Simulation of Cache Memory Systems on Symmetric Multiprocessors with Educational Purposes

Miguel A. Vega Rodríguez; Juan M. Sánchez Pérez; Raúl Martín de la Montaña; Francisco A. Zarallo Gallardo
Departamento de Informática
Universidad de Extremadura
Escuela Politécnica. 10071 Cáceres. Spain
E-mail: mavega@unex.es; Telephone: 0034 - 927 257263; Fax: 0034 - 927 257202

1. Resumen

En este documento presentamos un simulador para el análisis y la docencia de sistemas de memoria caché en multiprocesadores simétricos. La simulación se apoya en un modelo construido de acuerdo con los principios básicos arquitectónicos de estos sistemas. El simulador, denominado SMPCACHE, posee un interfaz gráfico completo y amigable, y opera en sistemas PC con Windows 98 (ó 95). Las experiencias más recientes nos han demostrado los beneficios del simulador con fines didácticos. Por un lado, los estudiantes lo han utilizado para experimentar los diferentes aspectos teóricos sobre memorias cachés y multiprocesadores en los cursos oficiales de Arquitectura de Computadores. Algunos de los parámetros que pueden estudiar con el simulador son: Localidad de los programas; influencia del número de procesadores, de los protocolos de coherencia caché, de los esquemas para arbitración del bus, de la función de correspondencia, de las políticas de reemplazo, del tamaño de caché (bloques en caché), del número de conjuntos en caché (para cachés asociativas por conjuntos), del número de palabras por bloque (tamaño del bloque de memoria), del ancho de palabra,... Por otro lado, los estudiantes deben construir una versión reducida del simulador. Para ello, se les da un conjunto de enunciados. Estos enunciados explican, paso a paso, todas las operaciones y algoritmos que los alumnos deben utilizar para construir un simulador similar a SMPCACHE. Además, los alumnos pueden usar SMPCACHE como referencia para testear sus propios simuladores. La construcción del simulador obliga a los estudiantes a conocer en profundidad las consideraciones teóricas sobre sistemas de memoria caché, y particularmente dentro de entornos multiprocesador (coherencia caché, protocolos de coherencia caché,...). De esta forma, hemos observado que los alumnos adquieren un conocimiento mejor y más amplio sobre estos aspectos. En conclusión, la utilización del simulador es una innovación en las clases prácticas que produce una mejora en la calidad de la educación. Además, es un claro ejemplo de cómo los resultados de la investigación desarrollada pueden revertir en la enseñanza.

2. Abstract

In this work we present a simulator for the analysis and teaching of cache memory systems on symmetric multiprocessors. The simulation is based on a model built according to the architectural basic principles of these systems. The simulator, called SMPCACHE, has a full graphic and friendly interface, and it operates on PC systems with Windows 98 (or 95). The most immediate experiences have demonstrated us the simulator benefits with didactic goals. On the one hand, students have used it for experimenting the different theoretical aspects about cache memories and multiprocessors in the regular courses of Computer Architecture. Some of the parameters that they can study with the simulator are: Program locality; influence of the number of processors, cache coherence protocols, schemes for bus arbitration, mapping, replacement policies, cache size (blocks in cache), number of cache sets (for set associative caches), number of words by block (memory block size), word wide,... On the other hand, students must build a reduced version of the simulator. For that, we give them a set of notes. These notes explain, step by step, all operations and algorithms students must use in order to build a simulator similar to SMPCACHE. Furthermore, students can use SMPCACHE as reference for checking their own simulators. The construction of the simulator forces students to know in depth the theoretical considerations about cache memory systems, and particularly on multiprocessor environments (cache coherence, cache coherence protocols,...). In this way, we have observed that students acquire a better and larger knowledge about these aspects. In conclusion, the use of the simulator is an innovation in the practical classes that produces an improvement of the quality in education. Furthermore, it is a clear example of how the results of the developed research can revert in the teaching.

3. Introduction

The performance of a computer system is a function of the speed of the individual functional units, and of the workload presented to the system. It is well known that caches are a critical component in the performance of any computer system. Cache is the simplest cost effective way to achieve high speed memory and its performance is extremely vital for high speed computers. This is the reason why caches are a investigation issue very active (for a partial bibliography see [1]), and why caches are studied by Computer Science students of any university.

The most prevalent form of parallel architecture is the multiprocessor of small to moderate scale that provides a global physical address space and symmetric access to all of main memory from any processor, often called a symmetric multiprocessor or SMP [2]. At present, among the categories of memory hierarchies on multiprocessors the most widespread is the bus-based shared memory approach. This is the reason why we focus on symmetric multiprocessors with bus-based shared memory in the practical classes about cache memories and multiprocessors. These systems exploit the fundamental properties of a bus to solve the cache coherence problem. In fact, many existing multiprocessors ([3], [4], [5], [6], [7] and [8]) rely on a shared bus and use a snoopy protocol to keep the caches coherent ([2] and [9]).

Trace-driven simulation is often a cost-effective way to estimate the performance of computer system designs. Especially when designing caches, TLBs (Translation-Lookaside Buffer), or paging systems, trace-driven simulation is a very popular way to study and evaluate computer architectures, obtaining an acceptable estimation of performance before a system is built. For the past two decades, a primary means of cache memory analysis has been the use of traces of memory access patterns to drive simulators that determine the miss rate of different cache designs.

