Reverse Engineering of Cache Replacement Policies in Intel Microprocessors and Their Evaluation

Andreas Abel and Jan Reineke
Department of Computer Science
Saarland University
Saarbrücken, Germany
Email: {abel, reineke}@cs.uni-saarland.de

***Abstract*—Performance modeling techniques need accurate cache models to produce useful estimates. However, properties required for building such models, like the replacement policy, are often not documented. In this paper, using a set of carefully designed microbenchmarks, we reverse engineer a precise model of caches found in recent Intel processors that enables accurate prediction of their cache performance by simulation. In particular, we identify two variants of pseudo-LRU that, unlike previously documented policies, employ randomization. We evaluate their performance and demonstrate that it differs significantly from known pseudo-LRU variants on some benchmarks.**

I. Introduction

To bridge the increasing latency gap between the processor and main memory, modern microarchitectures employ memory hierarchies with multiple levels of cache memory. These caches are small but fast memories that make use of temporal and spatial locality. Typically, they have a big impact on the execution time of computer programs.

In recent years, different approaches have been proposed to estimate the performance of software systems. This includes analytical modeling, as well as simulation-based and profile-based prediction techniques. All these approaches need sufficiently detailed models of the cache hierarchy in order to produce useful estimates. Similarly, such models are an essential part of worst-case execution time (WCET) analyzers for real-time systems [1]. Furthermore, information on cache properties is also required by self-optimizing software systems, as well as platform-aware compilers.

Unfortunately, documentation of relevant properties at the required level of detail is often not available, or may be misleading. One such property is the cache replacement policy. In this paper, we develop a novel set of microbenchmarks to reverse engineer replacement policies used in recent Intel processors. Based on the results we obtain from these microbenchmarks, we then propose models for the replacement policies that are detailed enough to precisely predict the performance of applications by simulation. Finally, we compare these policies to well-known existing ones by evaluating their performance on the PARSEC benchmark suite.

II. Problem Description

CPU Caches are structured as follows. They consist of a number of *cache sets*, each of which can store k memory blocks from the main memory (k is also called the *associativity* of the cache). A specific memory block can only be stored in one cache set; this set is determined by a part of the block's memory address. Upon a cache miss, a so called *replacement policy* must decide which memory block of the corresponding set to replace. One popular strategy is to replace the least-recently used (LRU) block. As the cost of implementing this policy is rather high for larger associativities, processors often use a tree-based approximation to LRU, called *pseudo-LRU* or PLRU (for details we refer to [2]).

In [3] we observed that several Intel Core 2 Duo CPUs appear to use different replacement policies for their L2 caches. According to Intel [4], these CPUs "use some variation of a pseudo LRU replacement algorithm". However, while we could verify that the Core 2 Duo E6300 (2MB, 8-way set-associative cache) uses the tree-based PLRU policy, the cache behavior of the E6750 (4MB, 16-way set-associative) and the E8400 (6MB, 24-way set-associative) was found to be different from previously documented PLRU variants. The following experiment reveals these differences:

1) Clear the cache.
2) Access one block in all cache sets.
3) Access n different blocks in all cache sets.
4) Access the blocks from 2) again and measure the misses.

Figure 1 shows the result of running this experiment with different values for n on the CPUs mentioned above. Note that all of those CPUs have 4096 cache sets. For the tree-based PLRU policy, we would expect to get 0 misses if n is smaller than the associativity, and 4096 misses otherwise, as is the case for the Core 2 Duo E6300. The goal of our work is to develop techniques to build a precise model of the policies used by the other two Core 2 Duo processors.

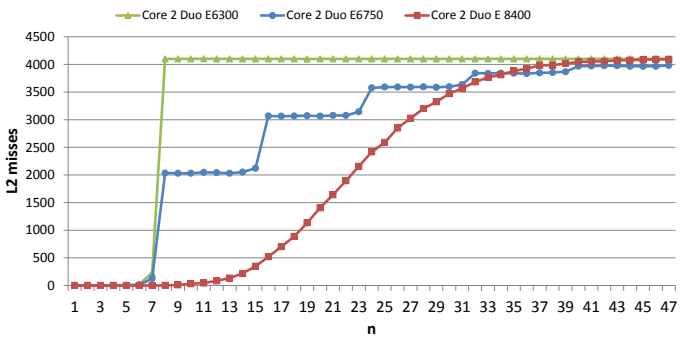

Fig. 1: Experimental analysis of the L2 cache behavior of the Intel Core 2 Duo E6300, E6750, and E8400.

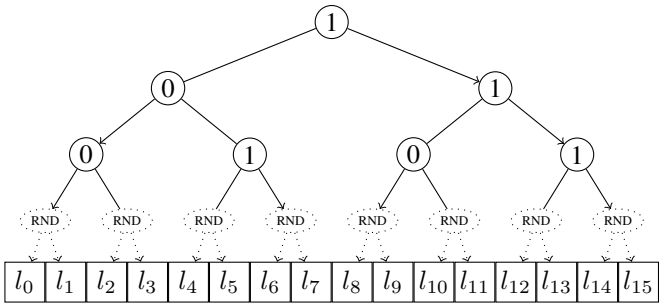

Fig. 2: PLRU-Rand state after an access to l_4.

