Microprocessors and Microsystems 38 (2014) 263–275

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Victim retention for reducing cache misses in tiled chip multiprocessors

Shirshendu Das *, Hemangee K. Kapoor

Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam 781 039, India

ARTICLE INFO

Article history:
Available online 13 November 2013

Keywords:
Chip-multiprocessor
Victim retention
NUCA

ABSTRACT

This paper presents CMP-VR (Chip-Multiprocessor with Victim Retention), an approach to improve cache performance by reducing the number of off-chip memory accesses. The objective of this approach is to retain the chosen victim cache blocks on the chip for the longest possible time. It may be possible that some sets of the CMPs last level cache (LLC) are heavily used, while certain others are not. In CMP-VR, some number of ways from every set are used as reserved storage. It allows a victim block from a heavily used set to be stored into the reserve space of another set. In this way the load of heavily used sets are distributed among the underused sets. This logically increases the associativity of the heavily used sets without increasing the actual associativity and size of the cache. Experimental evaluation using full-system simulation shows that CMP-VR has less off-chip miss-rate as compared to baseline Tiled CMP. Results are presented for different cache sizes and associativity for CMP-VR and baseline configuration. The best improvements obtained are 45.5% and 14% in terms of miss rate and cycles per instruction (CPI) respectively for a 4 MB, 4-way set associative LLC. Reduction in CPI and miss rate together guarantees performance improvement.

1. Introduction

Advancement in semiconductor technology is allowing to pack more and more transistors on a single die, thus increasing the complexity of the systems. Physical limitations such as heat dissipation and data synchronization tend to create an upper bound on the CPU performance. To improve CPU utilization the alternative generally used is instruction level parallelism, like pipelining. Many applications are better suited for thread level parallelism which is the major impetus behind multiprocessors systems, where each core can handle one thread at any moment of time. Chip Multiprocessors (CMPs) that contain multiple CPUs on the same die are the main components for building today's computer systems [1]. Several CMP based architectures [2–4] have already found their way into the commercial market. In the long run, it is expected that the number of cores in CMPs will increase and also accommodate large on-chip last level cache (LLC) [5].

CMP cache architectures are mainly of two types [1]: (i) CMP with a private LLC and (ii) CMP with a shared LLC. In this paper L2 cache is considered to be the LLC. Both types of architectures have separate L1 cache for data/instructions with each core, but they differ in the physical placement of the LLC (L2). Private L2 caches are relatively small and placed physically very near to the core, thus having the faster cache access. However as the cache size is small, it causes several capacity misses. Multiple copies of the same data block may be present in separate L2 caches, leading to maintain L2 level coherence. On the other hand, shared L2 is comparatively larger and only a single copy of each data block can be stored in it. All the requesting cores share the same data block and the cache storage can be dynamically allocated to the cores depending on their workloads. Shared LLC increases hit time,[1] due to a much larger cache size compared to the private LLC. Since both private and shared LLC have some advantages and some disadvantages, there exists research [7–9] combining the advantages of both.

Most of the cache addressing techniques map a particular block to a fixed set. This set may be distributed over multiple banks as in the case of D-NUCA [10] (Dynamic Non-Uniform Cache Access), but the block is always placed within the same set. Even if the block migrates from one bank to another it remains within the same set. There are some exceptions like NuRAPID [11] where data can be placed in any set, but NuRAPID also uses the same concept of NUCA for its tag array. The mapping occurs based on the set-index of the block address. Now, if in some kind of applications, a group of blocks are being used continuously, and they all map to the same set, then it may be possible that such sets become heavily used while certain others are not. Most of the CMP-cache architectures use basic replacement policies like LRU or Pseudo-LRU [1]. But these replacement policies manage each cache set separately.

* Corresponding author.
E-mail addresses: shirshendu@iitg.ernet.in (S. Das), hemangee@iitg.ernet.in (H.K. Kapoor).

[1] Hit time is the time required to access a block if the requested block is contained in the cache [6].

0141-9331/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2013.11.005

So the heavily used sets need to perform large number of replacements in order to allocate newly requested blocks. On the otherhand, some other sets may remain idle. We propose a cache management technique CMP-VR to handle such situations.

In LRU when a set is full and a new block request arrives, one block from the set (victim block) is selected to be replaced with the newly requested block. Now, for a heavily used set, this victim block may be required again in near future. Hence, CMP-VR gives the victim block another chance to remain in the cache; by moving the victim block into a special region of the cache called reserve storage (*ResS*). In each cache bank, some number of ways (25–50%) from every set are reserved for *ResS*. The remaining cache storage is called regular storage (*RegS*). CMP-VR allows a victim block from a heavily used set to be stored into the reserve space of another set. In this way the load of heavily used sets can be distributed among the underused/idle sets. On a cache miss in *RegS*, the block is searched in *ResS*. If the block is found in *ResS* then it is migrated into *RegS* and considered as a cache hit; otherwise, a cache miss occurs. A separate tag-array is used to manage (searching, replacement, allocation, etc.) the *ResS*. In the worst case (when all sets are equally used) the technique performs same as basic LRU without *ResS*, but never less than that.

