Microprocessors and Microsystems 38 (2014) 98–111

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

An effectiveness-based adaptive cache replacement policy

Geng Tian *, Michael Liebelt

School of Electrical and Electronic Engineering, The University of Adelaide, Adelaide, South Australia, Australia

ARTICLE INFO

Article history:
Available online 6 December 2013

Keywords:
Cache memory
Configurable cache
Performance
Thrashing
Scan resistance
Micro architecture

ABSTRACT

Belady's optimal cache replacement policy is an algorithm to work out the theoretical minimum number of cache misses, but the rationale behind it was too simple. In this work, we revisit the essential function of caches to develop an underlying analytical model. We argue that frequency and recency are the only two affordable attributes of cache history that can be leveraged to predict a good replacement. Based on those two properties, we propose a novel replacement policy, the Effectiveness-Based Replacement policy (EBR) and a refinement, Dynamic EBR (D-EBR), which combines measures of recency and frequency to form a rank sequence inside each set and evict blocks with lowest rank. To evaluate our design, we simulated all 30 applications from SPEC CPU2006 for uni-core system and a set of combinations for 4-core systems, for different cache sizes. The results show that EBR achieves an average miss rate reduction of 12.4%. With the help of D-EBR, we can tune the weight ratio between 'frequency' and 'recency' dynamically. D-EBR can nearly double the miss reduction achieved by EBR alone. In terms of hardware overhead, EBR requires half the hardware overhead of real LRU and even compared with Pseudo LRU the overhead is modest.

1. Introduction

The purpose of a cache is to temporarily hold data blocks accessed by the processor, keeping them close to the processor in case they are going to be used again in the future. In today's multi-core systems the latency of accesses to the last level cache (LLC) can reach as many as hundreds of cycles. Any hit in the LLC means a reduction in the overall running time since an extremely long main memory fetch latency is eliminated. Unfortunately this benefit is not achieved for free. Practical limitations on the area and power consumption of a cache constrain its size. This means the cache will be fully occupied within a short period of time and after that, if a new data block must be accommodated, a replacement decision has to be made. Many replacement algorithms have been proposed over time, but when we apply the power, area and hardware overhead constraints when designing a CPU, not a lot of them are practical. In this paper we will re-examine the essential function of a cache and find the underlying analytical model of cache replacement. Based on that model we will propose a new replacement policy, Effectiveness-Based Replacement policy (EBR).

The purpose of a good replacement policy is to make sure that the data stored in the cache can be re-used as many times as possible. We propose a new definition of an optimal cache replacement policy which can be used to explain many widely accepted replacement policies. Replacement policies involving PC indexes or past access intervals have been investigated by many researchers, but many of them dramatically complicate the structure of the cache. We propose to use the readily-obtainable measures of recency and frequency to develop a replacement scheme that is not significantly more complicated than the industry-preferred pseudo LRU. The ideal Least Recently Used (LRU) policy uses recency information only, while Least Frequently Used (LFU) uses frequency only. However in the absence of the other information, LRU and LFU can produce problems, such as *thrashing* and *scan* in the case of LRU and *ageing* in the case of LFU [1]. There has been previous research which proposed to integrate LFU with LRU [2–7], but these works either aim at the page level with unaffordable hardware complexity for caches or use set duelling to select between LRU and LFU, which is not an efficient method of combination.

EBR combines the measures of recency and frequency to form a rank sequence inside each set and, upon each miss, the block with lowest rank is evicted. EBR assumes that the weight of recency should be much higher than frequency. It divides the recency stack into several subgroups, each representing a recency level. Inside each group, frequency determines the ranking. However, there are cases when the assumption that recency is more important than frequency does not work effectively and to get best performance we should rely more on LFU than LRU. Dynamic-EBR uses a modified set duelling [8] technique to find the most suitable

* Corresponding author.
E-mail addresses: tian@eleceng.adelaide.edu.au (G. Tian), michael.liebelt@adelaide.edu.au (M. Liebelt).

0141-9331/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2013.11.011

way to divide recency subgroups. With the help of D-EBR, we can double the improvement made by EBR alone.

The rest of this paper is organised as follows. In Section 2 we discuss the objective of cache replacement and propose a new definition of optimal replacement. Section 3 discusses the deficiencies of commonly used replacement policies and their causes. Sections 4 and 7 introduce EBR and D-EBR, respectively. Sections 5, 6 and 8 present the results of our simulations and Section 9 focuses on the complexity sensitivity of D-EBR. Section 10 talks about the hardware overhead, Section 11 discusses related works and finally Section 12 previews future work.

2. Re-definition of cache optimal replacement

The essential function of a cache is to hold blocks that will be accessed again in the future. If a block is not going to be re-referenced in the future then there is no purpose served in keeping it in the cache. To do so wastes power and the potential chance for other blocks to achieve hits during that period of time.

Let $R_i(t, t+\Delta t)$ denote the number of re-references to block i if it can reside in cache during the future period between current time t and $t+\Delta t$. Assume for the moment that there is only one way in the cache, and that at time $T1$, two blocks A and B are candidates for eviction, whose future re-reference pattern follows Fig. 1a. One of these is a new block and the other is the block already present. In the future, $R_A(T1, T10)$ would be 4 and $R_B(T1, T10)$ would be 5. In other words, if we leave A in the cache in this example we will achieve 4 hits and 5 misses whereas if we retain B, we will achieve 5 hits and 4 misses.

