Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Journal of Systems Architecture 51 (2005) 251–264

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Evaluating IA-32 web servers through simics: a practical experience

F.J. Villa *, M.E. Acacio, J.M. García

Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, 30071 Murcia, Spain

Received 11 November 2003; received in revised form 26 July 2004; accepted 14 September 2004
Available online 7 January 2005

Abstract

Nowadays, the use of multiprocessor systems is not just limited to typical scientific applications, but these systems are increasingly being used for executing commercial applications, such as databases and web servers. Therefore, it becomes essential to study the behavior of multiprocessor architectures under commercial workloads. To accomplish this, we need simulators able to model not only the CPU, memory and interconnection network but also other aspects that are critical in the execution of commercial workloads, such as I/O subsystem and operating system. In this paper, we present our first experiences using *Simics*, a simulator which allows full-system simulation of multiprocessor architectures covering all the topics previously mentioned. Using *Simics* we carry out a detailed performance study of a static web content server, showing how changes in some architectural parameters, such as number of processors and cache size, affect final performance. The results we have obtained corroborate the intuition of increasing performance of a dual-processor web server opposite to a single-processor one, and at the same time, allow us to check out *Simics* limitations. Finally, we compare these results with those that are obtained on real machines.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Simics; Commercial applications; Full system simulators; Multiprocessor systems

1. Introduction

Multiprocessor systems have been traditionally used in scientific areas: weather study and modelling, universe modelling, molecular algorithms, etc. This kind of problems can be easily reproduced and studied using user-level simulators like RSIM [8] and scientific benchmarks as those provided by the SPLASH-2 suite [17].

However, multiprocessor systems are also currently being used for executing other kind of applications, usually known as commercial, among which we can find web servers, for example. It is well-known that the importance of the Internet

* Corresponding author. Tel.: +34 968 364665; fax: +34 968 364151.
E-mail addresses: fj.villa@ditec.um.es (F.J. Villa), meacacio@ditec.um.es (M.E. Acacio), jmgarcia@ditec.um.es (J.M. García).

has grown exponentially in the last years, to the point of becoming a part of our every day lives. Nowadays, all the medium-sized and large-scale companies, even the small ones, have a web portal that is suitable as a "shop window" for customers around the world. This situation can be extended to all types of organizations: governments, academic institutions, etc. Large organizations, which expect to receive a huge number of user connections everyday, need to have a powerful server, which usually is implemented as a multiprocessor.

As a consequence of the increasing use of multiprocessors in this field, simulating multiprocessor architectures running web servers accurately becomes important. Opposite to scientific applications, there are some characteristics of commercial workloads that make challenging their simulation. In particular, the activity of the operating system is very important in these applications, as well as the interaction with memory hierarchy, storage system and communication network. It is thus required that the simulators used in these studies model all these aspects if accurate simulation results want to be obtained. *Simics* [10] is a full-system simulator which allows to simulate operating system, memory hierarchy, storage, buses, microprocessor, communication network, and so on. *Simics* is increasingly being used as a platform for simulating multiprocessor architectures running commercial applications, and it is currently used in more than 300 universities all over the world.

In this paper, we study the possibilities *Simics* offers to characterize web servers. First of all, we describe the main characteristics of the simulator. Secondly, using *Simics* we evaluate the performance of a dual-processor web server and we compare it with the performance obtained when using a single-processor one. The results that are obtained corroborate the intuition that a dual-processor web server obtain higher performance. Finally, we repeat the experiments using real machines which allows us to highlight some of the current limitations of *Simics*.

The rest of the paper is organized as follows. Next Section presents some related work in the evaluation of commercial applications with *Simics*. Section 3 deals with the simulator's main characteristics. Section 4 describes the commercial workload we have used: a static web content server, being *Apache* the web server and *httperf* the utility which places the workload at the server. Section 5 contains the evaluation results. Section 6 compares the results obtained using *Simics* with those obtained using real web servers. Finally, Section 7 concludes the paper.

2. Related work

Up to not long ago, the methodology used for evaluating commercial workloads in multiprocessors consisted in generating firstly memory references of applications, and then, using these references to feed a user-level simulator such as RSIM [8]. For example, in [13] Ranganathan et al. study the performance of On-Line Transaction Processing (OLTP) and Decision Support Systems (DSS) based on this methodology.

The appearance of full-system simulators like *SimOS* [14] or *Simics* [10] has significantly simplified the evaluation of commercial workloads, as these simulators allow modelling elements such as the operating system, the I/O subsystem and so on. Recently, several studies have appear that use *Simics* as the simulation tool used for their evaluation. In [2], it is presented an exhaustive study of the TPC-C benchmark using this simulator. The authors also identify one of the problems concerned with simulation of commercial applications: the variability they show. A deeper analysis of this problem appears in [4], where the variability is found to be caused by the following reasons:

(1) The operating system can take different scheduling decisions.
(2) The locks may be acquired in different order, which can lead to significant resource contention in one run, but not another.
(3) A transaction can be finished during a measurement interval in one run, but no another.

Alameldeen et al. [3] study five commercial applications, including a static content web server, stressing again the indeterministic behavior of commercial workloads.

The Java-based middleware is characterized in [9]. In this work, Karlsson et al. study memory system behavior of *ECperf* and *SPECjbb* benchmarks using *Simics*. As the main result, they find that the memory primary working sets of these workloads are small compared to other commercial workloads such as On-Line Transaction Processing, and that a large fraction of the working sets are shared between processors.

