
16 Torrellas, J., Lam, M.S., and Hennessy, J.L.: 'False sharing and spatial locality in multiprocessor caches', *IEEE Trans. Comput.*, 1994, **43**, (6), pp. 651–663
17 TPC BENCHMARK W (Web Commerce) Specification, v. 1.0.1 (2000) Transaction Processing Performance Council
18 Robinson, D., and the Apache Group: APACHE – An HTTP Server, Reference Manual, 1995, http://www.apache.org, accessed January 1995
19 Yu, A., and Chen, J.: 'The POSTGRES95 user manual'. Computer Science Div., Dept. of EECS, University of California at Berkeley, July 1995
20 Giorgi, R., Prete, C.A., Prina, G., and Ricciardi, L.: 'Trace Factory: generating workloads for trace-driven simulation of shared-bus multiprocessors', *IEEE Concurr.*, 1997, **5**, (4), pp. 54–68
21 Prete, C.A., Prina, G., and Ricciardi, L.: 'A trace-driven simulator for performance evaluation of cache-based multiprocessor system', *IEEE Trans. Parallel Distrib. Syst.*, 1995, **6**, (9), pp. 915–929
22 Squillante, M.S., and Lazowska, D.E.: 'Using processor-cache affinity information in shared-memory multiprocessor scheduling', *IEEE Trans. Parallel Distrib. Syst.*, 1993, **4**, (2), pp. 131–143
23 Torrellas, J., Tucker, A., and Gupta, A.: 'Evaluating the performance of cache-affinity scheduling in shared-memory multiprocessors', *J. Parallel Distrib. Comput.*, 1995, **24**, (2), pp. 139–151
24 Barroso, L.A., Gharachorloo, K., and Bugnion, E.: 'Memory system characterization of commercial workloads'. Proc. 25th Int. Symp. on Computer Architecture, Barcelona, Spain, June 1998 pp. 3–14
25 Cao, Q., Torrellas, J., Trancoso, P., Larriba-Pey, J.L., Knighten, B., and Won, Y.: 'Detailed characterization of a quad Pentium Pro server running TPC-D'. Proc. Int. Conf. on Computer Design, Austin, TX, October 1999, pp. 108–115
26 Keeton, K., Patterson, D., He, Y., Raphael, R., and Baker, W.: 'Performance characterization of a quad Pentium Pro SMP using OLTP workloads'. Proc. 25th Int. Symp. on Computer Architecture, Barcelona, Spain, June 1998, pp. 15–26
27 Trancoso, P., Larriba-Pey, J.L., Zhang, Z., and Torrellas, J.: 'The memory performance of DSS commercial workloads in shared-memory multiprocessors'. Proc. 3rd Int. Symp. on High-performance Computer Architecture, San Antonio, TX, February 1997, pp. 250–260
28 Pai, V.S., Ranganathan, P., Abdel-Shafi, H., and Adve, S.: 'The impact of exploiting instruction-level parallelism on shared-memory multiprocessors', *IEEE Trans. Comput.*, 1999, **48**, (2), pp. 218–226
29 Saulsbury, A., Pong, F., and Nowatzyk, A.: 'Missing the memory wall: the case for processor/memory integration'. Proc. 23rd Int. Symp. on Computer Architecture, Philadelphia, PA, May 1996, pp. 90–103
30 Flynn, M.J.: 'Computer architecture, pipelined and parallel processor design' (Jones and Bartlett, Sudbury, MA, 1995)
31 Hennessy, J., and Patterson, D.A.: 'Computer architecture: a quantitative approach' (Morgan Kaufmann, San Francisco, CA, 1996, 2nd edn.)
