A VERSATILE SIMULATOR FOR CACHE MEMORIES ON DSM SYSTEMS

Miguel A. Vega-Rodríguez, R. Jorge Gil-Ramos, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez
Univ. Extremadura. Dept. Informatica
Escuela Politecnica. Campus Universitario s/n. 10071 Caceres. SPAIN
E-mail: mavega@unex.es

ABSTRACT

In this work we present a simulator for the analysis of cache memories on scalable systems with distributed shared memory (DSM). In this kind of systems, the cache coherence is usually provided by means of directory-based protocols. The simulator, called DSMCache, has a full graphic and user-friendly interface, and it operates on PC systems with Windows. We think this tool is useful for the analysis of programs and design strategies of memory systems on DSM architectures. Thus, the simulator could be used in order to understand the design of organizations that run optimally a determinate type of parallel programs or to improve the operating mode of a concrete parallel architecture. Furthermore, due to its interface, it is an interesting tool for the teaching of cache memories on DSM systems.

1. INTRODUCTION

Caches are a critical component in the performance of any computer system (part of the existent bibliography about caches can be found in (Smith 1986)).

A very important and promising part of the evolution of parallel architectures is the inclusion of cache coherence in a scalable machine with distributed shared memory (Culler et al. 1999). Most of these systems use directory-based coherence protocols and a scalable interconnection network. In this way, it is possible to build massively parallel processing systems (MPPs). In fact, DSM architectures are suitable for both commercial and scientific applications. Especially, cache coherent non-uniform memory access architectures (CC-NUMA), which can support both easy programming (due to their shared address space) and scalability (with a great number of processors) (Chung et al. 2001).

Trace-driven simulation is often a cost-effective method to estimate the performance of computer system designs. Above all when designing caches, translation-lookaside buffers (TLBs), 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.

In this paper we present a trace-driven simulator for cache memories on scalable systems with distributed shared memory. There are well-known cache memory simulators (see (Uhlig and Mudge 1997) for a detailed study about trace-driven simulators): TYCHO (Gee et al. 1993; Hill and Smith 1989), DINERO (Edler and Hill 2005; Hill 1985), ACS (Acme Cache Simulator) (Hunt 1997), SISMEC (Gómez et al. 1996; Gómez 2005), CVT (Deijl et al. 1997), bigDIRN (Agarwal 2004), SMPCache (Vega et al. 2001; Vega 2005),... Using these simulators as a point of departure we have developed a new simulator, where new considerations have been taken into account regarding its capacity for working on multiprocessor environments, its analysis potentiality (variability in the simulation process, obtained data and their format, etc.), its interface and portability. The design considerations have been oriented to satisfy a set of requirements with didactic and research goals.

In the following section we mention the different hardware architectures supported by the simulator. In section 3 we explain both its interface and its functionality. Section 4 details the memory trace formats that can be used with the simulator. Then, in section 5, we display the practical results that the simulator has obtained, and finally, the conclusions are presented in the last section. The theoretical considerations concerning cache memory systems, and particularly regarding their use in multiprocessor environments, are widely discussed in many computer architecture texts (Culler et al 1999; Sima et al. 1998; Hennessy and Patterson 2003; Hwang 1993; Stallings 2003), and we will not mention them here. All operations and algorithms we use are similar to those found in these computer architecture texts. As a consequence, the results obtained with the simulator are very close to the real world.

2. SUPPORTED HARDWARE ARCHITECTURES

As for the hardware architecture, the simulator offers the possibilities summarized in table 1. Furthermore, it has been designed so that it can be easily extended (for example, adding a new coherence protocol).

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

Table 1: Architectural characteristics supported by the simulator

Number of processors	1, 2, 4, 8, 16, 32, 64 or 128
Directory protocols	SGI Origin (Sequent NUMA-Q in project)
Snoopy protocols	MSI or MESI (for used directory protocols)
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 coherence protocols)
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

All these configuration parameters are related among themselves according to the theoretical models. If the user makes a choice which contradicts other parameters, the simulator warns him/her, and blocks the choice.

The user selects different choices in the simulator (by interactive windows) to configure a given architecture that may be stored on an ASCII data file for future loading, so the need to make many selections for configuring the same memory model is avoided. What is more, it is possible to set a default initial configuration for the simulator. These characteristics allow us to build a database with different memory structures, emulating architectures like Silicon Graphics, Sequent Computer Systems, and others.

