J. Parallel Distrib. Comput. 102 (2017) 163–174

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

An adaptive cache coherence protocol: Trading storage for traffic

Lucia G. Menezo *, Valentin Puente, Jose-Angel Gregorio

University of Cantabria, 39005, Santander, Spain

HIGHLIGHTS

- A new adaptive non-inclusive cache coherence protocol.
- Combination of snoop-based and directory-based coherence protocol.
- Non-inclusive directory able to reconstruct sharing information when needed.
- Adaptive filter to minimize coherence traffic.

ARTICLE INFO

Article history:
Received 12 February 2016
Received in revised form
15 November 2016
Accepted 18 December 2016
Available online 28 December 2016

Keywords:
Coherence protocol
Multicore
CMPs

ABSTRACT

This paper introduces a new adaptive cache coherence protocol which minimizes energy requirements and guarantees scalability. It includes two complementary parts: a non-inclusive sparse-directory to track only actively shared blocks and a structure to determine the presence of a block in the private caches based on an improved counting bloom filter. It uses token counting to preserve the system correctness, to improve performance and to reduce the implementation complexity. Combining all these characteristics, the proposal has a low storage overhead and is able to suppress most of the traffic inherent to snoop-based protocols and reduce the size of directory-based structures. Using a capacity to track only 40% of all the blocks allocated in the private caches, this coherence protocol is able to achieve better performance than an over-provisioned sparse-directory with a capacity to track 160% of the blocks kept in private caches. The complementarity of both structures enables the coherence controller to change dynamically the way the storage available is dedicated according to the data-sharing properties of the application. Thus, applications with high-sharing degree will need more directory space while low-sharing degree patterns will need more private block-presence space to include more information. With only 5% of the private cache entries tracked, the average performance degradation is less than 8% compared to a 160% over-provisioned sparse-directory.

1. Introduction

Cache coherence is a huge challenge of future chip multiprocessors (CMPs). In order to maintain the performance improvement, the number of cores will keep on increasing and the pressure on the bandwidth off-chip will continue to grow. One way of alleviating this problem comes by increasing the amount of memory on-chip. However, this increased amount of on-chip memory provokes longer access times, which need to be palliated using a more complex memory hierarchy.

It is the coherence protocol's responsibility to make sure that all potential copies of any block that are scattered over different caches are coherent, i.e. any processor should see the same content of all the memory locations under any circumstance. There is no universal solution and the chosen one will depend on the system. When the number of cores is low, the chosen solution is to use broadcast-based coherence protocols. Actually, this is the method used by current high-performance commercial systems [14,8,16]. Its main advantages come with better performance and lower complexity when compared with other coherence proposals such as directory-based coherence protocols. However, this is achieved with an increment in the total amount of traffic and cache snoops, which will decrease the energy efficiency of the system. It is clear that when the size of the system grows, the impact of this disadvantage will become unsustainable. A more subtle effect, but not less relevant, is the on-chip resource contention that characterizes these protocols. As a result, on-chip access latency can be affected, perhaps degrading the CMP performance under some particular usage scenarios.

* Corresponding author.
E-mail addresses: gregoriol@unican.es (L.G. Menezo), vpuente@unican.es (V. Puente), monaster@unican.es (J.-A. Gregorio).

http://dx.doi.org/10.1016/j.jpdc.2016.12.020
0743-7315/

On the other hand, there are directory-based coherence protocols. Broadcast cache snoops are avoided by using a specific structure to track the block's copies present in the cache hierarchy. However, with this solution new limitations emerge. On the one hand, this approach demands inclusivity. This property requires that the contents of all the smaller caches of a multi-level cache hierarchy have to be a subset of the last-level cache (LLC). When a line is evicted from the LLC, inclusion is enforced by removing that line from all the caches in the hierarchy where it is present. Although from a performance and a cost stand point of view non-inclusiveness is desired, the common assumption is that inclusiveness is difficult to avoid in order to maintain coherence protocol complexity limited. On the other hand, the associativity needed in the directory increases as the number of cores does, making solutions like the duplicate-tag unviable [20]. The solution adopted to overcome this issue is to overprovision the directory to minimize unnecessary evictions in the private caches due to directory conflicts under constrained (and realistic) associativity [15]. However, this method also causes scalability problems as the private cache sizes increase and so the number of tracked blocks does too [35].

