Death Threshold of L2 Cache Block Classes in CHAR Algorithms

Tuning Suggestions

Mainak Chaudhuri
Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur 208016
INDIA
mainakc@cse.iitk.ac.in
Date: 2nd October, 2012

This brief note should be read in conjunction with the proposal on making replacement and bypass algorithms for last-level caches (LLCs) hierarchy-aware [1]. That proposal introduced cache hierarchy-aware replacement (CHAR) and bypass algorithms. One central parameter of these algorithms is the threshold applied to the reuse probability (or hit rate) in a class of cache blocks to decide if the class of blocks is dead. Such blocks can be replaced early in an inclusive LLC or bypassed in an exclusive LLC. A dynamic algorithm for determining this threshold t is discussed and evaluated in [1]. This algorithm chooses t such that blocks from classes with hit rates below the prevailing baseline hit rate would be identified as dead. An implementable approximate version of this algorithm is discussed in [1] and reproduced in Equation (1) below.

$$t = \begin{cases} 1/16 & \text{if } E_4 \leq N_E/8 \\ 1/8 & \text{if } N_E/8 < E_4 \leq N_E/4 \\ 1/4 & \text{if } N_E/4 < E_4 \leq N_E/2 \\ 1/2 & \text{if } E_4 > N_E/2 \end{cases} \tag{1}$$

N_E maintains the total number of L2 cache evictions mapping to the LLC sample sets. E_4 maintains the number of L2 cache evictions of blocks belonging to class C_4. It is possible to replace such a dynamic value of t by a static predetermined constant t.

Figure 1 compares the dynamic algorithm with a number of static t values (1/2, 1/4, 1/8, 1/16, and 1/32) for one hundred four-way multi-programmed workloads with hardware prefetcher enabled (see [1] for configuration details). For both inclusive and exclusive LLCs, the baseline is an inclusive LLC implementing the SRRIP replacement policy [2]. As can be seen, the dynamic policy delivers performance better than static $t = 1/2$ but worse than $t = 1/4, 1/8, 1/16, 1/32$ for our choice of workloads. While our dynamic algorithm tries to eliminate blocks from classes with hit rates below the prevailing baseline hit rate, for certain workload classes $t = 1/2$ can be very aggressive, as can be seen from the static $t = 1/2$ results.

One possible tuning technique for the dynamic algorithm would be to choose t such that it eliminates blocks from classes with hit rates below, say, $1/2^k$th of the prevailing baseline hit rate. This would lead to the following approximate algorithm.

$$t = \begin{cases} 1/(16 * 2^k) & \text{if } E_4 \leq N_E/8 \\ 1/(8 * 2^k) & \text{if } N_E/8 < E_4 \leq N_E/4 \\ 1/(4 * 2^k) & \text{if } N_E/4 < E_4 \leq N_E/2 \\ 1/(2 * 2^k) & \text{if } E_4 > N_E/2 \end{cases} \tag{2}$$

Therefore, $k = 1$ would result in t values ranging from 1/4 to 1/32, while $k = 2$ would lead to t values in the range 1/8 to 1/64. The value $k = 0$ corresponds to the dynamic algorithm discussed in [1].

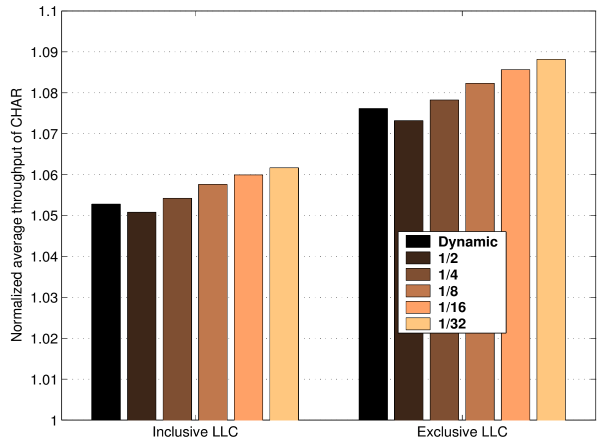

Figure 1. Normalized throughput comparison for static and dynamic t on one hundred four-way multi-programmed workloads with hardware prefetcher enabled.

Since too small a value of t may lead to lost opportunities and too large a value may lead to loss in performance due to aggressive death prediction, another alternative approach to tuning the dynamic algorithm would involve fixing a minimum and a maximum allowable t value, say, t_{min} and t_{max}, respectively. Next, we take Equation (2) and choose k such that t_{min} equals $1/(16 * 2^k)$, i.e., the minimum value of t in Equation (2). Finally, we merge all the ranges in Equation (2) that have values of t more than t_{max} with the range that has value t_{max}. As an example, suppose t_{max} is 1/8 and t_{min} is 1/32. This leads to $k = 1$ and the following dynamic algorithm.

$$t = \begin{cases} 1/32 & \text{if } E_4 \leq N_E/8 \\ 1/16 & \text{if } N_E/8 < E_4 \leq N_E/4 \\ 1/8 & \text{if } E_4 > N_E/4 \end{cases} \tag{3}$$

Similarly, if we set t_{max} to 1/8 and t_{min} to 1/16, we get the following dynamic algorithm.

$$t = \begin{cases} 1/16 & \text{if } E_4 \leq N_E/8 \\ 1/8 & \text{if } E_4 > N_E/8 \end{cases} \tag{4}$$

In summary, when choosing a value of t it should be kept in mind that too small a value may lead to performance close to the baseline due to lost opportunities, while too large a value may lead to loss in performance due to aggressive death prediction. In general, we have found that a small conservative static value of t works well e.g., $t = 1/8, 1/16$. However, a well-tuned dynamic algorithm may be desirable so that the CHAR policy can adapt to varying workload characteristics. In this brief note, we have proposed a couple of tuning strategies for choosing a dynamic value of t.

References

[1] M. Chaudhuri et al. Introducing Hierarchy-awareness in Replacement and Bypass Algorithms for Last-level Caches. In *Proceedings of the 21st IEEE/ACM International Conference on Parallel Architecture and Compilation Techniques*, pages 293–304, September 2012.

[2] A. Jaleel et al. High Performance Cache Replacement using Re-reference Interval Prediction (RRIP). In *Proceedings of the 37th International Symposium on Computer Architecture*, pages 60–71, June 2010.