32 Hwang, K., and Xu, Z.: 'Scalable parallel computing: technology, architecture, programming' (McGraw-Hill, New York, NY, 1998)
33 Agarwal, A., and Gupta, A.: 'Memory reference characteristics of multiprocessor applications under Mach'. Proc. ACM Sigmetrics, Santa Fe, NM, May 1998, pp. 215–225
34 Cox, A.L., and Fowler, R.J.: 'Adaptive cache coherency for detecting migratory shared data'. Proc. 20th Int. Symp. on Computer Architecture, San Diego, CA, May 1993, pp. 98–108
35 Jeremiassen, T.E., and Eggers, S.J.: 'Reducing false sharing on shared-memory multiprocessors through compile time data transformations', *ACM SIGPLAN Notice*, 1995, **30**, (8), pp. 179–188
36 Prete, C.A.: 'A new solution of coherence protocol for tightly coupled multiprocessor systems'. Proc. EUROMICRO 90: Hardware and Software in System Engineering, Microprocessing and Microprogramming, Vienna, Austria, August 1990, **30**, (1-5), pp. 207–214
37 Stenström, P., Brorsson, M., and Sandberg, L.: 'An adaptive cache coherence protocol optimized for migratory sharing'. Proc. 20th Annual Int. Symp. on Computer Architecture, San Diego, CA, May 1993, pp. 109–118
38 Tomasevic, M., and Milutinovic, V.: 'The word-invalidate cache coherence protocol', *Microprocess. Microsyst.*, 1996, **20**, (3), pp. 3–16
39 Eggers, S.J.: 'Simulation analysis of data sharing in shared-memory multiprocessors'. PhD thesis, UCB/CSD 89/501, University of California, Berkeley, April
40 Gupta, A., and Weber, W.-D.: 'Cache invalidation patterns in shared-memory multiprocessors', *IEEE Trans. Comput.*, 1992, **41**, (7), pp. 794–810
41 Chandra, R., Devine, S., Verghese, B., Gupta, A., and Rosenblum, M.: 'Scheduling and page migration for multiprocessor compute servers'. Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1994, pp. 12–24
42 Grizzaffi Maynard, A.M., Donnelly, C.M., and Olszewski, B.R.: 'Contrasting characteristics and cache performance of technical and multiuser commercial workloads'. Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1994, pp. 158–170
43 Torrellas, J., Gupta, A., and Hennessy, J.: 'Characterizing the caching and synchronization performance of a multiprocessor operating system'. Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Boston, MA, September 1992, pp. 162–174
44 Handy, J.: 'The Cache Memory Book', (Academic Press, San Diego, CA, 1998, 2nd edn.)
45 Shanley, T., and Mindshare Inc: 'Pentium Pro and Pentium II system architecture' (Addison Wesley, Reading, MA, 1999, 2nd edn.)
46 'AMD x86-64 Architecture Programmer's Manual Vol. 2: System Programming', Advanced Micro Device Inc., September 2002
47 Tendler, J.M., Dodson, J.S., Fields, J.S., Le, H., and Sinharoy, B.: 'POWER4 system microarchitecture', *IBM J. Res. Dev.*, 2002, **46**, (1), pp. 5–26
48 Martin, M.M.K., Sorin, D.J., Hill, M.D., and Wood, D.A.: 'Bandwidth-adaptive snooping'. Proc. 8th Int. Symp. on High-performance Computer Architecture, Anaheim, CA, February 2002, pp. 224–235
49 Martin, M.M.K., Sorin, D.J., Ailamaki, A., Alameldeen, A.R., Dickson, R.M., Mauer, C.J., Moore, K.E., Plakal, M., Hill, M.D., and Wood, D.A.: 'Timestamp snooping: an approach for extending SMPs'. Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, November 2000, pp. 25–36
50 Alameldeen, A.R., Martin, M.M.K., Mauer, C.J., Moore, K.E., Xu, M., Hill, M.D., Wood, D.A., and Sorin, D.J.: 'Simulating a $2M commercial server on a $2K PC', *IEEE Comput.*, 2003, **36**, (2), pp. 50–57
51 Limprecht, R.: 'Microsoft transaction server'. Proc. 42nd IEEE Int. Computer Conf., San Jose, CA, February 1997, pp. 14–18
52 Edwards, J.: 'The changing face of freeware', *IEEE Comput.*, 1998, **31**, (10), pp. 11–13
53 GNU Free Software Foundation, http://www.gnu.org/software/, accessed February 2003
54 Stunkel, C.B., Janssens, B., and Fuchs, W.K.: 'Address tracing for parallel machines', *IEEE Comput.*, 1991, **24**, (1), pp. 31–45
55 Uhlig, R.A., and Mudge, T.N.: 'Trace-driven memory simulation: a survey', *ACM Comput. Surv.*, 1997, pp. 128–170
56 Linux on SGI/MIPS, http://oss.sgi.com/mips/, accessed February 2003
57 Mauer, C.J., Hill, M.D., and Wood, D.A.: 'Full-system timing-first simulation'. Proc. 2002 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems, Marina del Rey, CA, June 2002, pp. 108–116
58 Goldschmidt, S.R., and Hennessy, J.L.: 'The accuracy of trace-driven simulations of multiprocessors'. Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, Santa Clara, CA, May 1993, pp. 146–157