3. SIMULATOR INTERFACE AND FUNCTIONALITY

DSMCache is a software tool for the evaluation of hierarchical memory systems on scalable architectures with distributed shared memory. This simulator operates on PC systems with Windows, and it has been written in a visual language. The simulator offers a Windows typical graphic interface, having a very complete contextual help system. Figure 1 shows an overall view of its graphic interface. This interface can be modified by the user: with or without XP style, with or without menu shadows, with or without certain bars and panels, etc.

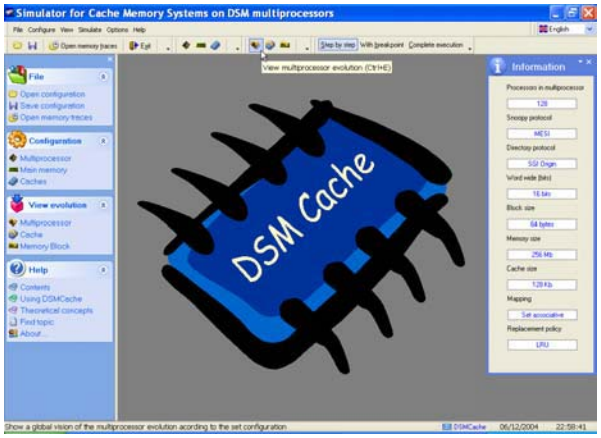

Figure 1: Graphic interface of the simulator

The simulator allows us to select the different choices for configuring a given architecture, and it shows us how the system responds to the memory accesses that the programs generate (memory traces used for the different processors during the simulation). Therefore, it is an application that could be used in order to evaluate memory systems on DSM architectures with research goals.

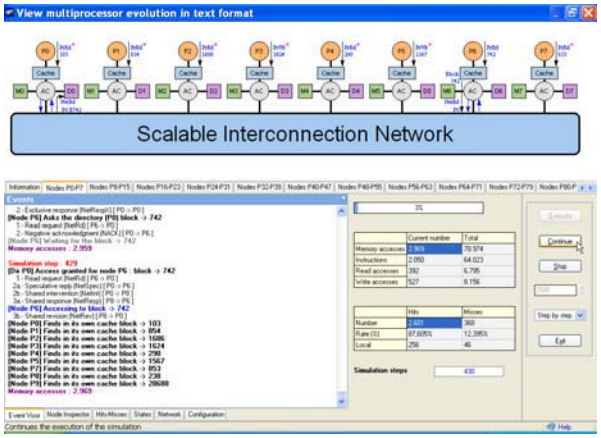

Figure 2: Multiprocessor vision during a simulation. Note that, in this case, at the top we can see the tab with the nodes P0-P7

Because of its easy and user-friendly interface, the simulator can also be used for teaching purposes; since it allows us to observe, in a clear and graphic way, how the complete system evolves as the execution of the programs goes forward (the memory traces are read). With the simulator, it is possible to obtain a global vision of the multiprocessor evolution, a vision of the evolution of a particular cache, or even, a vision of the evolution of a specific memory block. It always displays the memory accesses that every processor demands, the state of the interconnection network (network messages or transactions), of every communication assist, of every cache, of every directory, of every memory block within every cache, etc. Figure 2 shows the appearance of the screen in a simulation. At the top we can see a graphic representation of the multiprocessor: After a tab with generic information, the rest of tabs present the multiprocessor, 8 by 8 processors (P0-P7, P8-P15,...).

At the bottom there are also several tabs. The first one includes the event visor (see figure 2), in which the most important events are detailed after each simulation step. Another tab of great interest is the node inspector (figure 3), in which we can inspect (at any simulation step) the state of each node, including its processor, cache, directory, communication assist, and local main memory.

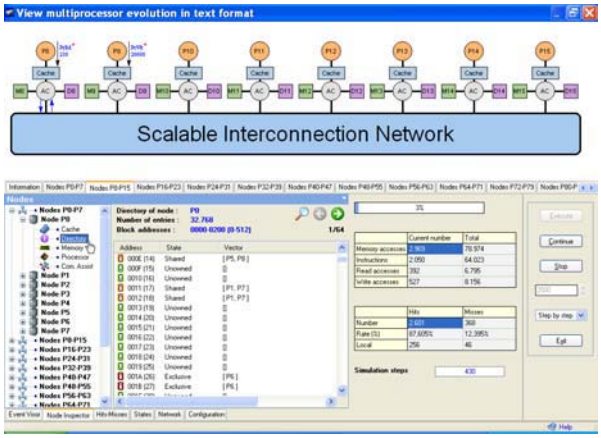

Figure 3: The node inspector during a simulation. Note that, now, at the top we can see the tab with the nodes P8-P15

The simulator also allows us to study the most suitable memory system for our needs before actually implementing it, or it simply helps us to simulate real systems in order to see their efficiency and compare results in an easy way. Some of the parameters we can study with the simulator are: program locality; influence of the number of processors, directory protocols, snoopy protocols, mapping, replacement policies, cache size (blocks in cache), number of cache sets (for set-associative caches), number of words by block (block size), word wide,...