In this paper we present a trace-driven simulator for cache memory systems on symmetric multiprocessors with educational purposes. There are well known cache memory simulators, like these ones: TYCHO, DINEROIII ([10], [11] and [12]), ACS (Acme Cache Simulator), SISMEC [13] and bigDIRN. These five simulators are trace-driven simulators. All of them are simulators working on a uniprocessor environment except bigDIRN. So, only bigDIRN could be appropiate for our aims. bigDIRN was produced at MIT and it is a simulator on multiprocessor environment that uses a directory scheme. It is a command line simulator for UNIX that produces a numeric table with a set of statistical data. This simulator has a set of drawbacks about its portability, interface and analysis of characteristics:

- It is an UNIX simulator, so, it has a restricted portability because students usually work on PC systems with Windows.
- An interface limited to provide input parameters in command line is uncomfortable for building memory models. bigDIRN is inappropiate for didactic goals because it has not diversification and adaptation (graphic representation,...) in the exhibition of the simulation results, and a suitable user-simulator interaction.
- The analysis consists in the final results of some parameters, without showing how these parameters evolve during the execution of the programs. bigDIRN does not allow us to store the simulator configuration data, set a default initial configuration for the simulator, modify the multiprocessor memory model and select different traces without going out of simulator, do different kinds of simulation (step by step, with breakpoint and complete execution), etc.

All these limitations were the basis for developing a new simulator. The design considerations have been oriented to satisfy a set of characteristics not only with didactic goals, but also with research goals. The didactic goal tries to approach to the students the problematic of cache memories on SMPs, shown graphically all knowledge they may acquire in diverse texts. In order to do that, it has been necessary to built a friendly interface that allows to develop all analysis job. The research goal is necessary in order to use the simulator in jobs such as analysis of program locality, behaviour of different architectures, developing of design strategies, evaluation of replacement policies, cache coherence protocols, etc.

In the next section we mention the different characteristics the simulator offers to the user. In section 5 are some of the practical experimentations that the students must do using the simulator. Furthermore, students must build

a limited version of the simulator. For that, we give them a set of notes. These notes explain, step by step, all operations and algorithms students must use in order to build a simulator similar to SMPCACHE. Section 6 gives an overview about these notes. Finally, conclusions and future work are presented in the last section. The theoretical considerations about cache memory systems, and particularly about them on multiprocessor environments, are well shown in many computer architecture texts ([2], [9], [14], [15], [16], [17] and [18]), and we will not mention them here. All operations and algorithms we use are similar to those found in these computer architecture texts. So, the results obtained with the simulator have a very close mapping with the real world.

4. Simulator characteristics

In this section we expose the main characteristics of SMPCACHE 1.0. It is a trace-driven simulator for cache memory systems on symmetric multiprocessors with bus-based shared memory that operates on PC systems with Windows 98 (or 95), and it has been written with a visual language. The simulator offers a Windows typical graphic interface, with menus, toolbars, statusbar,... and it has a very complete contextual help system. It is possible to use mouse and keyboard, with a lot of accelerator keys. Fig. 1(a) shows a global vision for the graphic interface.

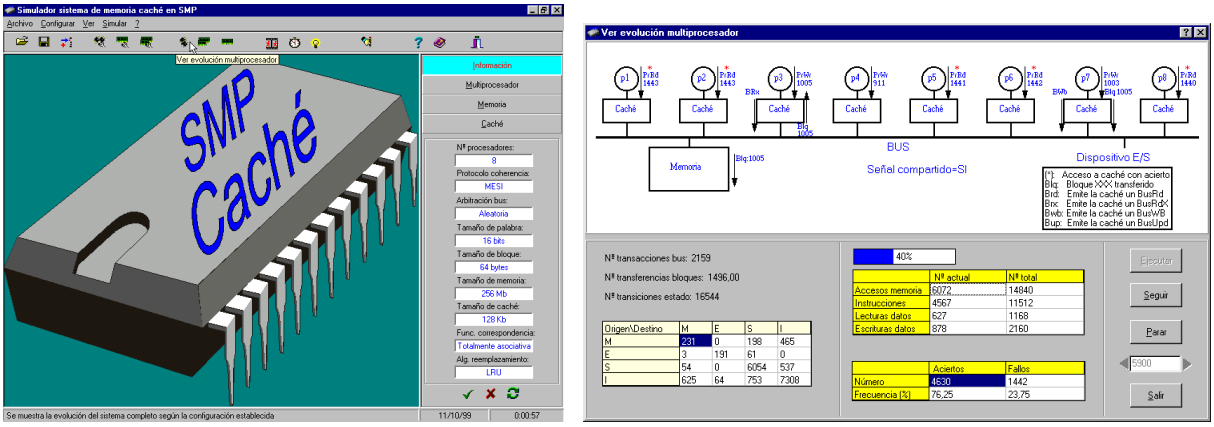

Fig. 1 - (a) Graphic interface of the simulator. (b) Multiprocessor vision during a simulation

The simulator allows us to select the different choices for configuring a given architecture (number of processors, coherence protocol, word wide, words by block, blocks in main memory, blocks in cache, mapping, replacement policies,...), and it gives us the system response for the memory accesses generated by the programs (memory traces used for the different processors during the simulation). So, it is an application that could be used in order to evaluate memory systems on multiprocessors with research goals.

Due to its easy and friendly interface, the simulator is being used with didactic goals; since it allows us to observe, in a clear and graphic way, how the multiprocessor evolves as the program execution goes forward (the memory traces are read). With the simulator, it is possible to obtain a global vision of the multiprocessor evolution (see Fig. 1(b)), a vision of the evolution of a particular cache, or even, a vision of the evolution of a specific memory block (Fig. 2(a)). Showing, all the time, memory accesses demanded by every processor, bus state, state of every cache, state of every memory block within every cache,...

