Microprocessors and Microsystems 39 (2015) 286–295

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

A new cache replacement algorithm for last-level caches by exploiting tag-distance correlation of cache lines

Cong Thuan Do [a], Hong-Jun Choi [b], Jong Myon Kim [c], Cheol Hong Kim [a,*]

[a] School of Electronics and Computer Engineering, Chonnam National University, Gwangju 500-757, Republic of Korea
[b] The Affiliated Institute of ETRI, Daejeon 305-390, Republic of Korea
[c] School of Electrical Engineering, University of Ulsan, Ulsan 680-749, Republic of Korea

ARTICLE INFO

Article history:
Available online 14 May 2015

Keywords:
Cache
Replacement policy
Performance
Cache miss
Cache tag

ABSTRACT

Cache memory plays a crucial role in determining the performance of processors, especially for embedded processors where area and power are tightly constrained. It is necessary to have effective management mechanisms, such as cache replacement policies, because modern embedded processors require not only efficient power consumption but also high performance. Practical cache replacement algorithms have focused on supporting the increasing data needs of processors. The commonly used Least Recently Used (LRU) replacement policy always predicts a near-immediate re-reference interval, hence, applications that exhibit a distant re-reference interval may perform poorly under LRU replacement policy. In addition, recent studies have shown that the performance gap between LRU and theoretical optimal replacement (OPT) is large for highly-associative caches. LRU policy is also susceptible to memory-intensive workloads where a working set is greater than the available cache size. These reasons motivate the design of alternative replacement algorithms to improve cache performance. This paper explores a low-overhead, high-performance cache replacement policy for embedded processors that utilizes the mechanism of LRU replacement. Experiments indicate that the proposed policy can result in significant improvement of performance and miss rate for large, highly-associative last-level caches. The proposed policy is based on the *tag-distance* correlation among cache lines in a cache set. Rather than always replacing the LRU line, the victim is chosen by considering the *LRU-behavior* bit of the line combined with the correlation between the cache lines' *tag*s of the set and the requested block's *tag*. By using the LRU-behavior bit, the LRU line is given a chance of residing longer in the set instead of being replaced immediately. Simulations with an out-of-order superscalar processor and memory-intensive benchmarks demonstrate that the proposed cache replacement algorithm can increase overall performance by 5.15% and reduce the miss rate by an average of 11.41%.

1. Introduction

In recent years, innovation in processor architecture has created a gap between processor and memory performance that has been rapidly increasing. Processor performance is now crucially determined by the amount of memory accesses served from on-chip caches since the cost of access to main memory has grown to hundreds of cycles. In response to the need for better caching, many researchers have extensively attempted to increase the cache size or to hide latency through sophisticated prefetching and out-of-order execution techniques. In fact, adding a cache hierarchy between the processor and main memory is still regarded as a primary means to hide the memory latency from the processor. However, when compared to the development of sophisticated prefetching or out-of-order techniques, cache replacement algorithms have had relatively little recent works to improve their performance. Indeed, the implicit assumption seems to be that the Least Recently Used (LRU) policy is good enough, and that there are few that can increase hit rates for a given size without adding expensive logic to the cache.

As the processor/memory gap continues to grow, it becomes necessary to improve the behavior of the cache hierarchy. This can be simply done by increasing its associativity to reduce the number of *conflict misses*. When cache associativity increases, however, the replacement algorithm becomes more important [1]. As a result, practical cache replacement policies, which attempt to emulate Belady's optimal replacement (OPT) [2] by predicting the re-reference of a cache block, play a central role in the design of memory hierarchy.

* Corresponding author. Tel.: +82 62 530 1806.
E-mail address: chkim22@jnu.ac.kr (C.H. Kim).

http://dx.doi.org/10.1016/j.micpro.2015.05.005
0141-9331/

An OPT policy executes replacement decisions by using perfect knowledge of the re-reference (or reuse) pattern of each cache block, and thus, OPT can ideally identify the block in a set that will be accessed farthest in the future [1]. Practical cache replacement policies, on the other hand, make their replacement decisions based on predictions of which one of the blocks will be re-referenced farthest in the future and select that block for replacement. Typically, on a miss, replacement policies make a prediction of when the missing block will be re-referenced next. These predictions can be improved when more information on the block is available.

In an LRU replacement mechanism, the LRU stack (or chain) represents the recency of the cache lines referenced. The MRU position represents a cache line that was most recently used while the LRU position represents a cache line that was least recently used. Although LRU policies provide good performance for workloads with high data locality, they shows poor performance when the prediction of a near-immediate re-reference interval is incorrect. Applications with re-references that only occur in the distant future perform poorly under LRU policy.