From this standpoint, it would appear that a pure coherence protocol might not be the most suitable approach to tackle the problem. Intuitively, it seems that the coherence protocol should somehow hybridize the best of both types: trying to attain the performance effectiveness and implementation cost of a broadcast-based coherence protocol with the energy efficiency of a directory-based one. This paper addresses this task and successfully attains a new coherence protocol, denoted Flask (FiLtered and Allocated just by Shared block Keeper) coherence, which can scale as a directory-based coherence protocol does, while achieving cache effectiveness similar to a broadcast-based one. A previous version of this paper was presented in [28]. Now, several additional explanations with rewritten sections have been added to clarify the description proposal and a more realistic memory model has been used to produce new results, generating a complete and self-contained work.

Flask is based on the idea of having two complementary structures working together on one logical framework. One of them, called *Dir-P*, which is basically an improved Bloom filter [6] to determine the presence or not of any block in the private caches. The other one, called *Dir-S*, works as a sparse directory and tracks the blocks that are actively shared among the cores. The whole system uses *token counting* to guarantee correctness while maintaining the complexity limited. The whole framework keeps detailed information about the location of data that are being actively shared while recording only the presence of the blocks that are allocated privately in the caches (with no sharers), which is the most common case.

In a directory-based protocol every block inside a private cache must be tracked by using an entry in the directory. This is known as *directory inclusivity* and it means that if a new block has to be tracked and there are no available entries in the directory, one of the existing ones has to be replaced. Thus, the block being tracked by the replace-to-be entry has to be invalidated in all the private caches where it is present. However, our protocol is able to handle incomplete information given by the framework, i.e. with the information of both structures, the protocol does not have exact information. This circumvents directory inclusivity and so it avoids the invalidation of blocks in the private caches due to conflicts in the directory structure. When a shared block is not being tracked by Dir-S, after a new request to the controller, a broadcast is sent to all coherence agents (the system structures in charge of the coherence maintenance). The replies are used to reconstruct the corresponding entry in the directory. This approach of reconstructing the directory entries on demand was first introduced by Mosaic [27]. Nevertheless Mosaic allocates directory entries for any on-chip miss (i.e. for both private and shared blocks) and it always generates a broadcast if there is a miss in the directory. On the contrary, the Flask structure *Dir-P* includes the (probable) presence information which avoids the unnecessary search for the block inside the chip. If the block is not present inside the chip, i.e. *Dir-P* does not include the block's information, it sends the request directly to the memory controller. Thus, nearly all of the off-chip requests are not delayed. In the least common case, misses in a private cache of an actively shared block are always tracked by *Dir-P* and dealt with through a multicast to the on-chip coherence agents inside the chip, but avoiding unnecessary off-chip requests.

Finally, the framework introduced allows us to dynamically assign, according to the sharing degree of the running workload, storage capacity in the coherence controller either to track shared blocks in *Dir-S* or to identify privately held blocks in *Dir-P*. This characteristic enables the area dedicated to both structures to be reduced considerably.

The main contributions of the paper are as follows:

- The hybridization of a directory-based and a broadcast-based coherence protocol in a unified logic substrate with optimized implementation and energy costs.
- The proposed strategy achieves the performance of a conventional, over-provisioned sparse directory, while tracking less than 40% of the private cache entries. Similarly, it improves on Token coherence protocol performance by 10% and energy delay product by 20%.
- With only 5% of tracked private cache entries, average performance degradation is less than 10% with respect to a 160% over-provisioned sparse-directory.
- We show that, using an adaptive storage assignation at the coherence controller according to the workload properties, we can reduce even further the resources of the proposal. Matching a sparse-directory performance while tracking only 20% of private cache entries.

2. Motivation

2.1. Directory and broadcast coherence approaches

While directory-based protocols seem to be an attractive approach to enforce cache coherence in a CMP, when the number of cores is high and the on-chip hierarchy complexity grows, the directory is difficult to scale. The main cause is the large number of blocks that have to be tracked as the caches sizes grow. In an on-chip cache, similar to state-of-the-art systems [14,16,9], in order to close the gap in the access time between a small L1 (dominated by processor clock cycle) and a very large LLC (dominated by main memory access time), at least one intermediate level is required. As a consequence the number of blocks that the directory has to track is even larger. Additionally, those intermediate levels usually have a substantial associativity. Moreover, recent designs [14,16] also require a large associativity for L1. In summary, the number of blocks that can be mapped in a set of the directory is high.

Although in some early CMPs [20] the directory has enough capacity to keep information about all the blocks allocated in the private caches, when the number of cores or private cache complexity and size grows, this is not feasible due to the enormous associativity required by the directory. However, reducing this associativity increases the eviction of blocks in the private caches due to conflicts in the directory. A rule of thumb [15] suggests that over-provisioning the directory with twice the capacity required to track the private caches will diminish the number of invalidations [11]. Nevertheless, the larger number of tracked