The simulator let us study the best memory system that fit in with our needs before a real implementation, or simply, it helps us to simulate real systems in order to see their efficiency and compare results in a easy way. Some of the parameters we can study with the simulator are:

- Analysis of program locality.
- Influence of number of processors.

- Influence of cache coherence protocols.
- Influence of schemes for bus arbitration.
- Influence of mapping.
- Influence of replacement policies.
- Influence of cache size (blocks in cache).
- Influence of number of cache sets (for set associative caches).
- Influence of number of words by block (block size).
- Influence of word wide.

SMPCACHE shows, using statistical data and several kinds of graphics (Fig. 2(b) is an example), interesting measurements like:

- Number of bus transactions (it depends on cache coherence protocol).
- Number of block transfers on the bus.
- Bus traffic taking into account the previous measurements.
- Number of state transitions (each block in a cache has a state associated with it, the state transitions specify how the disposition of a block changes).
- Number of state transitions from a particular state to other.
- Global number of memory accesses, and for types: instruction captures, data readings and data writings.
- Number of cache hits and misses, as well as hit and miss rate.

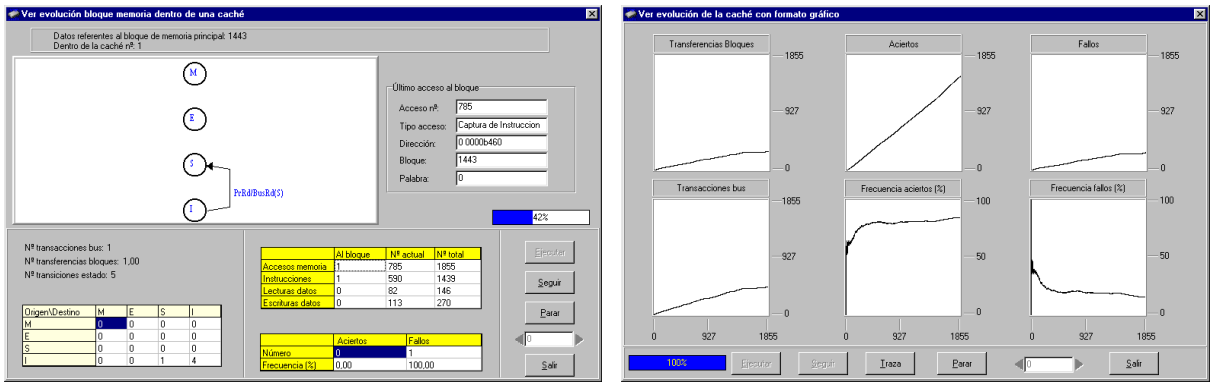

Fig. 2 - (a) State transition diagram for a particular cache block. (b) Data in graphic format for a specific cache

All these data are shown at very different vision levels, although in all cases, the relation among all the multiprocessor elements is took into account. So, we can realize a simulation:

- Observing the complete multiprocessor and all the memory blocks.
- Observing a specific cache and all the memory blocks.
- Observing the complete multiprocessor, but focusing on a particular block.
- Observing a specific cache and a particular memory block.

The simulation consists in the programmed reproduction of operations that would be really perfomed by the components of cache memory system on a real multiprocessor. For that, the adequate computations are performed and, at the same time, the present and accumulated results are shown. There are three kinds of simulation and it is possible to change from one to other without waiting the simulation end:

- *Step by step*. The simulation is stopped after every memory access.
- *With breakpoint*. The simulation is stopped when it is reached a specific number of memory accesses.
- *Complete execution*. The simulation is stopped when all the memory traces are finished.

It is also possible to abort the simulation in any time, for correcting any architectural detail.

4.1. Hardware organization

In relation to the organization characteristics on the memory hierarchy and the multiprocessor, the simulator offers the possibilities presented in Table 1.

Processors in SMP	1, 2, 3, 4, 5, 6, 7 or 8
Cache coherence protocols	MSI, MESI or DRAGON
Schemes for bus arbitration	Random, LRU or LFU
Word wide (bits)	8, 16, 32 or 64
Words by block	1, 2, 4, 8, 16, 32, 64, 128, 256, 512 or 1024
Blocks in main memory	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152 or 4194304
Blocks in cache	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 or 2048
Mapping	Direct, Set-Associative or Fully-Associative
Cache sets (for set associative caches)	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 or 2048
Replacement policies	Random, LRU, FIFO or LFU
Writing strategies	Write-Back (for cache coherence protocols used)
Cache levels in the memory hierarchy	1
References	To memory words
Maximum block size	8 KB
Maximum main memory size	32 GB
Maximum cache size (excluded labels, block state bits, counts, etc.)	16 MB

Table 1 - Architectural characteristics supported by the simulator

All these configuration parameters are related among them according to the theoretical models. The simulator avoids and warns the user made a choice whose value be contradictory with the choice for other parameters.

The different choices selected in the simulator for configuring a given architecture may be stored on a ASCII data file for a future load, so the need of making many selections for configuring the same memory model is avoided. Even more, it is possible to set a default initial configuration for the simulator. These characteristics allow us to build a database with different memory organizations, emulating architectures like Silicon Graphics, MIPS, Sequent Symmetry, PowerPC, Sparc, Sun, Pentium, and others.