Furthermore, DSMCache presents, using statistical data (in figures 2 and 3, tables in the bottom right hand corner, besides Hits-Misses, States and Network tabs at the bottom) and several kinds of graphics (see figure 4), interesting measurements like:

- Global number of network transactions/messages, and for types: read requests, read-exclusive requests, upgrade requests, invalidation requests, shared responses, exclusive responses, speculative replies, shared interventions, exclusive interventions, revisions, acknowledgments, NACKs, etc.

- Number of remote network messages, and number of local messages (inside the node).

- Total number of block transfers, and also distinguishing between block transfers through the interconnection network (remote block transfers) and inside the node (local block transfers).

- Network traffic taking into account the previous measurements.

- Total number of replacements, and number of replacements of local (memory blocks that are allocated in the local main memory) and remote blocks.

- Total number of write-backs, and number of write-backs by each node.

- Number of state transitions in cache (each block in a cache has a state associated with it) and directory (each memory block also has a directory state).

- Number of state transitions (in cache or directory) from a particular state to other.

- Global number of memory accesses, and for types: instruction captures, data readings and data writings.

- Number of remote accesses (to a remote block) and local accesses (to a local block) using the network.

- Number of global and local cache hits and misses, as well as the hit and miss rate.

- Distribution of the hits and misses (and their rates) for readings and writings: distinguishing between remote and local data.

- Number and rate of misses satisfied by the local main memory, by a remote cache or by a remote main memory.

- Number of simulation steps.

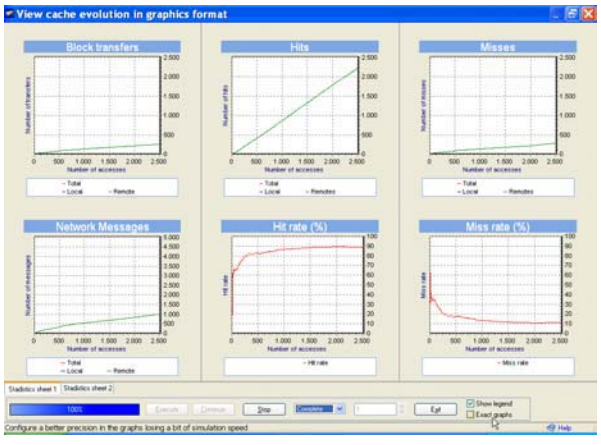

Figure 4: Data for a specific cache during a simulation in graphic format

All these data are shown at very different vision levels, although the relationship among all the system elements is taken into account in all cases. We can consequently carry out a simulation observing the

complete multiprocessor, and all the memory blocks (figure 2) or only one particular block. We can also observe a specific cache, and all the memory blocks or only a concrete block (figure 5).

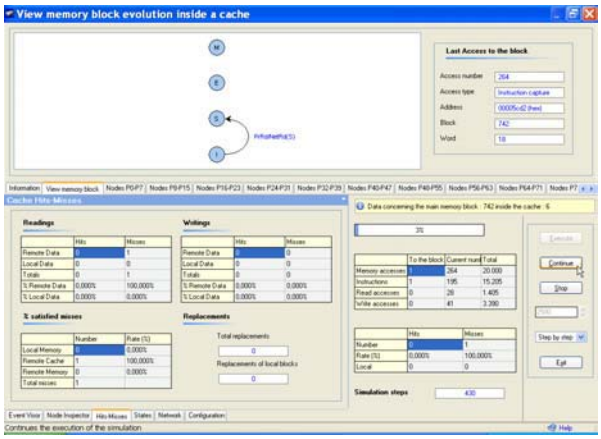

Figure 5: State transition diagram for a cache block

The simulation consists in the programmed reproduction of operations that would actually be performed by the components of cache memory system on a real multiprocessor. Suitable computations are performed to achieve that goal, and the present and accumulated results are shown at the same time. The simulation can be carried out as a whole (complete execution) or, as is usually much more interesting, step by step, in order to observe the internal operation of the system. For very long traces, breakpoints can also be inserted. There are therefore three kinds of simulation (step by step, with breakpoint and complete execution), and it is possible to change from one to another without waiting for the end of the simulation. It is also possible to abort the simulation at any time, in order to correct any architectural detail.

