AN INTERACTIVE ANIMATION FOR LEARNING HOW CACHE COHERENCE PROTOCOLS WORK

Alberto Alcón Laguéns, Sergio Barrachina Mir, Enrique S. Quintana Ortí

Universidad Jaume I (SPAIN)
albertoalcon@gmail.com, {barrachi,quintana}@icc.uji.es

Abstract

One of the problems a multiprocessor has to deal with is cache coherence. To put it into simple words, the memory system is coherent when each piece of data is up-to-date in all caches as well as in the main memory of the multiprocessor.

A variety of protocols (algorithms) can be used to maintain memory coherence. To introduce how these protocols work to our students, we looked for simulation tools or animations that show changes in the cache blocks, transitions between states, messages that were generated, etc. Although we found some multiprocessor simulators, they were far too complex as they really were not meant to illustrate how the cache coherence protocols worked. On the other hand, unfortunately, the animations we found did not address all the details that should be covered in our computer architecture course.

To solve this problem, we have developed a flash interactive animation that shows how some of the most frequently used cache coherence protocols work: MSI and MESI snoopy protocols and the MESI directory protocol. For each protocol, a sequence of read and write operations illustrates all possible situations that can take place in each protocol. The tool is interactive in that the student can go forward and backward to understand/study the different actions that occur at each step.

Keywords: Computer architecture, multiprocessors, cache coherence, interactive animation.

1 INTRODUCTION

Most current multiprocessors contain multiple cache memories to hide the gap between the processor and the memory speeds. In these systems, data can be stored/replicated in multiple places. This introduces a serious problem as, without careful precautions, different processors could end up seeing different values for the same piece of data. To avoid this, an scheme (a protocol or algorithm) is needed to enforce *coherence* in the memory system.

Snooping and directory-based protocols to tackle cache coherence in multiprocessors are taught in advanced courses of computer architecture. Although the algorithms are relatively simple, there are a lot of different situations that have to be considered in order to understand how these protocols deal with all the details of the problem. We believe that learning these particular solutions can be eased with a graphical illustration of how the various situations are handled by the different protocols.

While teaching the cache coherence protocols during our courses in advanced computer architecture, we searched for simulation tools or animations that could illustrate how these protocols worked. We wanted that our students could experience how the states in each cache block change when the different processors access different memory positions, which are the transitions between states, and which messages are generated in each case. Although we found some sophisticated simulators, they were way too complex as they really were not designed to serve as tools to aid in teaching. We also found several interactive, simple tools that captured simple cache coherence situations but, unfortunately, these were very simplistic and no exhaustive at all in covering the possible situations.

Given these premises, we decided to develop an interactive animation that could be used in a classroom to show the students how the cache coherence protocols worked, allowing them to gain more experience in these protocols. The flash interactive animation that we have developed covers

ISBN:978-84-614-7423-3

the MSI and MESI snoopy protocols as well as the MESI directory-based protocol. The situations that are captured in these animations are comprehensive, showing all possible cases that can occur in each protocol. A full description of what is going on is included with each example, and the student can go forward and backward to study how each step is performed. Furthermore, the tool is intuitive to use and requires no previous preparation.

The rest of the article is organized as follows. In Section 2, we offer a brief survey of the cache coherence problem. Section 3 describes the developed interactive tool. Finally, in Section 4, conclusions and future work are summarized.

2 THE CACHE COHERENCE PROBLEM

Multiprocessors [1], specially those constructed of relatively low-cost microprocessors, offer a cost-effective solution to the continually increasing need for more computing power. However, designing and programming multiprocessors systems correctly and efficiently pose complex problems, with one of them being memory coherence.

The cache coherence problem arises when different processors cache and update values of the same memory location [2]. A clever solution, which builds on the bus interconnect, addresses the cache coherence in small-scale shared memory multiprocessors. The basic idea is to guarantee that, before a memory location is written, all other copies of the location, which may be present in other caches, are invalidated. Thus, the system permits multiple read copies of a memory location to exist, but only one write copy.