4.2. Memory traces

We dispose of a wide set of memory traces for studying locality, coherence protocols, and better cache organizations for a certain type of programs. Many of these traces come from tests performed with benchmarks and application programs on different real (uniprocessor and multiprocessor) architectures on the Parallel Architecture Research Laboratory (PARL), New Mexico State University (NMSU), and they are availables by anonymous ftp to *tracebase.nmsu.edu*. The traces have different formats like dinero (developed by Mark Hill at U.C. Berkeley), PDATS [19], or the canonical format for multiprocessor traces developed by Anant Agarwal.

5. Practical experimentations

Although it is difficult to validate a simulator, we have two reasons for expecting reliable simulation results. On the one hand, the simulator gives us consistent results with the theory about cache memory systems. For example, if the obtained miss rate decreases as the cache size increases, we can say that, in this aspect, the simulator is correct. However, the validation must be more complete because there are multiple and interdependent parameters in these systems. So, on the other hand, in order to validate the simulation results, we have compared our experimental results with those found by other authors ([2], [9], [10], [11], [14], [16], [20], [21] and [22]). This comparison has been satisfactory. After validating the simulator, students are using it for

experimenting the different theoretical aspects about cache memories and multiprocessors. To do that, they are doing similar experimentations to those we have done for validation. The main difference is the length of the traces, due to the limited duration of practical classes. In this section we show the experimentations for a set of determinate architectures with a set of concrete traces.

5.1. Uniprocessor traces

First, we exhibit the practical experimentations when we configure the simulator with a single processor. So, we can use uniprocessor traces, and it is possible to validate and analyse the basic algorithms that are present in every cache memory system, uniprocessor or multiprocessor.

For this first set of experiments we consider traces based on the first thousands of memory accesses of some SPEC'92 benchmarks (*hydro*, *nasa7*, *cexp*, *mdljd*, *ear*, *comp*, *wave*, *swm* and *ucomp*), according to real tests made on a MIPS R2000 system. The traces used represent a wide variety of "real" application programs. A summary of the traces is given in Table 2. These traces were provided by Nadeem Malik of IBM. Send e-mail to tracebase@nmsu.edu for information on the availability of these and other traces.

Name	Classification	Language	Description
hydro	Floating point	---	Astrophysics: Hydrodynamic Naiver Stokes equations
nasa7	Floating point	Fortran	A collection of 7 kernels. For each kernel, the program generates its own input data, performs the kernel and compares the result against an expected result
cexp	Integer	C	Portion of a Gnu C compiler that exhibits strong random behaviour
mdljd	Floating point	Fortran	Solves the equations of motion for a model of 500 atoms interacting through the idealized Lennard-Jones potential. It is a numerical program that exhibits mixed looping and random behaviour
ear	Floating point	---	---
comp	Integer	C	Uses Lempel-Ziv coding for data compression. Compresses a 1 MB file 20 times
wave	Floating point	Fortran	Solves Maxwell's equations and electromagnetic particle equations of motion
swm	Floating point	Fortran	Solves a system of shallow water equations using finite difference approximations on a 256*256 grid
ucomp	Integer	C	The uncompress version of *comp*

Table 2 - Uniprocessor traces

At first, we consider a simple architecture of word wide of 16 bits, 32-byte blocks, with a fully associative cache and LRU replacement. On Fig. 3(a) is shown miss rate vs. cache size. The miss rate decreases as the cache size increases, as we can see for all benchmarks, and this indicates that, independently of different locality grades, all programs have the same general behaviour. This is consistent with the theory because capacity and conflict[1] (collision) misses are reduced by enlarging the cache. Also, it may be observed that for great cache sizes, the miss rate is stabilised, this shows the compulsory or cold start misses, which are independent of cache size. The great differences of miss rate for a determinate increment of cache size indicates the memory addresses are so near that a little increase of cache size brings about a great increase of performance. Clearly, this point depends on the program.

[1] Notice that, in this experimentation, there are not conflict misses because we use a fully associative placement.

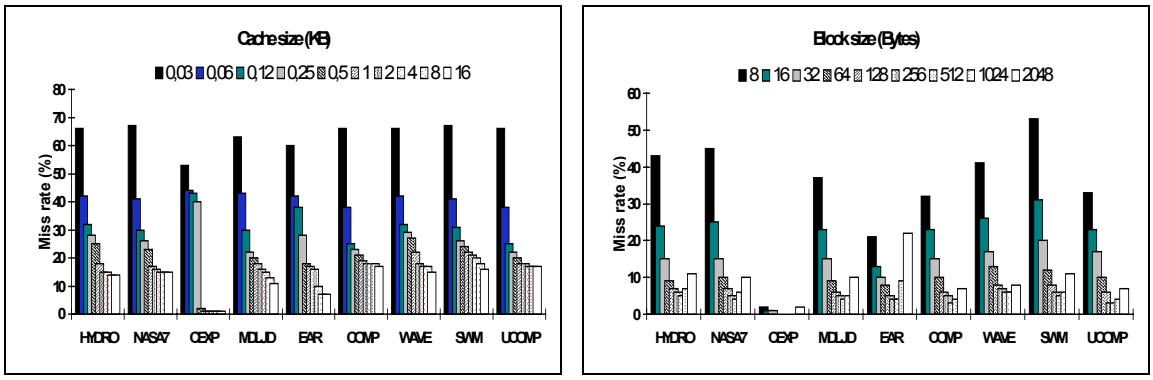

Fig. 3 - (a) Miss rate versus cache size. (b) Miss rate versus block size

Now we study the influence of the block size on the miss rate. The same architecture is considered (16-bit words, fully associative cache and LRU replacement). We have gathered the results shown in Fig. 3(b). These results are fully consistent with the theory. Increasing the block size, the greater number of words in the block, the less miss rate, because of spatial locality since the probability, in a near future, for accessing close data to the accessed word increases. Furthermore, compulsory misses decrease due to reduction of the number of cache lines. However, from a given time (pollution point), the greater number of words in the block, the more stride between some words, so referencing one, the probability for accessing the stridest one decreases. These words with low probability of use displace from cache useful information which will be referenced again, and increase the miss rate.