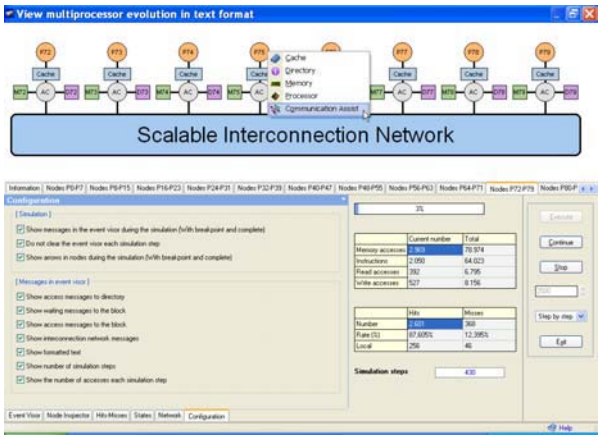

Figure 6: Contextual menu to obtain data of the different elements in the system, and Configuration tab for configuring the information and messages shown in a simulation. Note that, at the top, we can see the tab with the nodes P72-P79

Another aspect to highlight is the possibility of changing the information and messages shown in a simulation using the Configuration tab (see figure 6). Figure 6 also includes an example of use of the contextual menus to obtain information about a particular system element (cache, directory, communication assist,... of certain node).

Finally, it is important to point out that, at present, the user can select between two languages for the simulator: English or Spanish.

4. ADMITTED MEMORY TRACE FORMATS

In order to check and use the simulator we have a set of memory traces. Many of these traces come from tests performed with benchmarks and with application programs on different real (multiprocessor and uniprocessor) architectures. Some have been obtained by anonymous ftp from different trace databases (for example, from the PARL (PARL 2004)). Others have been created and edited using a common text editor. In this way, we can create, for example, memory trace files for teaching purposes (with some special feature). Finally, we have also generated some trace files for basic operations with arrays by means of C++ programs.

These traces have different formats. In particular, the DSMCache simulator, until now, admits the following memory trace formats: the trace format of SMPCache (Vega 2005), the trace format of LIMES (Ikodinovic et al. 1999), the canonical format for multiprocessor traces developed by Anant Agarwal (Agarwal 2004), besides the new trace format created for DSMCache.

Figure 7 presents an example in which a memory trace with DSMCache format (files with "*.spr" extension) is being loaded. In this window, we can also observe several trace files with SMPCache format ("*.prg" extension), which could also be loaded in the simulator.

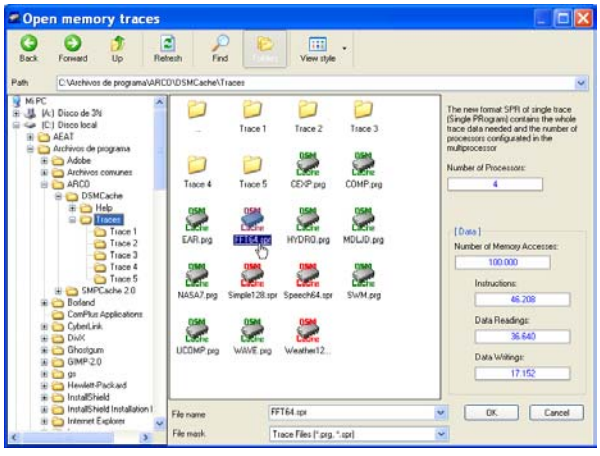

Figure 7: Window for loading memory traces in the simulator. It is possible to load memory traces in both SMPCache and DSMCache format

Figure 8 shows an example of the new trace format for DSMCache. The first comments indicate the number

of processors for which the trace was created, as well as the quantity of accesses that contains, and the distribution for types (instruction captures, data readings or data writings) and for processors. These data are not obligatory, although they speed up the initial management of the trace by the simulator.

```
# Number of Processors
4
# Number of Memory Accesses
11
# Number of Instructions
4
# Number of Data Readings
5
# Number of Data Writings
2
# Accesses by Processor
P0 Acce=3 Inst=1 Read=1 Writ=1
P1 Acce=3 Inst=1 Read=2 Writ=0
P2 Acce=3 Inst=1 Read=1 Writ=1
P3 Acce=2 Inst=1 Read=1 Writ=0
# Trace Data
T=0
P0=0 d80d0200
P1=0 d80d0208
P2=0 d80d0210
P3=0 d80d0218
T=4
P0=2 980e0200
P1=2 e80d0200
P2=2 242a0400
P3=2 b6150200
T=10
P0=3 980e0210
P1=2 e80d0220
P2=3 242a0420
```