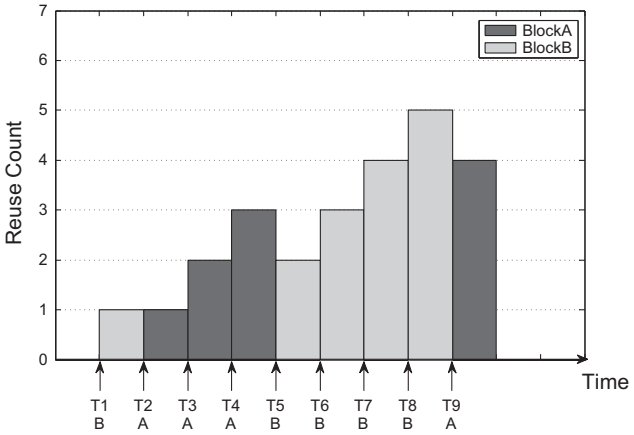
To evaluate the above situation more precisely, we introduce a new metric. We define *Effectiveness* $E_i(t, t+\Delta t)$ of a block i as the rate of re-use of the block over future time period Δt if this block was allocated in the cache. $E_i(t, t+\Delta t) = \frac{R_i(t,t+\Delta t)}{\Delta t}$. The higher the Effectiveness of a block, the more benefit we would gain if we let this block reside in cache during that period of time. If we always allocate blocks with maximum E to the cache, we will maximise the number of hits. Eq. (1) gives the total Effectiveness, ξ, of all of the on-chip blocks during the period of execution.

$$\xi = \sum_{j=0}^{last-1} E_i(t_j, t_{j+1}) \quad \text{where } t_0 = start \text{ and } t_{last} = end \tag{1}$$

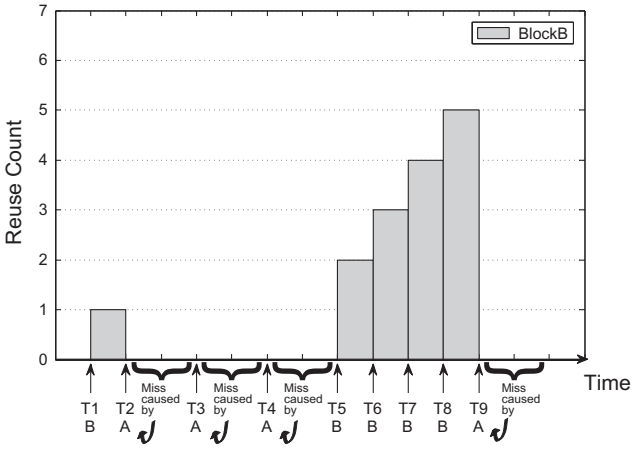
A maximised ξ will guarantee a maximised number of cache hits and minimum number of cache misses, so any policy that maximise ξ can be considered optimal. Hence we now propose a new definition for the ultimate goal of a cache replacement policy:

Definition 1. The Optimal Replacement Policy is the one that maximises ξ.

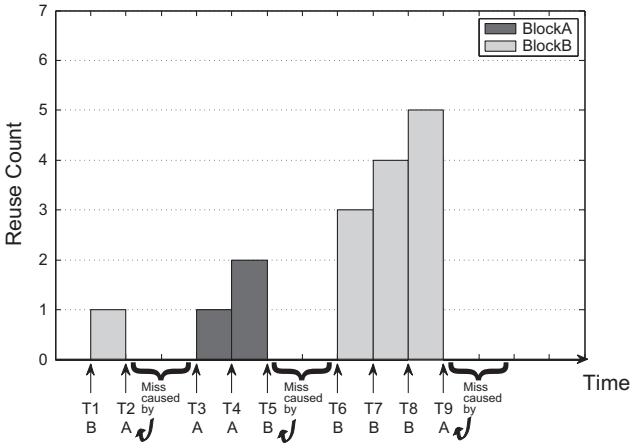
Considering that the chance to rearrange in-cache block combinations only occurs when there is a replacement, and a replacement only occurs when a miss has occurred, it will be too late to include the effect of the miss on deciding the following replacement. (We do not consider early replacement here, since bringing the block into the cache at a time before it is to be used is actually a waste of efficiency through occupying the cache earlier without any contribution to performance.) Hence, we propose the following replacement strategy: upon a replacement, those blocks that have the largest Effectiveness during the time interval, from the moment when the current replacement ends to the moment when the next replacement ends, can remain in the cache. We denote the time interval from the moment when the current replacement finishes until the moment when the next replacement finishes as a *time slice*. We can then define the optimal Effectiveness based replacement strategy as:



(a) Future access sequence of block A and block B



(b) Block B remains in the cache all the time



(c) Swap block B with block A between T2 to T5

Fig. 1. Example 1 to illustrate block Effectiveness.

Definition 2. Optimal effectiveness based replacement strategy: Upon a replacement, retain those blocks that will maximise Effectiveness in the following time slice.