All the previous works use the UltraSparc processor architecture as the CPU model of the simulated system. It is one of the most important differences between these works and ours, as we pretend to analyse the behavior of a web server based on the x86 *Simics* simulated architecture. To the best of our knowledge no other work has addressed this issue.

3. Simics simulator

Simics is a platform which allows us to develop both hardware and software, providing the necessary components for the simulation of both elements in the same context. The different functionalities offered by *Simics* are grouped into *modules*. A module is a file written in C which implements a class that defines an object type.

This tool allows simulating several architectures (single-processor and SMPs), as well as to execute upon them operating systems and commercial applications, which can vary from benchmarks such as SPEC CPU2000 [15] or TPC-C [16], to desktop applications or games. This is actually one of the most interesting characteristics of the simulator, since it allows that commercial workload running on a multiprocessor system can be easily and accurately simulated, which is much harder to achieve with other simulators like RSIM.

3.1. Simulated architectures

Simics simulates nine processor architectures: UltraSparc II, UltraSparc III, x86, AMD x86-64, Alpha, PowerPC, IA-64, ARM and MIPS. Using the UltraSparc III architecture, we can achieve a theoretical limit of up to 384 processors. The experiments we have performed in this work are based on the x86 architecture, which allows for up to 15 processors.

3.1.1. The Simics/x86 class family of processors

Simics/x86 simulates various x86 class processors, ranging from 486 to Pentium 4, and is capable of booting Linux up to version 2.4, Windows NT 4.0, 2000 and XP in both single-processor and multi-processors (SMP) mode. Simics/x86 includes various PC devices needed to boot the virtual PC such as graphic controllers, bus controllers, floppy and hard disks.

The x86 processors are modelled with a rough level of detail: [1] microarchitectural issues such as the reorder buffer, branch predictor or number of functional units are not considered. Thus, the only difference between processors is the instruction set supported by each model.

3.2. Execution modes and timing interfaces

Simics is an event-driven simulator, which employs a maximum time resolution of one clock cycle. The length of the clock cycle is an user-defined parameter. An event is an interrupt of one device or the execution of one instruction, for example. Events can be scheduled to occur once a specific number of steps have been executed. A step means a completed instruction, an instruction generating an exception, or an external interrupt.

The execution time of an instruction is an important definition. In a *single-processor* machine, there will be exactly one notion of time with an *execution mode* defining instruction execution, and *timing interfaces* controlling the latency of the global execution. When *multiprocessors* are simulated, this becomes somewhat more complex as each processor could have its own notion of time. *Simics* serializes execution when simulating multiprocessors in order to improve performance. This is achieved by dividing the time into segments and by serializing the execution of separate processors within a segment.

The simulator provides two execution modes: in-order and out-of-order execution modes. The

[1] The last release of the simulator includes a more detailed model of the x86 family of processors.

latter is only available for SPARC processors. We focus on in-order execution mode, since our simulations use x86 processors. In this mode, each instruction becomes a single event and instructions are scheduled sequentially in program order. So, when an instruction is stalled a certain number of cycles (due to a cache miss, for example), the instructions that follow it are also stalled until the preceding instruction is completed. When simulating a multiprocessor, it is possible that an instruction is stalled in those cases in which its execution extends across a quantum boundary. In such cases, it is possible for another processor to observe and manipulate the instruction partially executed.

Simics provides two generic abstractions that relate simulated time with the number of executed instructions. When using in-order execution mode, each instruction takes exactly one clock cycle by default. In the multiprocessor case, this means that at any time all the processors have executed the same number of instructions. By default, there are no timing memory system models, so memory access are stalled zero cycles, and each instruction only takes one cycle to complete. The simulator offers the possibility of creating a memory hierarchy, through which this limitation can be removed.

3.3. Memory hierarchy creation

There is a module that implements the *generic-cache* class, which defines the cache type object. Each object of the *generic-cache* class has some attributes that can be configured by the user. Through the *next-level* attribute, it is possible to create a cache hierarchy, making this attribute to point to the next object in the memory hierarchy. The number of levels in the cache hierarchy is not limited, and the only limitation is that the cache memory cannot be shared by two processors.

There are other attributes to control the number of cache rows, row size, associativity, write policy (write-back or write-through), hit time and miss penalty. Likewise, we must indicate which processor is each cache connected to. When a SMP multiprocessor is to be modelled, the simulator employs a four-state memory coherence protocol and snooping caches.

3.4. Network simulation

Simics is also able to simulate several nodes interconnected through a local area network. To this end, it is provided the *ethernet-central* module, which can be considered as the interconnection network (a simulated network). Every time the *ethernet-central* module is executed, it remains waiting for incoming connections. Subsequently, each one of the nodes are simply connected to the network using the suitable command.

4. Working environment

4.1. Apache web server

Apache [5] is a static and dynamic web content server, although in the evaluations performed in this paper it has been used only as static server. The server has been compiled including all the options indicated by the server development group in order to increase performance [6]. Among these options, it is recommended the use of the *worker* multiprocessing module instead of the *prefork* one, which is the default option. After compiling the server with the worker module, the incoming connections are dispatched using threads instead of processes. The server has also been configured to maintain a pool of inactive threads, so that requests are dispatched without waiting for the creation of a new thread.

4.2. Benchmarks used

The *httperf* utility [11] is a tool to measure web server performance. In its basic operation mode, *httperf* generates a fixed number of GET HTTP requests and measures the number of responses and their latency. The most important options offered by this program are:

- it allows simulating different users, that is, it models the concept of *session*. It can specify the number of requests emitted in each session, as well as the frequency of the emissions.
- it allows specifying the rate of connections or sessions created.