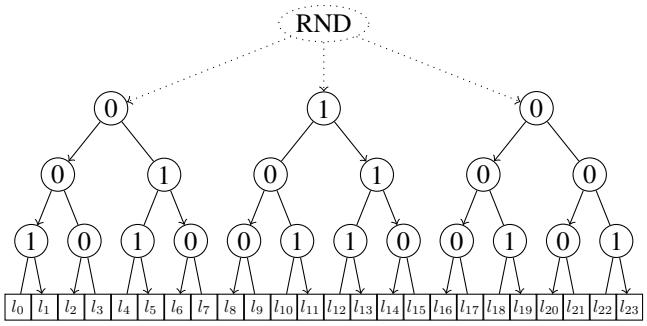

Fig. 3: Rand-PLRU state after an access to l_4.

III. Methodology & Challenges

We developed several microbenchmarks to analyze specific properties of replacement policies, for example the effects of additional hits to elements in the cache, whether the replacement behaviors in different cache sets are independent of each other, and a test for pseudo-randomness.

Implementing these microbenchmarks was challenging in several aspects. As we consider the L2 cache, the access sequences had to be designed in a way such that all accesses lead to misses in the L1 cache and are thus passed to the L2 cache. Furthermore, we had to find ways to minimize the effect of performance enhancing techniques like out-of-order execution or non-blocking caches. Finally, we developed techniques to accurately measure the number of cache misses. Unlike previously described approaches, our techniques are able to analyze individual memory accesses.

IV. Results

A. Models

Based on the results from running the microbenchmarks, we built the following models.

For the Core 2 Duo E6750, the following model agrees with our observations: Consider a PLRU-like policy in which the lowest bits of the tree (i.e., the bits closest to the leaves) are replaced by (pseudo-)randomness. Figure 2 illustrates this policy. Under such a policy, one of the two elements to which the tree bits point is replaced with a probability of 50%. Furthermore, after every eight subsequent misses the tree bits point to the same subtree. So the probability that an element is replaced after n subsequent cache misses can be determined by the following function: $P(n) = 1 - \left(\frac{1}{2}\right)^{\lfloor \frac{n}{8} \rfloor}$. This corresponds well to the results from the experiment in Section II. In the following, we will call this policy PLRU-Rand.

The behavior of the Core 2 Duo E8400, on the other hand, can be described by the following model: Consider a PLRU-like policy in which the root node is replaced by (pseudo-)randomness, as illustrated in Figure 3. In this policy, the elements are separated into three groups with 8 elements each; within each group they are managed by a tree-based PLRU policy. Upon a miss, one of these groups is chosen randomly. For this policy, the probability that an element is replaced after n subsequent cache misses can be determined by the function $P(n) = \sum_{a=8}^{n} \left(\frac{1}{3}\right)^{a} \cdot \left(\frac{2}{3}\right)^{n-a} \cdot \binom{n}{a}$. In the following, we will call this policy Rand-PLRU.

B. Benchmarks

We have compared the performance of the discovered replacement policies to other popular policies on the PARSEC benchmark suite [5]. To this end, we have implemented these policies in the Cachegrind cache simulator.

We found that the performance of Rand-PLRU is on most benchmarks comparable to PLRU. In one case (the *blackscholes* benchmark on a 4MB cache), the miss ratio of Rand-PLRU was about 45% higher than PLRU, and in another case (the *vips* benchmark on a 6MB cache), the miss ratio of Rand-PLRU was close to a third of the miss ratio of PLRU, but about three times the miss ratio of LRU. However, in both cases, the absolute values of the miss ratios are rather low, which means that the impact on the overall performance of an application would be rather small.

For PLRU-Rand, on the other hand, we observed a large difference in the miss ratio on the *streamcluster* benchmark. On a simulated 4MB cache, the miss ratio for PLRU is about 10% higher, and on a 6MB cache it is almost 50% higher. Moreover, as the absolute values of the miss ratio and the L2 access rate are also very high for this benchmark, this would lead to a significantly higher overall execution time.

V. Future Work

Arriving at a model of a replacement policy by using the microbenchmarks we presented requires some manual work. We are currently exploring the use of techniques from machine learning for building such models automatically. Furthermore, we plan to extend our work to other architectural features, such as translation lookaside buffers, branch predictors, and prefetchers.

References

[1] R. Wilhelm *et al.*, "The worst-case execution time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, 2008.

[2] D. Grund and J. Reineke, "Toward precise PLRU cache analyis," in *Proceedings of 10th International Workshop on Worst-Case Execution Time (WCET) Analysis*, July 2010, pp. 28–39.

[3] A. Abel and J. Reineke, "Measurement-based modeling of the cache replacement policy," in *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2013, pp. 65–74.

[4] R. Singhal, Personal communication, Intel, August 2012.

[5] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.