Prior studies have shown that there are some main disadvantages in using an LRU policy. The first one is that cache lines, called *dead lines* (or *dead blocks*), will eventually be replaced after their last use [3]. Under LRU replacement, dead lines are not immediately replaced since they have to traverse through the LRU stack before reaching the LRU position. Such dead lines waste the cache capacity that would otherwise be available for other lines. Moreover, the travel time of such dead lines will be longer when the associativity is high, a recent trend in the design of lower-level caches like Last-Level Caches (LLCs). As a consequence, the performance gap between LRU and OPT increases despite the fact that larger associativity provides better cache performance.

The second disadvantage of LRU replacement is that temporal locality may become inverted when there are multiple levels of caches due to the filtering effect that the upper-level caches hide the lower-level caches upon an upper-level hit. This causes inversion of the temporal re-reference patterns of the addresses as observed by the lower caches.

In modern embedded systems, applications have become very large, and the presence of caches is inevitable. Therefore, recent embedded processors have cache architectures that are as complex as those for general purpose processors. Embedded processor caches, especially for mobile devices, have complicated architectures since all three metrics (performance, power, and area) have to be simultaneously satisfied within given constraints. However, since the L1 cache is optimized for short hit time, the hit time of L2 cache is less critical, allowing for smarter cache management strategies [3,4]. An efficient cache replacement algorithm is important not only to increase the performance, but also to reduce power consumption. According to previous studies [5–9], memory system access accounts for about half of the total energy consumed by embedded systems. Thus, reducing memory accesses would provide higher energy savings. Over the last few decades, many good cache replacement policies have been proposed, but none achieve comparable performance to the LRU policy without incurring high overhead, which is an important factor in the design of embedded processors. This motivates the development of a new, effective cache replacement policy for embedded processors.

In this paper, we propose a low-overhead, high-performance L2 cache replacement policy for embedded processors that improves the hit rates of the L2 cache. We consider the L2 cache to be an LLC in this work, and we utilize the existing mechanism of the LRU policy. Our replacement policy makes a replacement decision by considering the *tag-distance* correlation among the lines of a cache set and the *LRU-behavior* bit of each line. Detailed simulations on 16 SPEC applications and 4 MiBench applications show that our mechanism improves the performance for 9 memory-intensive applications by 5.15% on average and reduces the overall L2 cache miss rate by 11.41% without slowing down any of the remaining 11 applications by more than 1%. Furthermore, our proposed policy only incurs a very small additional storage overhead, about 1.1 KB (0.2% of 512 KB cache).

The rest of this paper is outlined as follows. In the next section we discuss related work. Section 3 describes and analyzes the LRU replacement policy in detail. In Section 4, we present our proposed cache replacement policy. Sections 5 and 6 discuss the evaluation methodology and the experimental results obtained. Finally, Section 7 concludes the paper.

2. Related work

Cache replacement algorithms have been developed to improve cache performance in modern microarchitectures. Belady was the first author to propose an optimal (OPT) replacement algorithm in the context of paging systems that victimizes the block in a cache set with the longest distance with respect to future usage [2]. Later, Mattson et al. introduced an OPT to compute the optimal miss count of a trace [10]. Although OPT guarantees a minimal miss count, it requires knowledge of the future, and thus remains theoretical and can only be emulated for real systems [11].

Recent studies have shown that the performance gap between LRU and OPT has become larger for highly-associative caches. In order to bridge the LRU/OPT gap, dead-block predictors [1,12,13] were proposed to predict blocks that will no longer be used during current generation times in the cache, and these blocks are then evicted early while preserving the live blocks. Such dead-block prediction can be performed in software [14] or hardware [3,15–17]. According to the state of the predictor, hardware solutions can be classified into three categories: trace-based, count-based, and time-based.

In modern systems, observation of the non-uniform nature of cache misses has led to a new approach for cache replacement algorithms. These algorithms assign different costs to cache blocks according to well-defined criteria, and they rely on these costs to select the victim on a cache miss. In the context of uniprocessors, Jeong et al. [18,19] proposed a cost-sensitive cache replacement algorithm that assigns cost based on whether a block's next access is predicted to be a load (high-cost) or store (low-cost). Cost-sensitive replacement algorithms to reduce the cost of cache misses rather than miss rates have been proposed [10–22].

Recently, there has been a renewed interest in cache bypassing schemes. Several schemes have been proposed to improve cache efficiency, and they can be categorized into static and dynamic approaches. Static approaches [23,24] rely on a profile-guided compiler to identify the lines that should bypass the cache, and dynamic cache bypassing uses runtime behavior for the prediction to identify bypassing opportunities [3,25–27].

3. Background

In this section, we review the mechanism of the LRU replacement policy on the LLC and analyze its shortcomings in detail.

Fig. 1 illustrates the LRU stack (or LRU chain) of the LRU replacement policy in the LLC. The data in the LRU stack is sorted according to the last time it was used. When a miss occurs in L1, the request is forwarded to the LLC. If the requested data is not cached in the LLC, it will be copied from the main memory into the corresponding set in the LLC. In case of a full LRU stack, the LRU policy will pick the line in the LRU position for replacement. The new item, then, is placed into the top of the stack in the MRU position.