- N different requests can refer to N different pages. The performance metric used is the mean response time, which is defined as the mean time between sending the first byte of the request and receiving the first byte of the response.

4.3. Methodology

The evaluations performed have been based on varying the configuration of the server using *Simics* and then, by means of *httperf*, executing the same test for every configuration. This allows carrying out comparisons among the architectural configurations that have been evaluated. Unfortunately, we noted that *httperf* generated requests randomly, which caused that consecutive executions of the tool showed different request traces. In order to ensure the repeatability of the tests, we had to modify *httperf* so that it could generate a file with request traces. This way, making a base test we firstly caught requests that were generated and write them in a file; later tests used this file to retrieve requests in the same order they were generated in the base test.

We solved the variability problem mentioned in Section 2 applying standard statistic techniques [4]. Each test was repeated five times, and the arithmetic average was taken as the result of the test. It was also necessary to use standard deviation as a metric that allows us to discard those tests whose results were too far away from the mean.

4.4. Hardware configuration

In our evaluations, we have considered three different server architectures: two single-processor architectures with L2 cache size of 512 KB and 1024 KB, respectively and a dual-processor architecture in which each processor has a L2 cache of 512 KB. All these servers use Pentium three processors. Table 1 summarizes the basic parameters of the simulated systems.

Both the server and client are virtual hosts simulated by *Simics*; these hosts are interconnected using a 10 Mb/s Ethernet network, which is also simulated by *Simics*.

Table 1
Basic parameters of the simulated systems

Processor speed	500 Mhz
Model	Pentium 3
Cache line size	64 bytes
L1 cache WB	Direct mapped, 32 KB
L1 hit time	2 cycles
L2 cache WB	4-way associative, 512 KB/1024 KB
L2 hit time	10 cycles
Memory size	512 MB
Memory access time	50 cycles
Operating system	Red Hat Linux 7.3
Kernel version	2.4.18-3
Network	Ethernet 10 Mb/s

5. Simulation results and analysis

In this Section, we present the results that have been obtained using *Simics*. First of all, we show the variations in *Apache* response time as a function of the number of requests that are simultaneously sent to the server. Using this metric, we compare the three hardware configurations presented. Then we provide detailed statistics of the CPU and caches for each one of the configurations.

Clearly, this comparison could have been carried out using real computers (as we will see later); however, the use of *Simics* provides more flexibility to easily change system configuration. It is also possible to obtain detailed CPU and cache statistics, something that is harder in a real environment and that allows us to analyse in a deeper way the obtained results.

5.1. Apache response time as a function of the number of requests received

In order to compare the different hardware configurations of the servers, we measure the response time of *Apache* in each case as a function of the number of requests that are received. For this, we have executed two kind of tests:

(1) First, we have executed 1000 requests referred to 10 web pages whose sizes vary between 338 KB and 545 KB, with an average page size of 110 KB.

(2) Second, we have also executed 1000 requests, but these refer 10 web pages with an average page size of 537 bytes.

The motivation behind these two different tests is to check the influence of the interconnection network on the results. In both cases we have firstly generated a file containing the requests that are sent (remember that these requests are generated randomly). Additionally, we must take into account the use of *timeouts* for requests: if after a lapsed time, a response from the server has not been received, the connection is given up for dead and closed, and the total number of timeout errors is increased. The *httperf* program allows specifying a timeout value, and automatically control it. In general, it is suggested a timeout value between 5 and 15 s [12]; we have used a value of ten seconds in the tests.

We have carried out eight tests for each sever architecture, in which the total number of requests that *Apache* must process has been set to 25, 50, 75, 100, 125, 150, 175 and 200, respectively. Fig. 1 shows the average response time. This metric is provided by *httperf*. On the other hand, Fig. 2 presents the evolution of dispatched requests as a function of the total number of requests. This metric is provided by the *Apache* server. In both cases, results are shown for large and small pages.

From Fig. 1(a), we can see how from the httperf's perspective, the two single-processor architectures have almost the same performance than the dual-processor one. However, if we take a look to Fig. 2(a), we notice that the dual-processor server is able to process from 7% to 30% more requests than the single-processor one with a L2 cache of 1024 KB. The difference is even larger when we compare the dual-processor server with the single-processor one with a L2 cache of 512 KB (from 25% to 39%). This behavior is motivated by the communication network that connect the host running *httperf* and the server. The reason is that network requirements for these tests are greater than the bandwith offered by the *Simics* simulated network.

We can also see in Fig. 2(a) that the number of requests dispatched per second decreases once the point of 175 requests per second has been reached, and it happens for all the configurations. This phenomenon is also a consequence of the saturation found in the network, which causes that lower rates of arriving requests are obtained as we increase the rate of requests per second that *httperf* generates.

Comparing the results of Figs. 1(a) and 2(a) with those of Figs. 1(b) and 2(b) we can conclude that the communication network was indeed saturated. In the tests performed over the pages of average size of 537 bytes we can see how the dual-processor server has greater performance than those that employ a single-processor, with an average response time of half the response time of the single-processor severs.