For *ear* benchmark and with the same architecture, we study again the influence of the block size on the miss rate, but in this case, for serveral cache sizes. Fig. 4(a) shows us the existence of a pollution point, which influence is smaller when the cache size augments. Obviously, the more cache capacity, the less negative effects when the block size increases.

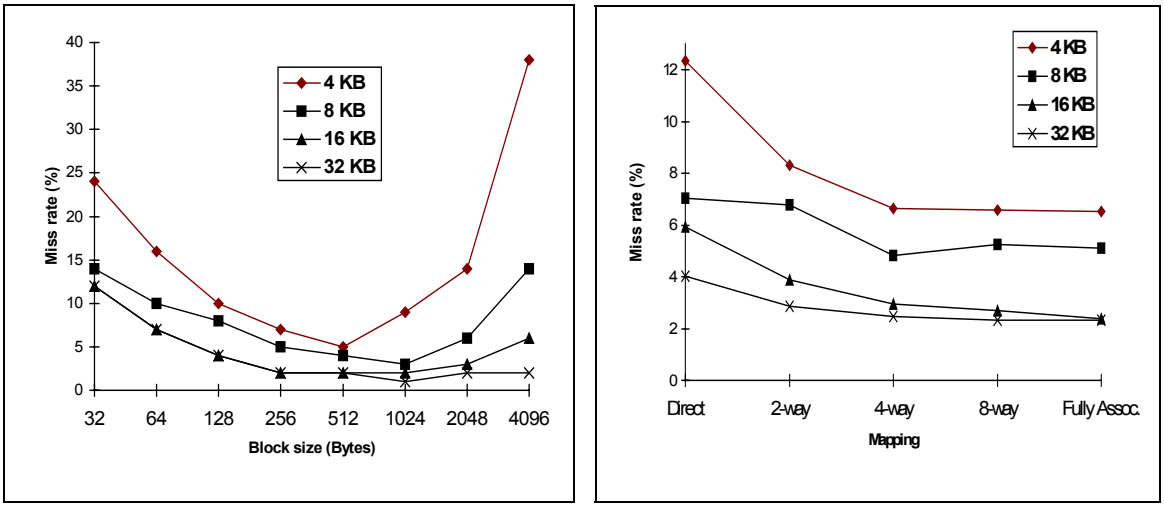

Fig. 4 - (a) Miss rate vs. block size for different cache sizes. (b) Miss rate vs. mapping for different cache sizes

Finally, in order to study the influence of mapping we have used other architecture (32-bit words and 64-word blocks). Fig. 4(b) presents the found results using the *ear* benchmark. It represents miss rate versus different mapping grades (with LRU replacement for the associative mapping case). This figure indicates that the miss rate improves when the associativity grade increases, although, with larger caches, the benefits are every time less significant. This conclusion coincides with the theory because conflict misses are reduced by increasing the

associativity, however, in large caches this kind of misses are less frequent. While the benefit of going from one-way (direct mapped) to two-way set associative is significant, the benefits of further associativity shrink.

5.2. Multiprocessor traces

After validating and analysing the basic algorithms that are present in every cache memory system (uniprocessor or multiprocessor), we exhibit the found results when we configure the simulator with more than one processor and we use multiprocessor traces.

For this second set of experiments we consider traces with tens of millions of memory accesses (references) for four benchmarks (*FFT*, *Simple*, *Speech* and *Weather*). Students consider a smaller portion of every trace in the practical classes. These traces were provided by David Chaiken (then of MIT) for NMSU PARL. The traces represent several real parallel applications ([23] gives details about these parallel applications for *FFT*, *Simple* and *Weather* traces). *FFT*, *Simple* and *Weather* traces were generated using the post-mortem scheme implemented by Mathews Cherian with Kimming So at IBM. Kirk Johnson and David Kranz (both at MIT) are responsible for the *Speech* trace. Send e-mail to tracebase@nmsu.edu for information on the availability of these and other traces. A summary of the traces is shown in Table 3.

Name	References	Language	Description
FFT	7,451,717	Fortran	Parallel application that simulates the fluid dynamics with FFT
Simple	27,030,092	Fortran	Parallel version of the SIMPLE application
Speech	11,771,664	---	---
Weather	31,764,036	Fortran	Parallel version of the WEATHER application, which is used for weather forecasting. The serial version is from NASA Space Flight Center, Greenbelt, Md.

Table 3 - Multiprocessor traces

At first, we consider a multiprocessor architecture with eight processors, MESI (or Illinois, for a detailed description see [2], [17], [18] and [24]) cache coherence protocol, 16-bit words, 64-byte blocks, four-way set associative caches and LRU replacement. On Fig. 5(a) is represented miss rate versus cache size. From this figure we can obtain the same conclusions than for the uniprocessor traces (Fig. 3(a)). The global miss rate for the system decreases as the cache size increases because capacity and conflict misses are reduced. For great caches sizes, the miss rate is stabilised, this shows the compulsory and coherence[2] misses, which are independent of cache size. Current measurements demonstrate that the shared data have less spatial and temporal locality than other data types. That is, in general, parallel programs exhibit less spatial and temporal locality than serial programs. So, it is normal that the miss rates are greater for multiprocessor traces than for uniprocessor traces (for an example compare Fig. 3(a) and Fig. 5(a)).