Figure 8: DSMCache trace format

After that, the trace lists the accesses grouped by time/simulation steps (T=*nnn*). Each time step usually includes one access by processor, though it is not required that all the processors have an associate access. Each line means one memory access, using the following format:

Pi=Type Address

Where:

- *Pi* indicates the processor that does the access, where *i* ranges from 0 to the configured number of processors (minus 1, because *i* begins from 0).
- *Type* is a decimal number that identifies the memory access operation type demanded by that processor in a given time: to capture an instruction (0), to read a memory data (2), or to write a data in memory (3).
- *Address* is a hexadecimal number that indicates the effective address of the memory word to be accessed. This address will be translated by the simulator for locating the word in the memory system block structure.

As it can be observed, we have looked for an easily intelligible format, which allows us to modify the trace using a common text editor. In conclusion, in the example of figure 8, this memory trace presents 4 instruction captures of a certain parallel program. Five accesses imply data reading, and two require writing in memory. In total, those 4 instructions imply 11 memory accesses.

As for the LIMES trace format and the canonical format for multiprocessor traces proposed by Anant Agarwal, we have also developed a converter from these formats to both the DSMCache and SMPCache format. Figure 9 shows the interface of this trace format converter.

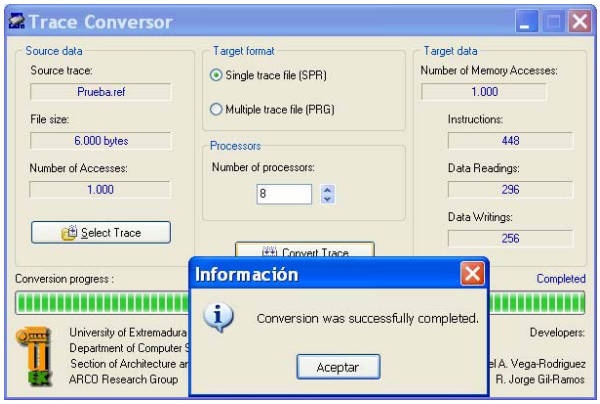

Figure 9: Graphic interface of the trace format converter

5. PRACTICAL RESULTS

In this section we show some experiments and the associated conclusions for a set of specific architectures with a set of concrete traces. We will thus illustrate the use of the simulator and some of the possible practical experiments to be carried out. A wide set of experiments has been carried out, studying parameters of interest like the locality of different programs, the influence of the cache size, the mapping, the replacement policies, the block size,... on the miss rate, network traffic, etc. For reasons of space, we will only show some of these experiments.

Table 2: Multiprocessor traces used

Name	References	Language	Comments
FFT	7,451,717	Fortran	Parallel application that simulates the fluid dynamics with FFT
Speech	11,771,664	---	Kirk Johnson and David Kranz (both at MIT) are responsible for this trace
Simple	27,030,092	Fortran	Parallel version of the SIMPLE application
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.

In these experiments we have studied traces with tens of millions of memory accesses (references) for four benchmarks (*FFT*, *Simple*, *Speech* and *Weather*). These traces were provided by David Chaiken (then of MIT) for PARL (PARL 2004). The traces represent several real parallel applications. A summary of the traces is shown in table 2. *FFT*, *Simple* and *Weather* traces were generated using the post-mortem scheme implemented by Mathews Cherian with Kimming So at IBM.

We are first going to analyse an architecture with eight processors, SGI-Origin directory protocol, MESI (or Illinois, for a detailed description see (Culler et al. 1999)) snoopy protocol, 16-bit words, 64-byte blocks, four-way set associative caches and LRU replacement. On figure 10 the miss rate versus cache size is represented. From this figure we can obtain the conclusion that the global miss rate for the system decreases as the caches size increases, because capacity and conflict misses are reduced. For large cache sizes the miss rate is stabilized, this shows us the compulsory and coherence misses, which are independent of the cache size. Current measurements demonstrate that the shared data has less spatial and temporal locality than other data types. In other words, in general, parallel programs exhibit less spatial and temporal locality than serial programs. It is thus usual for the miss rates to be higher for multiprocessor traces than for uniprocessor traces. Figure 11 shows the network traffic on the system per memory access for this same experiment. The conclusions we obtain are similar to the previous ones for the miss rate. The network traffic is reduced as the miss rate decreases because of two fundamental reasons.