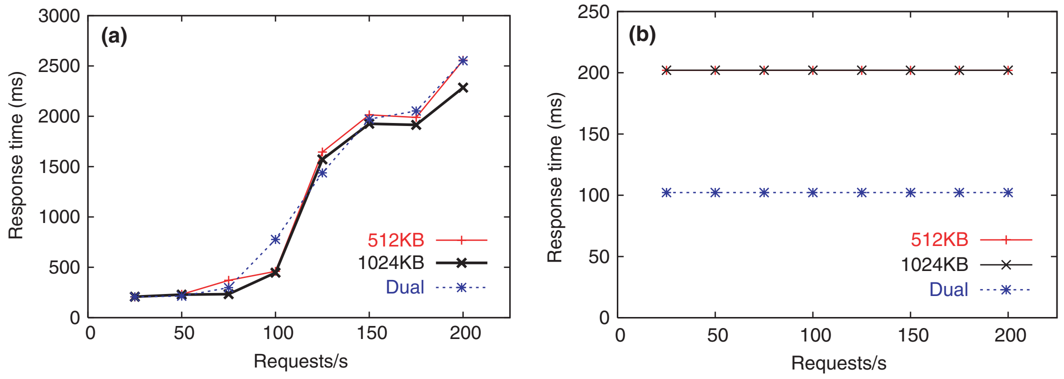

Fig. 1. Average response time as a function of the number of received requests per second. (a) Large pages; (b) small pages.

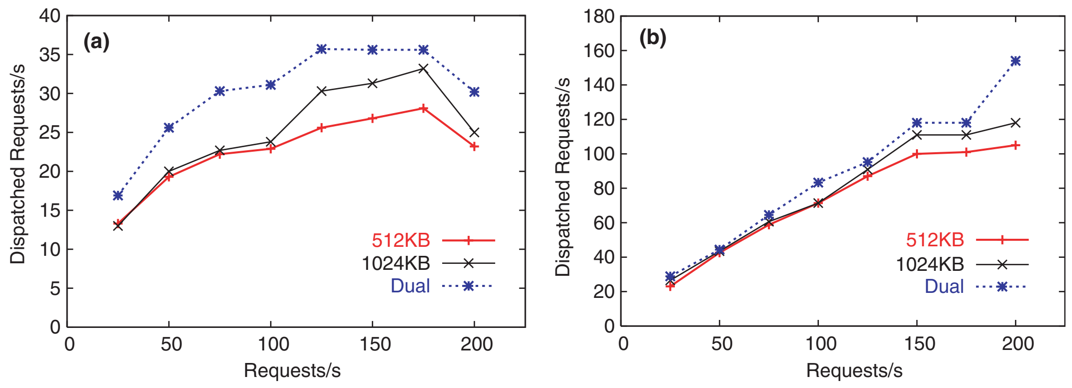

Fig. 2. Dispatched requests per second as a function of the number of received requests per second. (a) Large pages; (b) small pages.

5.2. Detailed statistics obtained with Simics

5.2.1. Server statistics for large pages

CPU statistics. Table 2(a) and (b) show the number of instructions executed by the CPU of the 2 single-processor server architectures. In the same way, Table 3 presents the instructions that are executed by the CPUs of the dual-processor server. In all the cases, the instructions executed in user mode and those executed in supervisor mode are shown in the tables. The first important fact is that the number of instructions executed in user mode is 200 times smaller than the number of instructions executed in supervisor mode. If we compare Tables 2(a) and 3, we notice that the number of instructions executed in user mode is almost the same in the two cases, but it is distributed between the two processors in the case of the dual-processor server.

It does not happen the same with the instructions executed in supervisor mode, since in this case each CPU executes the same number of instructions that the single-processor server. These numbers corroborates the great influence that the operating system has on the final results.

In the case of the single-processor machine with a L2 cache of 512 KB, we see that the number of instructions executed is slightly larger than when a L2 cache of 1024 KB is employed. This increase would be justify by a larger cache miss rate, as we will see next.

Looking at Table 2(a) (or Table 2(b)) we notice that the results are not predictive: 50 requests per second causes 64.306.550 instructions to be executed in user mode while 200 requests per second causes 62.392.239 instructions to be executed. Something similar happens for the instructions executed in supervisor mode. This lack of predictability is mainly motivated by the influence of the operating system.

To understand this influence, we must review the process of serving HTTP requests. When a request comes in, a thread is associated to it. This

Table 2
CPU statistics for the single-processor architectures and large pages

Executed instructions		
Req/s	User mode	Supervisor mode
(a) *L2 cache of 1024 KB*		
25	60.672.719	24.674.350.265
50	64.306.550	14.970.932.911
75	64.334.396	11.758.236.372
100	63.820.362	10.066.423.534
125	63.036.580	10.916.560.146
150	63.973.983	12.197.420.294
175	62.724.364	10.824.982.548
200	62.392.239	11.760.974.821
(b) *L2 cache of 512 KB*		
25	60.644.606	24.749.847.588
50	64.532.373	14.331.897.533
75	63.997.248	12.609.442.919
100	63.351.220	11.998.445.012
125	62.783.306	11.524.716.694
150	62.551.989	12.013.025.991
175	62.343.761	12.886.153.678
200	62.092.364	12.774.560.372

Table 3
CPU statistics for the dual-processor architecture and large pages

Req/s	CPU1 user mode	CPU1 supervisor mode	CPU2 user mode	CPU2 supervisor mode
25	36.390.337	24.691.231.835	25.184.939	24.702.438.061
50	39.683.545	14.999.605.554	24.870.596	15.014.419.404
75	39.895.103	11.753.254.897	25.331.670	11.767.818.330
100	27.013.742	11.184.858.132	37.279.986	11.174.592.014
125	38.362.852	10.414.023.825	25.219.142	10.427.167.858
150	39.533.435	11.003.628.789	24.228.880	11.018.934.120
175	38.612.952	11.003.658.048	24.098.889	11.618.172.111
200	38.702.460	12.257.364.894	24.062.891	12.272.005.109

thread reads the request, parses the URL, finds the file name corresponding to the URL, checks the file states, performs security checking, opens the file, reads its content, and finally, sends the content to the client. Additionally, the thread writes the information of the request to a log file and waits for the next request to come.