The bus that interconnects the processors and the distributed memory modules is the key to the common snoopy coherence protocols. When a processor wants to write into a cache block that may be shared, a snoopy protocol places the request on the bus, so that all caches that have a copy of the cache block, upon seeing the request, simply invalidate the copy.

In summary, the snooping operations are implemented by placing the request on the bus and having all the caches read the address and (if the address matches an address in the processor's cache) either perform an invalidation or supply data from the cache. Because all requests must be placed on the bus, which accommodates only one request at a time, the bus serializes concurrent writes from two or more processors. This imposes an ordering on all writes (including those to the same address) and is critical to maintaining coherence.

The reduced programming complexity offered by cache coherence, together with its relatively cheap implementation, led to its adoption in all small-scale, bus-based multiprocessors. In the last few years, microprocessors have been designed with a small number of cores (two to eight) within the processor die, and cache coherence protocols, further reducing the cost of small-scale multiprocessors and significantly increasing their popularity.

Unfortunately, the snoopy schemes used in small-scale symmetric multiprocessors do not scale, as the bus rapidly becomes a bottleneck when the number of cores connected to it grows.

The solution to this problem was to develop a coherence mechanism that could be efficiently extended to other interconnects/architectural designs, and directory-based designs were the answer. These schemes rely on an extra structure, called the directory, which tracks which processors have cached any given block in main memory. Because the directory maintains up-to-date information of which caches have copies of any given memory block, a coherence protocol can use it to attain a consistent view of memory. To guarantee coherence, the state of each cache block is tracked in the cache and additional information is kept in the directory for each block.

The directory protocols were never widely deployed because, in a small-scale machine, a bus suffices and the snoopy schemes are easy and cheap to implement [2]. While a directory-based approach avoids the use of broadcast to interrogate all the caches, a single centralized directory would simply move the bottleneck from the bus to the directory.

The snoopy and directory-based cache coherence protocols can use three or four different states to

track the state of a cache block [3]. In the MSI cache coherence protocol, each cache block can be in one of the three following states: Modified, Shared or Invalid. The cache block is marked as modified when its data has been modified, this also implies that it has the only valid copy of that piece of data. The cache block is marked as shared when the data has been read, being in this state implies that there are other valid copies of the data (at least in main memory). Finally, the cache block is marked as Invalid when some other processor has requested a write operation over it.

In the MESI cache coherence, one more state, Exclusive, is included. This state indicates that the cache has an unmodified valid copy of the data and that there are no other copies of this piece of data in the rest of caches. Thanks to this state, when a write request is performed over a cache block that is only in that cache (in exclusive state), the write request is not placed on the bus. This alleviates the usage of the bus specially when executing programs were the amount of actually shared data is low.

The purpose of the proposed interactive animation is to facilitate the understanding of how the MSI and MESI cache coherence snoopy and directory-based protocols work.

3 THE CACHE COHERENCE PROTOCOL INTERACTIVE ANIMATION

To develop the animation we have used Adobe Flash. This tool provides two ways to develop animations. The first one consists in defining a *time line* and drawing the frames that encompass the animation. Fortunately, not all the frames must be manually drawn as Adobe Flash is able to automatically create the required frames between an initial and a final frame. In particular, Adobe Flash does so by tracking which objects have changed in size or position between these two frames and generates the intermediate ones. Using this method, simple animations can be easily developed.

The other way to define the animation behaviour in Adobe Flash is to use *ActionScript*, an object-oriented programming language, which can be used to access the methods and properties of objects drawn in the working area. More complex effects can be achieved when using ActionScript and, what is more important, it allows to create an animation the user can interact with.

Both methods, time line and ActionScript programming, have been used to develop the animations. The first one mainly to define all the case studies and the second to allow the user to navigate through the whole animation.

The animation contents have been actually divided into three different Flash animation files, one for each of the following cache coherence protocols: MSI cache coherence snoopy-based protocol, MESI cache coherence snoopy-based protocol, and MESI cache coherence directory-based protocol (with the last one using the full vector of bits directory implementation). These animations can be downloaded from “http://lorca.act.uji.es/projects/ccp/”.