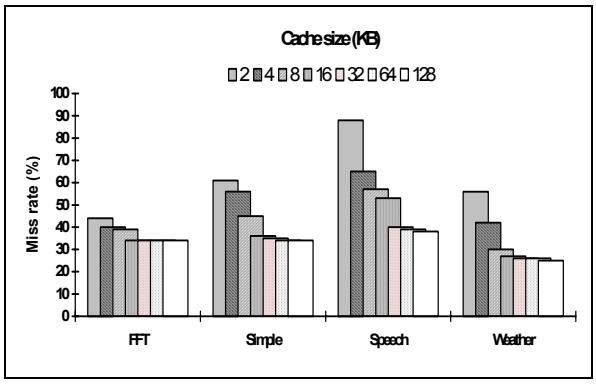

Figure 10: Miss rate versus cache size

On the one hand, there are less data transfers from the distributed shared main memory to the caches. On the other hand, due to there being less misses, less network transactions are necessary in order to manage the cache coherence protocols.

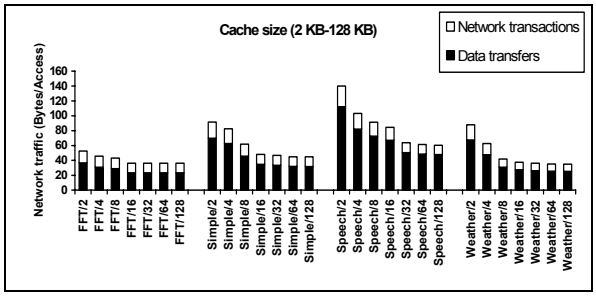

Figure 11: Network traffic versus cache size. The traffic is split into data transfers and network transactions

We are now going to discuss the three traces that were generated using the post-mortem scheme (*FFT*, *Simple* and *Weather* traces). We will study the influence of the number of processors on the miss rate, the network traffic, the execution time, and the speedup, bearing in mind the previously mentioned architecture (SGI-Origin directory protocol, MESI snoopy protocol, 16-bit words, 64-byte blocks, four-way set associative caches and LRU replacement). Figures 12 and 13 present the results for the miss rate and the network traffic.

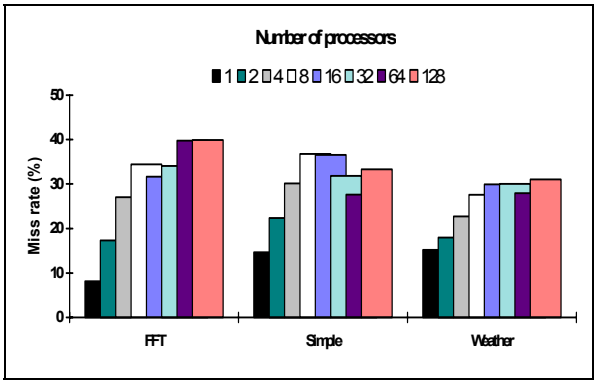

Figure 12: Miss rate versus number of processors

We can conclude that, in general, the greater the number of processors for a parallel application, the higher the miss rate and network traffic. This is possible because with a invalidation-based protocol, like the MESI protocol, the more processors there are, the more possible it is that several caches will share the same block, and hence that in a writing operation, a cache forces the other caches to invalidate that block, producing new misses (coherence misses) and increasing the number of block transfers.

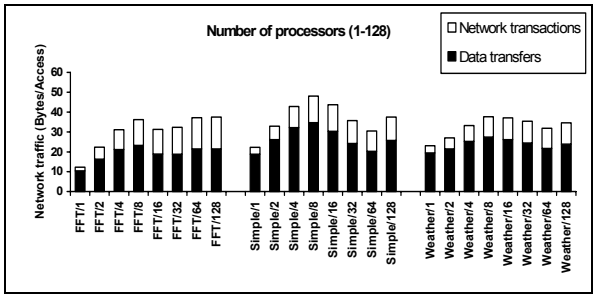

Figure 13: Network traffic versus number of processors

On the other hand, the greater the number of processors, the greater the number of network transactions that are needed to hold the cache coherence. In short, as the number of processors increases for a given problem size, the *working set* (Denning 1968) starts to fit in the cache, and local misses (mainly, capacity misses) are replaced by coherence misses.

In this line, figure 14 presents the network traffic, splitting into local and remote traffic. We can see clearly how the local traffic (traffic inside the node) is replaced by remote traffic (traffic using the interconnection network) when the number of processors grows.