In the previous processing, the operating system executes most of the time, performing operations belonging to six main categories [7,18]:

- Low-level networking: Ethernet interrupt and device driver routines.
- High-level networking: TCP, IP, and socket routines.
- Low-level file system: disk interrupt and device driver routines.
- High-level file system: read, write and buffer cache routines. This includes calls to the *open*, *read* and *stat* functions, which are used to open, read and determine file states.
- Timer management: timer routines used by TCP and device drivers. Calls to *alarm* and other similar functions.
- Other: routines for system calls, interrupt handling, scheduling and memory management (including data page faults). This includes calls to the *getpid* and *exit* functions.

If we take a look at Table 2(a) again we notice that the number of instructions executed decreases as the number of requests per second increase. This phenomenon is motivated by the saturation of the network: less requests reach the server, and then, the low-level and high-level networking components described previously appear to a lesser extent. This is true always except in the case of 25 requests per second. In this case, the number of instructions executed in supervisor mode is much larger than in the rest of cases, while the number of instructions executed in user mode is smaller. The explanation for the increased number of instructions being executed in the supervisor mode is that in this test the operating system runs for longer time. As the simulator accounts all the instructions executed by the operating system (not only those implied in executing our test), the overall number is larger than in the rest of the tests. Regarding the number of instructions executed in user mode, it is smaller because the number of threads accessing simultaneously to the resources that are shared (such as the log file) is also smaller, and then, the *Apache* server executes less synchronization instructions.

Cache statistics. The results showed in Tables 4 and 5 correspond to the experiments when 25 requests per second are emitted. Since similar results have been obtained for the rest of the experiments, we just show those that are obtained for one of the cases. Table 4(a), (b) are referred to the single-processor server with 1 MB and 512 KB L2 caches, respectively. On the other hand, Table 5 presents the results that are obtained for the dual-processor server. If we compare the results presented in Table 4(a) with those that are shown in Table 4(b) we see that the most notable difference is the increase in the second level cache miss rate. The increase of 2.5 times in the number of L1 cache invalidations is also a remarkable result. This fact is a consequence of the increase in the number of replacements (what is caused by the increase in the miss rate), which leads to invalidate more

Table 4
Cache statistics for the single-processor architectures and large pages

	L1 Cache	L2 Cache
(a) *L2 cache of 1024 KB*		
Total accesses	853.019.931	59.572.430
Reads	202.450.622	13.986.825
Writes	148.847.460	18.408.032
Instruction fetches	501.721.849	16.181.040
Read misses (%)	6′65%	32′21%
Write misses (%)	4′58%	14′65%
Inst. fetch misses (%)	3′23%	7′91%
Miss rate	4′27%	14′24%
Replacements	36.424.934	8.436.345
Copy backs	10.996.533	3.731.791
Invalidates	24.639	0
(b) *L2 cache of 512 KB*		
Total accesses	849.761.691	59.572.421
Reads	201.546.686	13.947.106
Writes	148.236.166	18.369.073
Instruction fetches	499.978.839	16.300.024
Read misses (%)	6′64%	39′85%
Write misses (%)	4′57%	18′61%
Inst. fetch misses (%)	3′26%	15′36%
Miss rate	4′29%	19′27%
Replacements	36.398.294	11.401.900
Copy backs	10.956.218	4.596.250
Invalidates	60.191	0

L1 blocks in order to maintain the inclusion property.

Finally, we observe results corresponding to the dual-processor server configuration (Table 5). In this case, the large number of L1 cache invalidations must be considered again, although the explanation is just as the previous one. Referring to miss rates, they are just like the preceding ones for first level caches, whereas for the second level caches this rate ranges between the values that are obtained for the single-processor configuration with a L2 cache of 1024 KB and the values obtained for the configuration with a L2 cache of 512 KB.

5.2.2. Server statistics for small pages

CPU statistics. We can see again the disproportion between the number of instructions executed is user mode and supervisor mode in Tables 6 and 7; in this case, the number of instructions executed in supervisor mode is more than 50 times the instructions executed in user mode. As in previous Section, the instructions executed in user mode are divided among the two processors, but not the instructions executed in supervisor mode.

Cache statistics. As we could have expected, the greater miss rate belongs to the single-processor server with a L2 cache of 512 KB, although it is very similar to the miss rate of the dual-processor server (see Tables 8 and 9). One more time, the single-processor server with a L2 cache of 1024 KB has the lower miss rate.