Fig. 5(b) shows the bus traffic on the multiprocessor per memory access for this same experimentation. The traffic is split into data traffic and address (including command) bus traffic. The obtained conclusions are similar to the previous ones for the miss rate. The bus traffic is reduced as the miss rate decreases because of two fundamental reasons. On the one hand, there are less data transfers from the shared main memory to the caches. On the other hand, due to there are less misses, less bus transactions are necessary in order to manage the cache coherence protocol.

[2] Hill [25] proposed the "three C's" (compulsory, capacity and conflict) in order to explain the different sources for cache misses on uniprocessor systems. On multiprocessor systems there is a new miss type, the coherence misses [2], so now, we have the "four C's".

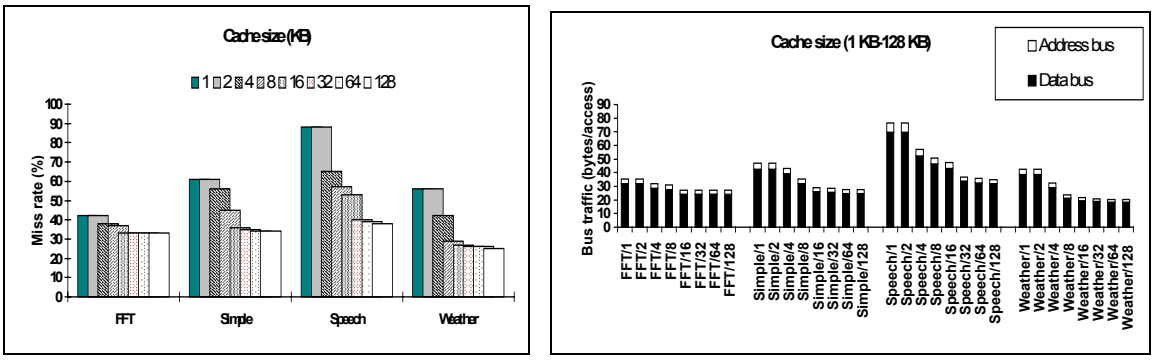

Fig. 5 - (a) Miss rate versus cache size. (b) Bus traffic versus cache size

Now we study the influence of the cache coherence protocol on the miss rate and the bus traffic. As before, the traffic is split into data traffic and address (including command) bus traffic. The same architecture is also considered (8 processors, 16-bit words, 64-byte blocks, four-way set associative caches and LRU replacement). We have gathered the results shown in Fig. 6(a) and Fig. 6(b). The miss rate for the MSI ([2], [17] and [18]) and MESI protocols is the same. This is consistent with the theory because the MESI protocol is only an improvement of the MSI protocol (it adds the "Exclusive" state) in order to reduce the number of bus transactions due to the coherence protocol [2]. In fact, Fig. 6(b) tell us that, although the MSI and MESI protocols have the same miss rate, the MESI protocol generates less bus traffic. Also, it may be observed that the Dragon protocol ([2], [6], [17] and [18]) has the least miss rate and bus traffic. This is possible because the Dragon protocol is a update-based protocol whereas the MSI and MESI protocols are invalidation-based protocols. In update-based protocols, whenever a shared location is written to by a processor, its value is updated in the caches of all other processors holding that memory block. Furthermore, in the Dragon protocol, updates are improved to a single-word write (the specific modified word) rather than a full cache block transfer. In contrast, with invalidation-based protocols, on a write operation the cache state of that memory block in all other processors' caches is set to invalid, so those processors will have to obtain the block through a miss (a coherence miss) and hence a larger bus traffic. Of course, it is easy to construct other scenarios in which the invalidation protocol does much better than the update protocol.

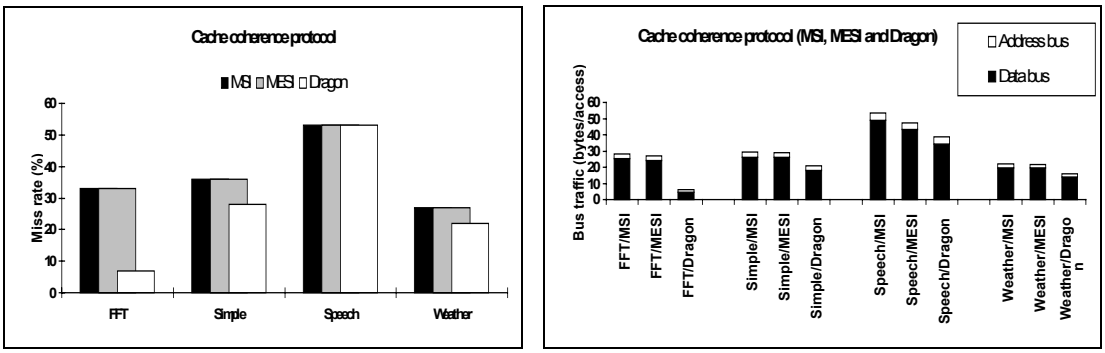

Fig. 6 - (a) Miss rate versus cache coherence protocol. (b) Bus traffic versus cache coherence protocol

Finally, we consider the three traces that were generated using the post-mortem scheme implemented by Mathews Cherian with Kimming So at IBM (*FFT*, *Simple* and *Weather* traces). We will study the influence of the number of processors on the miss rate and the bus traffic. The previous architecture is considered (MESI protocol, 16-bit words, 64-byte blocks, four-way set associative caches and LRU replacement). Fig. 7(a) and Fig. 7(b) show the found results.