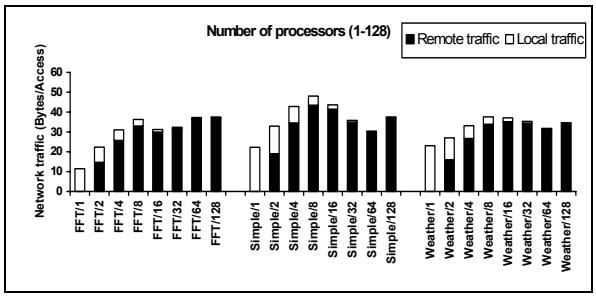

Figure 14: Network traffic versus number of processors. The traffic is split into remote and local traffic

Figures 15 and 16 display the results for the execution time and the speedup.

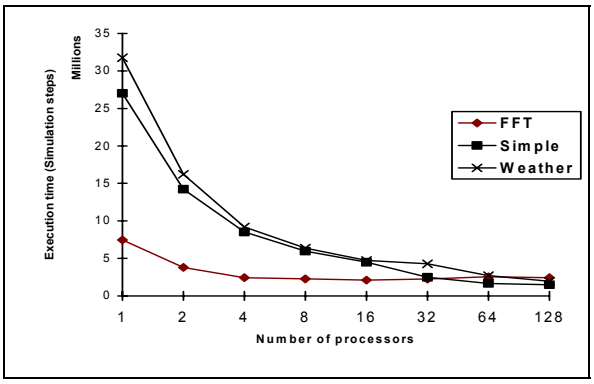

Figure 15: Execution time versus number of processors

In this case, the theoretical predictions are also fulfilled, which indicate a decrease of the execution time, or an increase of the speedup, as the number of processors is scaled. Anyway, this performance increment in the execution of the parallel programs also depends on the programs, and not only of the number of used processors (double processors does not indicate double speedup). Furthermore, in order to obtain a better performance we should have improved the balance of work done by each processor, as the number of processors was increased.

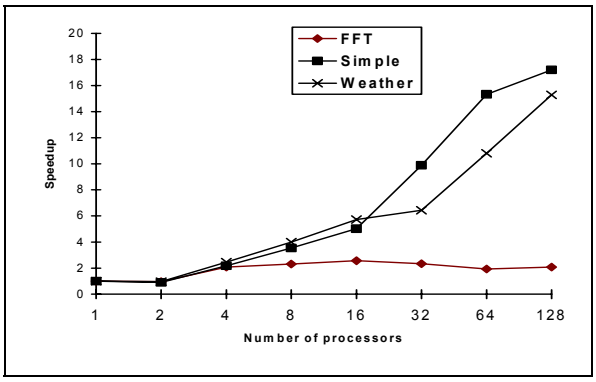

Figure 16: Speedup versus number of processors

6. CONCLUSIONS

In this paper we have shown the main features of a cache memory system simulator on scalable architectures with distributed shared memory. We think the simulator has many advantages from an educational point of view. Students could use it as a tool for experimenting the different theoretical aspects about cache memories and DSM systems in the regular courses of Computer Architecture. In this way, students would acquire better and larger knowledge about these subjects.

With research goals, we think it is an attractive and easy tool in order to study memory models on DSM systems that have a better performance for certain parallel programs.

At present, we are extending the simulator with all the SMPCache functionalities (Vega 2005). In this way, we will have a simulator for cache memory systems on any kind of multiprocessor: scalable systems (to many processing nodes) with distributed shared memory (usually based on directory coherence protocols and a scalable interconnection network), and symmetric multiprocessors (small-to-moderate scale multiprocessors, usually based on a centralized shared memory, a shared bus and snoopy coherence protocols).

7. REFERENCES

Agarwal, A. 2004. *bigDIRN simulator*. Available at ftp://tracebase.nmsu.edu/pub/tracebase4/MP/mit/scheduled/src/dirsim/.

Chung, Y.; H. Kim; J.-W. Park and K. Lee. 2001. "Performance Evaluation for CC-NUMA Multiprocessors Using and OLTP Workload". *Microprocessors and Microsystems*, Elsevier, vol. 25, no. 4 (Jun), pp. 221-229.

Culler, D.E.; J.P. Singh and A. Gupta. 1999. *Parallel Computer Architecture. A Hardware/Software Approach*. Morgan Kaufmann.

Deijl, E.; G. Kanbier; O. Temam and E.D. Granston. 1997. "A Cache Visualization Tool". *Computer*, IEEE, vol. 30, no. 7 (Jul), pp. 71-78.