If we compare these data with those that were obtained for large pages, the main difference is found in the second level cache miss rate; this miss rate decreases in a factor of 3.5 times for the case of a single-processor architecture with a L2 cache of 1024 KB, for example. This reduction is a

Table 5
Cache statistics for the dual-processor architecture and large pages

	CPU1 L1 Cache	CPU1 L2 Cache	CPU2 L1 Cache	CPU2 L2 Cache
Total accesses	519.494.022	35.505.482	435.295.180	30.508.935
Reads	111.218.129	8.355.003	97.576.046	7.167.021
Writes	81.950.556	10.293.278	71.646.618	8.896.988
Instruction fetches	326.265.992	10.574.259	266.012.222	9.897.637
Read misses (%)	7′05%	36′1%	4′36%	17′64%
Write misses (%)	4′54%	18′06%	4′4%	17′19%
Inst. fetch misses (%)	3′24%	14′18%	3′34%	13′52%
Miss rate	4′26%	17′95%	4′36%	17′64%
Replacements	22.100.601	6.325.123	18.946.037	5.339.169
Copy backs	6.223.927	2.512.316	5.486.995	2.117.483
Invalidates	33.774	0	28.784	0

Table 6
CPU statistics for the single-processor architectures and small pages

Executed Instructions		
Req/s	User mode	Supervisor mode
(a) *L2 cache of 1024 KB*		
25	110.531.147	10.531.130.567
50	110.219.010	10.089.023.129
75	110.146.230	13.426.854.778
100	109.556.549	10.094.852.236
125	110.095.832	8.095.070.541
150	110.992.009	6.705.120.689
175	110.566.335	5.809.750.245
200	110.008.766	5.097.114.227
(b) *L2 cache of 512 KB*		
25	111.263.145	31.864.745.123
50	110.556.012	20.088.956.124
75	110.322.015	9.817.170.458
100	110.066.187	10.094.368.779
125	110.737.588	2.257.630.148
150	110.605.288	6.761.849.447
175	110.658.169	1.555.847.129
200	110.691.369	5.095.747.259

Table 8
Cache statistics for the single-processor architectures and small pages

	L1 Cache	L2 Cache
(a) *L2 cache of 1024 KB*		
Total accesses	343.139.000	23.850.521
Reads	81.586.315	5.800.108
Writes	52.531.261	4.756.823
Instruction fetches	209.021.714	10.321.827
Read misses (%)	6′76%	4′76%
Write misses (%)	2′88%	5′73%
Inst. fetch misses (%)	4′94%	1′89%
Miss rate	5′06%	3′12%
Replacements	17.350.585	725.114
Copy backs	2.071.763	389.351
Invalidates	24.205	0
(b) *L2 cache of 512 KB*		
Total Accesses	388.210.015	21.542.644
Reads	84.329.449	6.047.956
Writes	53.795.289	4.800.681
Instruction fetches	250.099.513	10.655.567
Read misses (%)	6′85%	8′7%
Write misses (%)	2′77%	6′52%
Inst. fetch misses (%)	4′26%	7′76%
Miss rate	4′62%	7′73%
Replacements	17.922.942	1.656.343
Copy backs	3.038.440	482.331
Invalidates	243.310	0

consequence of the small page sizes that we have used in these experiments, as loading a large page into the L2 cache causes a lot of lines associated with other pages to be replaced. We confirm this observation seeing the number of replacements of the second level cache. There were 8.436.345 replacements for the 1024 KB L2 cache in the experiments with large pages, and only 725.114 in the experiments with small pages.

Regarding the number of L1 invalidations, we notice again a significant increase for the single-processor with 512 KB L2 cache and the dual-processor architectures; as in previous Section, this phenomenon is motivated by the need for maintaining the inclusion property.

6. Comparing simulation results to real servers

Once we have seen how *Simics* can help us to analyse the behavior of a commercial web server, we want to check how accurate are the results that the simulator provides. In order to accomplish

Table 7
CPU statistics for the dual-processor architecture and small pages

Req/s	CPU1 user mode	CPU1 supervisor mode	CPU2 user mode	CPU2 supervisor mode
25	56.418.325	20.043.568.234	54.264.897	20.045.741.115
50	87.483.114	13.349.235.177	22.946.114	13.413.745.125
75	91.424.012	10.013.200.025	18.818.906	10.085.876.125
100	55.158.326	19.021.256.214	54.788.951	20.015.625.559
125	58.265.905	8.047.090.235	51.207.354	8.045.145.169
150	67.867.524	6.705.120.523	42.228.847	6.730.764.125
175	56.953.145	5.763.010.215	53.228.415	5.766.763.585
200	78.363.600	5.028.520.456	31.469.012	5.075.421.064

Table 9
Cache statistics for the dual-processor architecture and small pages

	CPU1 L1 Cache	CPU1 L2 Cache	CPU2 L1 Cache	CPU2 L2 Cache
Total accesses	294.767.000	19.339.960	99.172.100	6.423.026
Reads	66.055.838	4.856.005	20.571.934	1.605.217
Writes	41.668.672	3.771.544	13.253.933	1.352.386
Instruction fetches	187.013.184	8.295.460	65.312.828	2.566.539
Read misses (%)	7%	10′71%	7′58%	14′11%
Write misses (%)	2′72%	9′3%	3′06%	13′63%
Inst. fetch misses (%)	4′44%	5′63%	3′93%	3′66%
Miss rate	4′75%	6′91%	4′57%	7′86%
Replacements	13.909.760	1.319.439	4.279.917	495.379
Copy backs	2.386.793	643.346	862.778	343.545
Invalidates	85.563	0	136.619	0

this, we have repeated the experiments presented in Section 5, but this time we have employed real computers. In particular, we have evaluated only the single-processor architecture with an L2 cache of 512 KB, and the dual-processor server. Hardware configurations of the computers used in these experiments are the same as the ones used in the previous case. The operating system used in the experiments also coincide with the employed previously, as well as the versions of the *Apache* web server and the *httperf* tool. Figs. 3 and 4 show the results we have obtained in these experiments.