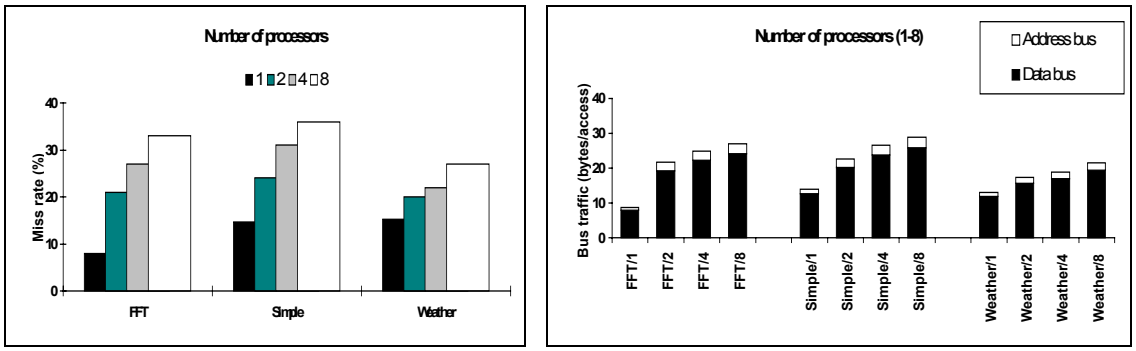

Fig. 7 - (a) Miss rate versus number of processors. (b) Bus traffic versus number of processors

We can conclude that, the greater number of processors for a parallel application, the more miss rate and bus traffic. It is possible because with a invalidation-based protocol, like the MESI protocol, the more processors are, the more possible is that several caches share the same block, and hence that, on a write operation, a cache force the other caches to invalidate that block, producing new misses (coherence misses) and increasing the number of block transfers. On the other hand, the greater number of processors, the greater number of bus transactions is needed to hold the cache coherence. In short, as the number of processors increases for a given problem size, the *working set* [26] starts to fit in the cache, and a domination by local misses (mainly, capacity misses) is replaced by a domination by coherence misses.

6. Notes for building the simulator

Students also must build a limited version of the simulator. For that, we give them a set of four notes. These notes explain, step by step, all operations and algorithms students must use in order to build a simulator similar to SMPCACHE. The notes are based some on others so that at the end students have the complete description of the simulator. The first note explains all the theoretical concepts about hierarchical memory systems that students must be based on to build the simulator, emphasizing the multiprocessor systems and their cache coherence problems. The rest of notes is dedicated to explain all the details of the application to build. A brief overview of these notes is given next.

In the first note, the needy basic theory is described. This theory can be complemented by other bibliographical sources that are given to the students. In this note the basic concepts about cache memories are remembered. The possible memory hierarchies on multiprocessors are explained. The cache coherence problem is described in detail. The existence of cache coherence schemes based on software and on hardware is shown. Within the hardware-based coherence schemes, it is differentiated between cache-coherent network architetures and cache coherence protocols. The cache coherence protocols are divided in directory schemes and snoopy cache protocols. A total of seven different snoopy cache protocols are exposed. Three of these seven procotols (MSI, MESI or Illinois, and Dragon) are described in detail (level of state transition).

In the second note, the environment that the simulator must possess is explained. The menus that the application must have are shown. The dialog boxes related with the administration of the configuration, the configuration files and the memory traces are described. The default values for the simulator configuration are given. The dialog boxes of other options of low difficulty, like those associated with the help system, the exit of the program, etc., are also explained.

In the third note, the software implementation of all the concepts of cache theory to take into account to develop the simulator is analysed in depth. The format of configuration files is explained. The format of memory trace files is also described. A possible pseudocode is shown for the software implementation of the cache coherence protocols, the mapping, the replacement policies and the schemes for bus arbitration.

In the fourth and last note, the data to show in screen during the different simulation types are explained detailedly. It is differentiated among a simulation observing the behaviour of the complete multiprocessor, a specific cache and an only memory block. In the simulations where one observes the behaviour of the complete multiprocessor or a specific cache, it is told the difference between a simulation studying all the memory blocks and only a particular block. It is differentiated among a simulation step by step, with breakpoint and complete execution. The statistical data that must be shown in screen are indicated. Some of these data are: number of memory accesses, number of cache hits and misses, as well as hit and miss rate, number of bus transactions, number of block transfers on the bus, number of state transitions, and division of these in different types (from a particular state to other), etc. The data to show by means of all kind of graphics are also indicated.

7. Conclusions and future work

In this paper we have shown the main characteristics of a cache memory system simulator on multiprocessor environments. The most immediate experiences have demonstrated us the simulator benefits with didactic goals. Students have used it as tool for experimenting the different theoretical aspects about cache memories and multiprocessors learned in the regular courses of Computer Architecture. Furthermore, students must build a restricted version of the simulator. For that, we give them a set of notes and they can use SMPCACHE as a reference for testing their own simulators. The construction of the simulator forces students to know in depth the theoretical considerations about cache memory systems, and particularly on multiprocessor environments (cache coherence, cache coherence protocols,...). In this way, we have observed that students acquire a better and larger knowledge about these aspects.