The mechanism is implemented separately for each LLC bank and it is completely transparent from outside the bank. In this paper CMP-VR is only implemented for SNUCA (static NUCA) based shared LLC, but it can also be implemented for any other type of NUCA designs (considering both private and shared LLC). Also, other inter-bank data management policies like co-operative caching [8] and victim-replication [7] can be implemented on top of CMP-VR.

The rest of the paper is organized as follows. The next section presents the related works in CMP cache architectures and cache management policies. Motivation behind the proposed technique is presented in Section 3. Section 4 describes the proposed cache management technique (CMP-VR). Experimental details are given in Section 5. Concluding remarks are given in Section 6.

2. Related work

Recent chip multiprocessors (CMPs) use low-radix (2D-mesh) network of interconnected tiles. Each tile can be composed of a single processor or a set of few processors. The tiles usually have their private L1 cache. Each tile may have a private L2 cache or alternatively, all the tiles may share a global L2 cache. The global L2 cache is logically a monolithic block, but due to physical constraints it is efficient to divide this L2 into banks and place each bank at a separate location. The banks of L2 are shared by all the tiles/processors. Each bank maintains its own directory to store coherence information. Another variant is to store all the directory information in one centralized directory. In a shared-banked L2 cache, a L1 cache miss is mapped to exactly one bank depending on the address of the data block. As the requesting L1 may be located far from the serving L2 bank, this can lead to varying access times for the same memory block. However, in this style the total size of L2 available to all the tiles is very large.

Many cache management policies have been proposed in the recent years trying to balance the factors: latency and cache capacity. In [12], they divided 16 tiles into four groups where each group contains four tiles with four L2 banks. The four L2 banks are in shared scheme and the distributed directories control the cache coherence through query messages. This reduces the access latency as the L2 is nearby in the cluster.

In [13–15] the evicted L1 cache blocks are removed and stored in the L2 bank of the same tile; these blocks are called replicas. In this hybrid scheme, the replica is accessible to its own processor and not the nearby cores; such cores have to access it via the global L2 bank, which may be far away.

In [16] the L2 banks are private to start with but become shared due to spilling of data. The evicted L2 cache blocks are spilled onto a neighboring L2 bank increasing the lifetime of the cache block; however this does not scale as the number of processors increases and has the overhead of cache coherence engine.

In cluster based approach of [17–19], the processors are divided into clusters where each cluster has an L2 cache bank and a directory to maintain the information about the cluster. Another directory sits near the shared main memory to maintain information about all the clusters.

In [9], the number of L2 banks a processor can access varies dynamically and known as its own cache dimension. A mapping function is used at each L1 for sending query messages to the L2 cache. If the block is not in the L2 cache of the current cache dimension, the directory is contacted since other processors might have already loaded the data. This increases the number of shared data copies on the chip and thus decreases the overall cache capacity. Also, even if a neighboring L1 has the block, the requester has to wait for the directory to respond (which increases the access latency).

In CMP, each core can run different applications and hence it is beneficial to partition the LLC among the cores [1]. Dynamic partitioning of LLC based on the applications requirement is also an important factor [20]. Several approaches for LLC partitioning have already been proposed [21–23]. Some recent proposals for improving the performance of LLC are [24–26].

Replacement policies have been well-studied in the past [27,28], but the emergence of large shared LLCs has led to innovations in this area. It is generally believed that some variations of LRU policy are best at retaining relevant blocks in the cache. Many alternative innovations for replacement policy has also been proposed [28–30]. Hybrid architectures use replacement policies to allow victim blocks to remain on the chip for as long as possible. They spill the victim block into a separate tile but not within the same tile. So an access to the spilled block causes longer hit latency as it has to search other tile to get the block. CMP-VR allows the victim block to remain in the same tile (same L2 bank), which reduces the access time for further requests to the victim block.

The non-uniform usage of cache sets is a major cause for higher conflict misses [31]. Way sharing cache [32], allows a pair of sets to share some common ways between them such that the load of one set can be shared with the other one. Both [31,33] solve this issue by dynamically increasing the associativity of the heavily used sets without increasing the cache size. In [31], the associativity is increased by increasing the number of tag-store entries relative to the number of data lines. For example, with twice the tag-store entries as data lines the associativity of a heavily used set can be doubled. CMP-VR is also based on a similar concept but uses a different technique (as discussed in Section 1).

2.1. Tiled chip multiprocessor

The baseline design of this work is a Tiled chip multiprocessor (TLA) [1] as shown in Fig. 1. TLA can scale well as the number of processors increase. Each tile has a processor, a private L1 cache and an L2 cache. The tiles (or processor nodes) are connected to each other over a 2D mesh popularly known as network-on-chip (NoC). The L2 cache with each tile can be private, or shared among all the processors on the chip. This paper assumes a shared cache, where the slice located in each tile is called a cache-bank. The cache-bank within which a block resides can be identified by applying a hash function on the block address. This cache-bank is called the home-tile for the given address (or its associated cache block). Thus, every address maps to exactly one home-tile and