Comparing these results to the obtained with *Simics*, we find that there are notable differences between them. In the case of the response time, it is scaled down by a factor of almost 100 times in the study of small pages (Figs. 1(b) and 3(b)). In fact, the performance difference between dual and single-processor real servers is negligible, in contrast to the results obtained in the simulations, where the response time for the dual-processor server is approximately one half of the response time for the single-processor architectures (Fig. 1(b)). In the experiments performed on real machines we have observed that issuing 200 requests per second does not stress the CPU of the single-processor server, so we cannot take advantage of using two CPUs.

Something similar occurs with the number of requests that are dispatched. Although simulation results showed that the dual-processor server could sustain a larger requests per second rate than the single-processor one (Fig. 2(a)), in the real environment we find that for the experiments we

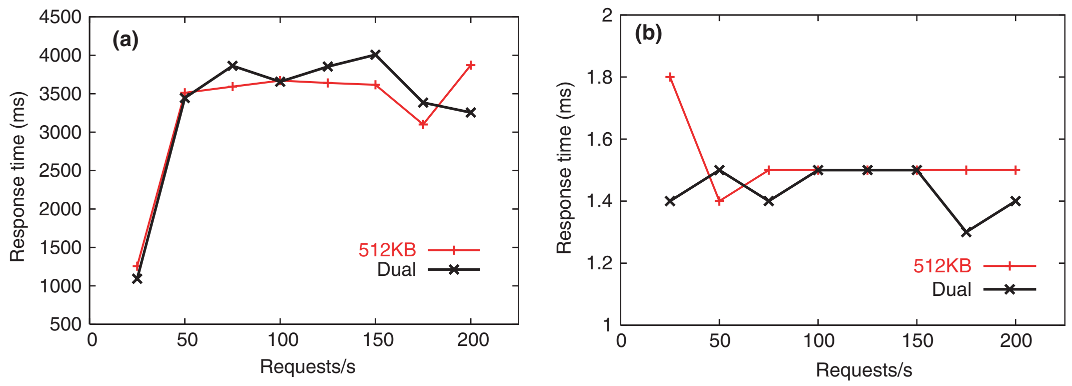

Fig. 3. Average response time as a function of the number of received requests per second for the real case. (a) Large pages; (b) small pages.

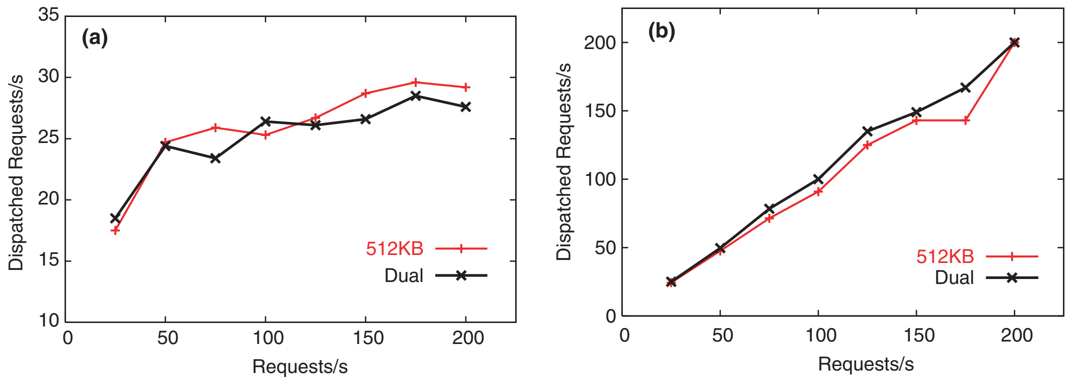

Fig. 4. Dispatched requests per second as a function of the number of received requests per second for the real case. (a) Large pages; (b) small pages.

have carried out, single and dual-processor servers provide almost the same results in terms of the number of requests that are dispatched (Fig. 4).

In this way, we can conclude from the results that the rough detail level when modeling in-order execution x86-like processors prevents *Simics* from be able to reproduce the results that would be reached in the real world. In this way, we think that the in-order execution *x86-Simics* machine is appropriate as functional simulator but not as timing simulator.

7. Conclusions

Multiprocessor systems are increasingly being used for executing commercial applications, such as databases and web servers, so it becomes essential to study the behavior of multiprocessor architectures under commercial workloads. For this, we need simulators able to model not only the CPU, memory and interconnection network but also other aspects that are critical in the execution of commercial workloads, such as I/O subsystem and operating system.

In this paper we have introduced the evaluation of a functional simulator which allows us to simulate all these elements and we have also shown how the simulator can help us to obtain CPU and memory hierarchy statistics of the simulated system. However, we have found that the simulator does not provide an accurate model of the x86 family of processors when using the in-order execution mode, which leads to obtain different results that those that would be obtained using real computers. We think that the impossibility of using an out-of-order execution model for this family has a negative influence in the results that we have obtained. This does not happen with the UltraSparc family of processors, as we can see in [2,3]. Other limitations that prevent us to do a more rigorous study of commercial workloads execution are the impossibility of simulating cc-NUMA architectures and obtaining a detailed cache miss taxonomy. In short, we think *Simics* is insufficient in order to evaluate commercial workloads at the present time, and that it has to be extended with more functionality if we want to use it for this purpose.