With research goals, professors have used the simulator because it allows to analyse different situations with an easy and modest tool. In short, at first, the simulator was conceived as a tool for applying it to teaching of cache memories. However, the potentiality of the developed system has proved its utility on program analysis and design strategies of memory systems on multiprocessors. The above characteristics enable the simulator to be used for designing systems that run optimally a determinate kind of programs and improve the operating mode of a determinate architecture.

In conclusion, the use of the simulator is an innovation in the practical classes that produces an improvement of the quality in education. Furthermore, it is a clear example of how the results of the developed research can revert in the teaching.

At present, we are working for adding new capabilities to the simulator, such as average access time, miss penalties, etc.

Finally, in order to contribute to a better knowledge of the cache problematic on multiprocessor systems, the simulator is available, with educational purposes, for universities and research centers, free of fee. People can contact with the authors.

8. References

[1] Smith, A. J.; “Bibliography and Readings on CPU Cache Memories and Related Topics”. Computer Architecture News, January 1986, 22-42

[2] Culler, D. E.; Singh, J. P.; Gupta, A.; “Parallel Computer Architecture. A Hardware/Software Approach”. Morgan Kauffmann, 1999

[3] Encore Computer Corp.; “Multimax Technical Summary”. 1986

[4] Lovett, T.; Thakkar, S.; “The Symmetry Multiprocessor System”. Proc. Int. Conf. Parallel Processing, Vol. I, August 1988, 303-310

[5] Monier, L.; Sindhu, P.; “The Architecture of the Dragon”. Proc. 30th IEEE Int. Conf., IEEE, February 1985, 118-121

[6] McCreight, E. M.; “The Dragon Computer System: An Early Overview”. Technical Report, Xerox Corporation, September 1984

[7] Thacker, C. P.; Stewart, L. C.; Satterthwaite Jr., E. H.; “Firefly: A Multiprocessor Workstation”. IEEE Transactions on Computers, Vol. 37, 8, August 1988, 909-920

[8] Baskett, F.; Jermoluk, T.; Solomon, D.; “The 4D-MP Graphics Superworkstation: Computing + Graphics = 40 MIPS + 40 MFLOPS and 100.000 Lighted Polygons per Second”. Proc. 33rd IEEE Computer Society Int. Conf. - COMPCOM 88, February 1988, 468-471

[9] Sima, D.; Fountain, T.; Kacsuk, P.; “Advanced Computer Architectures. A Design Space Approach”. Addison-Wesley, 1998

[10] Hill, M. D.; Smith, A. J.; “Evaluating Associativity in CPU Caches”. IEEE Transactions on Computers, Vol. 38, 12, December 1989, 1612-1630

[11] Hill, M. D.; Smith, A. J.; “Correction to ‘Evaluating Associativity in CPU Caches’ ”. IEEE Transactions on Computers, Vol. 40, 3, March 1991, 371

[12] Hill, M. D.; “DineroIII Documentation”. Unpublished Unix-style Man Page, Univ. of California, Berkeley (USA), October 1985

[13] Gómez, J. A.; Sánchez, J. M.; Vega, M. A.; “Simulación de Configuraciones de Memorias Caché Multinivel con Fines Didácticos”. IV Jornadas de Informática, Las Palmas de Gran Canaria (Spain), July 1998, 117-126

[14] Hennesy, J. L.; Patterson, D. A.; “Computer Architecture. A Quantitative Approach”. Morgan Kauffmann, 2nd edition, 1996

[15] Hwang, K.; Briggs, F. A.; “Computer Architecture and Parallel Processing”. McGraw-Hill, 1984

[16] Patterson, D. A.; Hennesy, J. L.; “Computer Organization and Design. The Hardware/Software Interface”. Morgan Kauffmann, 1994

[17] Archibald, J.; Baer, J.-L.; “Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model”. ACM Transactions on Computer Systems, Vol. 4, 4, November 1986, 273-298

[18] Eggers, S.; “Simulation Analysis of Data Sharing in Shared Memory Multiprocessors”. Ph. D. Thesis, Univ. of California, Berkeley (USA), Computer Science Division, Technical Report UCB/CSD 89/501, April 1989

[19] Johnson, E. E.; Ha, J.; “PDATS: Lossless Address Trace Compression for Reducing File Size and Access Time”. Proc. IEEE International Phoenix Conference on Computers and Communications, April 1994, 213-219

[20] Smith, A. J.; “Line (Block) Size Choice for CPU Cache Memories”. IEEE Transactions on Computers, Vol. 36, 9, September 1987, 1063-1075

[21] Smith, A. J.; “Correction to ‘Line (Block) Size Choice for CPU Cache Memories’ ”. IEEE Transactions on Computers, Vol. 38, 6, June 1989, 927

[22] Obaidat, M. S.; Khalid, H.; Sadiq, K.; “A Methodology for Evaluating the Performance of CISC Computer Systems under Simple and Two Cache Environments”. Microprocessor and Microprogramming Journal, July 1994, 411-421

[23] Darema, F.; Karp, A.; Teller, P.; “Applications Survey Reports-I”. IBM RC 12743, May 1987

[24] Papamarcos, M.; Patel, J.; “A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories”. Proc. 11th Annual Symposium on Computer Architecture, June 1984, 348-354

[25] Hill, M. D.; “Aspects of Cache Memory and Instruction Buffer Performance”. Ph. D. Thesis, Univ. of California, Berkeley (USA), Computer Science Division, Technical Report UCB/CSD 87/381, November 1987

[26] Denning, P. J.; “The Working Set Model for Program Behavior”. Communications of the ACM, Vol. 11, 5, 1968, 323-333