Fig. 1 captures the graphical interface of the MESI cache coherence snoopy-based protocol. The graphical interface is divided into the following parts:

1. The title of the animation.

2. A detailed description of what is happening in the current step.

3. Previous, play, and next buttons. The user can use them to go back to the previous step, to (re)play the current step, and to move forward to the next step, respectively.

4. The multiprocessor representation (the same representation is used for the MSI snoopy-based protocol, while a quite different one is used for the directory-based protocol).

5. Step buttons. The user can use them to move directly to a given step. This part also shows which is the current step and which steps have already been completed.

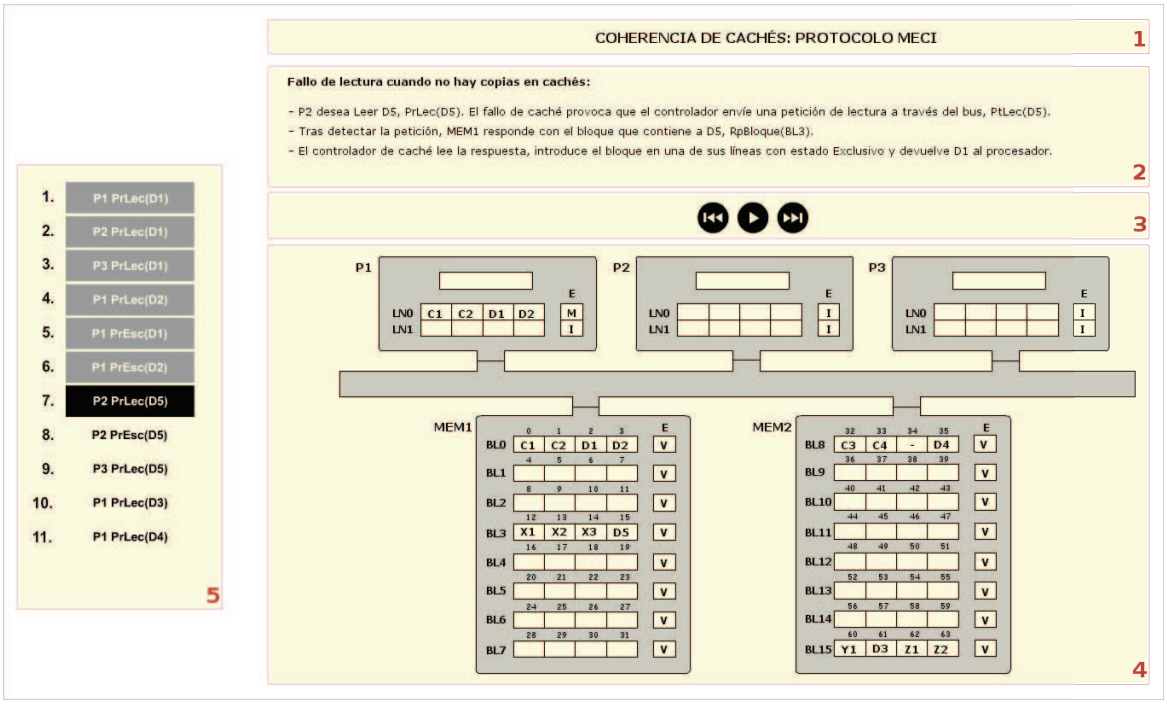

Figure 1. Graphical interface of the MESI snoopy-based cache coherence protocol animation

The multiprocessor shown in the snoopy-based protocols animations consists of three processors (P1, P2, and P3) and two memory banks (MEM1 and MEM2), all of them interconnected via a bus. Each processor has the following elements (from top to bottom and left to right):

- A text area that represents the core of the processor, used to indicate which read or write request is being executed.
- A table that represents the processor's cache contents. The cache has two cache blocks with 4 words each.
- A column that displays the current state of each cache block: one of modified (M), exclusive (E), shared (S), or invalid (I).