Denning, P.J. 1968. "The Working Set Model for Program Behavior", *Communications of the ACM*. ACM, vol. 11, no. 5 (May), pp. 323-333.

Edler, J. and M.D. Hill. 2005. *Dinero IV Trace-Driven Uniprocessor Cache Simulator*. Available at http://www.cs.wisc.edu/~markhill/DineroIV/.

Gee, J.D.; M.D. Hill; D.N. Pnevmatikatos and A.J. Smith. 1993. "Cache Performance of the SPEC Benchmark Suite". *IEEE Micro*, IEEE, vol. 3, no. 2 (Aug), pp. 17-27.

Gómez, J.A.; J.M. Sánchez and J.A. Moreno. 1996. "An Educational Tool for Testing Hierarchical Multilevel Caches". *ACM Computer Architecture News*, ACM, vol. 24, no. 4 (Sep), pp. 11-15.

Gómez, J.A. 2005. *SISMEC Simulator*. Available at http://arco.unex.es/jangomez/investigacion/proyectos/sismec/sismec.html.

Hennessy, J.L. and D.A. Patterson. 2003. *Computer Architecture. A Quantitative Approach*. 3rd edition, Morgan Kaufmann.

Hill, M.D. 1985. *DineroIII Documentation*. Unpublished Unix-style man page, Univ. of California, Berkeley (Oct).

Hill, M.D. and A.J. Smith. 1989. "Evaluating Associativity in CPU Caches". *IEEE Transactions on Computers*, IEEE, vol. 38, no. 12 (Dec), pp. 1612-1630.

Hunt, B.R. 1997. *Acme Cache Simulator*. Available at ftp://tracebase.nmsu.edu/pub/tracebase4/src/acs/ (Jun).

Hwang, K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill.

Ikodinovic, I.; D. Magdic; A. Milenkovic and V. Milutinovic. 1999. "Limes: A Multiprocessor Simulation Environment for PC Platforms". *3rd International Conference on Parallel Processing and Applied Mathematics (PPAM)*, Poland (Sep), pp. 398-412.

PARL (Performance and Architecture Research Lab), New Mexico State University (NMSU). 2004. *The NMSU TraceBase*. Available by anonymous ftp at ftp://tracebase.nmsu.edu.

Sima, D.; T. Fountain and P. Kacsuk. 1998. *Advanced Computer Architectures. A Design Space Approach*. Addison-Wesley.

Smith, A.J. 1986. "Bibliography and Readings on CPU Cache Memories and Related Topics". *ACM Computer Architecture News*, ACM, vol. 14, no. 1 (Jan), pp. 22-42.

Stallings, W. 2003. *Computer Organization & Architecture: Designing for Performance*. 6th edition, Prentice-Hall.

Uhlig, R.A. and T.N. Mudge. 1997. "Trace-Driven Memory Simulation: A Survey". *ACM Computing Surveys*, ACM, vol. 29, no. 2 (Jun), pp. 128-170.

Vega, M.A.; J.M. Sánchez and J.A. Gómez. 2001. "An Educational Tool for Testing Caches on Symmetric Multiprocessors". *Microprocessors and Microsystems*, Elsevier, vol. 25, no. 4 (Jun), pp. 187-194.

Vega, M.A. 2005. *SMPCache simulator*. Available at http://arco.unex.es/smpcache.

AUTHOR BIOGRAPHIES

MIGUEL A. VEGA-RODRIGUEZ is professor of Computer Architecture in the Department of Computer Science, University of Extremadura, Spain. He received a PhD degree in Computer Science from the University of Extremadura. Dr. Vega-Rodriguez's main research interests are cache memory systems on multiprocessors (including their simulation), and applications of reconfigurable computing (using FPGAs) to image processing and cryptography.

R. JORGE GIL-RAMOS received a MS degree in Computer Science from the University of Extremadura, Spain, in 2004. His main research interest is simulation of cache memory systems on multiprocessors.

JUAN A. GOMEZ-PULIDO is professor of Computer Architecture in the Department of Computer Science, University of Extremadura, Spain. He received a PhD degree in Computer Science from the Complutense University of Madrid in 1993. His main research interests are performance evaluation of the multilevel cache memory systems and applications of reconfigurable hardware to different fields of signal processing.

JUAN M. SANCHEZ-PEREZ is Professor of Computer Architecture in the Department of Computer Science, University of Extremadura, Spain. He received a PhD degree in Physics from the Complutense University of Madrid in 1976. His research interests are modern computer architectures, applications of reconfigurable hardware and logic design.