As future work we are modifying the source code of the simulator in order to include new modules that will allow us to extend the work presented in this paper. Specifically, we are extending the cache model of the simulator in order to obtain a cache miss taxonomy similar to the proposed in [1]. We also want to include a mechanism that integrates *Simics* with a user-level simulator, in particular with RSIM.

Acknowledgments

The authors would like to thank the anonymous referees for their comments and suggestions, which have helped to improve the quality of the paper. This work has been supported in part by the Spanish Ministry of Ciencia y Tecnología and the European Union (Feder Funds) under grant TIC2003-08154-C06-03.

References

[1] M.E. Acacio, J. González, J.M. García, J. Duato, A novel approach to reduce l2 miss latency in shared-memory multiprocessors, in: International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida, April 2002.

[2] A.R. Alameldeen, M.M.K. Martin, C.J. Mauer, K.E. Moore, M. Xu, M.D. Hill, D.A. Wood, D.J. Sorin, Simulating a $2M Commercial Server on a $2K PC, IEEE Computer (February) (2003).

[3] A.R. Alameldeen, C.J. Mauer, M. Xu, P.J. Harper, M.M.K. Martin, D.J. Sorin, M.D. Hill, D.A. Wood, Evaluating non-deterministic multi-threaded commercial workloads, in: 5th Workshop Computer Architecture Evaluation using Commercial Workloads (CAECW-02), February 2002.

[4] A.R. Alameldeen, D.A. Wood, Variability in architectural simulations of multi-threaded workloads, in: 9th International Symposium on High-Performance Computer Architecture (HPCA-9), Anaheim, CA, February 2003.

[5] Apache HTTP Server Project. Available from <http://httpd.apache.org>.

[6] Apache Performance Notes. Available from <http://httpd.apache.org/docs/misc/perf-tuning.html>.

[7] Y. Hu, A. Nanda, Q. Yang, Measurement, analysis and performance improvement of the apache web server, in: 18th IEEE International Performance, Computing and Communications Conference (IPCCC'99), Phoenix/Scottsdale, Arizona, February 1999.

[8] C.J. Hughes, V.S.P. Pai, P. Ranganathan, S.V. Adve, RSIM: simulating shared-memory multiprocessors with ILP proccesors, IEEE Computer (February) (2002) 68–76.

[9] M. Karlsson, K.E. Moore, E. Hagersten, D.A. Wood, Memory system behavior of java-based middleware, in: 9th Annual International Symposium on High-Performance Computer Architecture (HPCA-9), Anaheim, CA, February 2003.

[10] P.S. Magnusson, M. Christensson, J. Ekilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, B. Werner, Simics: a full system simulation platform, IEEE Computer (February) (2002) 50–58.

[11] D. Mosberger, T. Jin, httperf: A tool for measuring web server performance, Performance Evaluation Review (December) (1998) 31–37.

[12] D. Mosberger and T. Jin. httperf man pages, March 1998.

[13] P. Ranganathan, K. Gharachorloo, S.V. Adve, L.A. Barroso, Performance of database workloads on shared-memory systems with out-of-order processors, in: 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), October 1998, pp. 307–318.

[14] M. Rosemblum, S. Herrod, E. Witchel, A. Gupta, Complete computer system Simulation: the SimOS approach, IEEE Parallel & Distributed Technology: Systems & Applications (Winter) (1995) 34–43.

[15] SPEC CPU2000 User's Guide, 2000.

[16] Transaction Processing Performance Council. TPC Benchmark C, Standard Specification, Revision 5.0, 2001.

[17] S.C. Woo, M. Ohara, E. Torrie, J.P Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations in: 22nd International Symposium on Computer Architecture, June 1995, pp. 24–36.

[18] D.J. Yates, V. Almeida, J.M. Almeida, On the interaction between an operating system and web server. Technical Report CS 97-012, Boston University, July 1997.

Francisco J. Villa received the MS degree in Computer Science in 2003 from the University of Murcia in Spain. Since 2003 he is a PhD student at the Research Group on Parallel Computing Architecture, working on evaluating and designing high-performance memory hierarchies for multi-processor-on-a-chip architectures. His research interests are multiprocessor memory systems, chip-multiprocessor architectures, and power-aware cache-coherence protocol design.

José M. García received the MS and the PhD degrees in electrical engineering from the Technical University of Valencia (Spain), in 1987 and 1991, respectively. At present, Dr. García is a Full Professor at the Computer Engineering Department of the Universidad de Murcia (Spain), and also the Head of the Research Group on Parallel Computing Architecture. He specializes in computer architecture, parallel processing and interconnection networks. He has developed several courses on computer structure, peripheral devices, computer architecture, and multicomputer design. Dr. García served as Vice-dean of the School of Computer Science from 1995 to 1997, and also as Director of the Computer Engineering Department from 1998 to 2004. His current research interests lie in high-performance coherence protocols for shared-memory multiprocessor systems, and high-speed interconnection networks. He has published more than 60 refereed papers in different journals and conferences in these fields. Dr. García is member of several international associations such as the IEEE and ACM, and also member of some European associations (Euromicro and ATI).

Manuel E. Acacio received the MS and PhD degrees in computer science from the Universidad de Murcia, Spain, in 1998 and 2003, respectively. He joined the Computer Engineering Department, Universidad de Murcia, in 1998, where he is currently an Assistant Professor of computer architecture and technology. His research interests include prediction and speculation in multiprocessor memory systems, multiprocessor-on-a-chip architectures, and power-aware cache-coherence protocol design.