59 Yeager, K.C.: 'The MIPS R10000 superscalar microprocessor', *IEEE Micro*, 1996, **16**, (4), pp. 42–50
60 Ranganathan, P., Gharachorloo, K., Adve, S.V., and Barroso, L.: 'Performance of database workloads on shared-memory systems with out-of-order processors'. Proc. 8th Int Conf. on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1998, pp. 307–318
61 Kroft, D.: 'Lockup-free instruction fetch/prefetch cache organization'. Proc. 8th Annual Int. Symp. on Computer Architecture, Minneapolis, MN, June 1981, pp. 81–87
62 Adve, S.V., and Gharachorloo, K.: 'Shared memory consistency models: a tutorial', *IEEE Comput.*, 1996, **9**, (12), pp. 66–76
63 Gharachorloo, K., Gupta, A., and Hennessy, J.: 'Performance evaluation of memory consistency models for shared-memory multiprocessors'. Proc. 4th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Santa Clara, CA, April 1991, pp. 245–357
64 Nanda, A.K., Nguyen, A., Michael, M., and Joseph, D.: 'High throughput coherence controller'. Proc. 6th Int. Symp. on High-performance Computer Architecture, Toulouse, France, January 2000, pp. 145–155
65 An overview of the UltraSPARC III Cu Processor v1.1, Sun Microsystems, Inc., Palo Alto, CA, June 2002
66 The POWER4 Processor Introduction and Tuning Guide, Int. Business Machine Corp., Austin, TX, November 2001

67 Archibald, J.K., and Baer, J.L.: 'Cache coherence protocols: evaluation using a multiprocessor simulation model', *ACM Trans. Comput. Syst.*, 1986, **4**, pp. 273–298
68 Foglia, P.: 'An algorithm for the classification of coherence-related overhead in shared-bus shared-memory multiprocessors', *IEEE TCCA News.*, 2001, pp. 53–58
69 Hyde, R.L., and Fleisch, B.D.: 'An analysis of degenerate sharing and false coherence', *J. Parallel Distrib. Comput.*, 1996, **34**, (2), pp. 183–195
70 Dubois, M., Skeppstedt, J., Ricciulli, L., Ramamurthy, K., and Stenström, P.: 'The detection and elimination of useless miss in multiprocessors'. Proc. 20th Int. Symp. on Computer Architecture, San Diego, CA, May 1993, pp. 88–97
71 Lepak, K.M., and Lipasti, M.H.: 'On the value locality of store instructions'. Proc. 27th Annual Int. Symp. on Computer Architecture, Vancouver, Canada, June 2000, pp. 182–191
72 Kalamatianos, J., Khalafi, A., Kaeli, D., and Meleis, W.: 'Analysis of temporal-based program behavior for improved instruction cache performance', *IEEE Trans. Comput.*, 1999, **48**, (2), pp. 168–175
73 Lorenzini, S., Luculli, G., and Prete, C.A.: 'A fast procedure placement algorithm for optimal cache use'. Proc. 9th IEEE Mediterranean Electrotechnical Conference MELECON, Tel Aviv, Israel, May 1998, pp. 1279–1284
74 Torrellas, J., and Daigle, R.: 'Optimizing the instruction cache performance of the operating system', *IEEE Trans. Comput.*, 1998, **47**, (12), pp. 1363–1381
75 Torrellas, J., Tucker, A., and Gupta, A.: 'Benefits of cache-affinity scheduling in shared-memory multiprocessors'. Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, Santa Clara, CA, May 1993, pp. 272–274
76 Cain, T., Rajwar, R., Marden, M., and Lipasti, M.: 'An architectural characterization of Java TPC-W'. Proc. 7th Int. Symp. on High-performance Computer Architecture, Monterrey, Mexico, January 2001, pp. 229–240
77 Karlsson, M., Moore, K.E., Hagersten, E., and Wood, D.A.: 'Memory system behavior of Java-based middleware'. Proc. 9th Int. Symp. on High-performance Computer Architecture, Anaheim, CA, February 2003, pp. 217–228
78 'TPC Benchmark B (Online Transaction Processing) Standard Specification'. Transaction Processing Performance Council, 1994
79 'TPC Benchmark D (Decision Support) Standard Specification'. Transaction Processing Performance Council, Santa Margherita Ligure, Italy, 1995
80 Woo, S.C., Ohara, M., Torrie, E., Shingh, J.P., and Gupta, A.: 'The SPLASH-2 programs: characterization and methodological considerations'. Proc. 22nd Int. Symp. on Computer Architecture, May 1994, pp. 24–36
81 Cvetanovic, Z., and Bhandarkar, D.: 'Characterization of Alpha AXP performance using TP and SPEC workloads'. Proc. 21st Int. Symp. on Computer Architecture, Chicago, IL, April 1994, pp. 60–70
82 Chapin, J., Herrod, S., Rosenblum, M., and Gupta, A.: 'Memory system performance of UNIX on CC-NUMA multiprocessors'. Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, Ottawa, Canada, May 1995, pp. 1–13