The memory has the following elements:

- A table that represents the memory contents, containing 32 words organized as 8 memory blocks with 4 words each.
- A column that shows the current state of each memory block: valid (V) or invalid (I).

The MESI directory-based cache coherence protocol animation uses a different multiprocessor (see Fig. 2), consisting of four nodes interconnected by a network. Each node has a processor, a cache, and a memory alike the ones just described. The main difference between the two multiprocessors is that, in the directory-based one, for each memory block there is a vector of bits that holds the directory information.

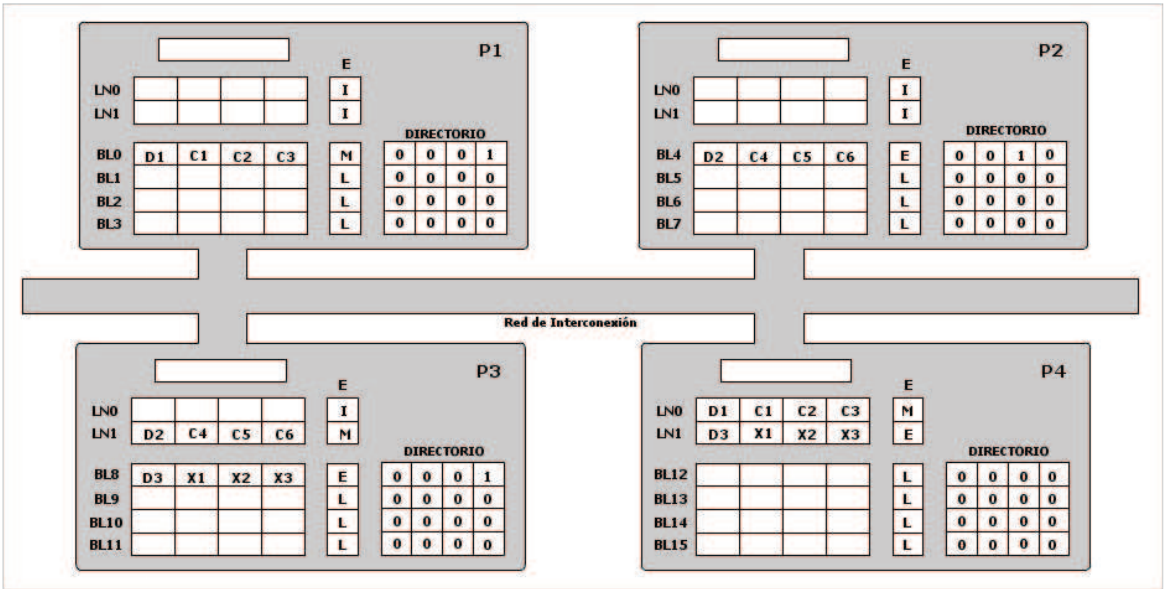

Figure 2. Multiprocessor used by the directory-based cache coherence protocol animation

4 CONCLUSIONS AND FUTURE WORK

We have developed three flash animations that encompass all the possible cases that occur during real operation in the three cache coherence protocols considered. We believe that these animations can help to understand how these cache coherence protocols work and can be effectively used by the teacher in a class. Furthermore, we believe that these animations can also be used by the students to analyse the cache coherence protocols on their own.

Developing the animations has been far more complex than we expected, especially when we needed to introduce changes to the animation's time line. Experience has taught us that details should be carefully planned before commencing the actual animation coding. Last time changes were quite difficult to get right.

Finally, we plan to use the animations in our advanced computer architecture course and obtain feedback from the students in which improvements they think could help to achieve a better understanding of how the different cache coherence protocols work.

REFERENCES

[1] David E. Culler, Jaswinder Pal Singh, Anoop Gupta. Parallel computer architecture: a hardware/software approach. Morgan Kaufmann. 1998.

[2] John L. Hennessy, David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann. 2006.

[3] Julio Ortega, Mancia Anguita, Alberto Prieto. Arquitectura de Computadores. Editorial Thomson. 2005.