RETROSPECTIVE: Adaptive Insertion Policies for High-Performance Caching

Moinuddin Qureshi[⊕], Aamer Jaleel[†], Yale Patt[‡], Simon C. Steely Jr.[§], Joel Emer[†*]
[⊕]Georgia Tech [†]Nvidia [‡]The University of Texas at Austin [§]Intel [*]MIT

This ISCA-2007 paper was the culmination of more than a year of research by two teams, initially performed independently within the VSSAD advanced R&D group at Intel and the University of Texas at Austin, and then later done in collaboration. One of the hallmarks of this paper is the simplicity of the proposed solution, however, that belies the several dozens of ideas (and millions of simulations) that were explored and discarded in search of a simpler and more effective solution. In this retrospective, we hope to share the context and backdrop in which this work was done, the core contributions, and the impact and legacy of this paper.

I. The Backdrop

During this work, Moin Qureshi was a graduate student at UT Austin, working under the advisement of Prof. Yale Patt. His thesis work was on high-performance caching, and some of his work prior to this paper included MLP-Aware Cache Replacement (which introduced the concept of dynamic set sampling), and Utility-Based Cache Partitioning (UCP, which used dynamic set-sampling to learn utility curves). As part of his graduate work, he was investigating improved cache management for L2 caches, especially the problem of lines getting evicted without any reuse. In Summer 2006, he was an intern at Intel Portland, looking at L2 replacement policies.

In 2005, during his summer internship with VSSAD, Aamer Jaleel had developed a Pin-based workload characterization tool, called CMP$im (resulting in an HPCA-2006 paper, which looked at cache behavior of bio-informatics workloads). In subsequent years, CMP$im would go on to become the workhorse for developing state-of-the-art cache management schemes. Aamer graduated in 2005 and joined VSSAD full-time. As part of the *Dalton* cache research group meetings, Aamer (along with Simon Steely and Joel Emer) was using CMP$im to study cache reuse behavior and noted poor reuse at the LLC cache. Their team had started to look at solutions to reduce cache miss rates by exploiting variation in reuse.

The discussion and collaboration between the two teams started during Moin's 2006 summer internship at Intel (Portland) when it was clear that both teams were independently working on similar problems. While the solutions were still not clear during the summer, an interesting event that happened during this internship was a presentation from us to the Intel Nehalem design team. Our presentation was with the message that some applications have low locality, and for such applications even random can beat LRU, so why not have two policies (LRU and Random) and select the policy dynamically based on set-sampling (having an extra tag directory that simulated Random for a few sampled sets). While the idea of adaptive replacement was received well, Andy Glew had a strong criticism that adaptive replacement should be for free (without any auxiliary tags) as extra tag structures require verification, testing, and floorplanning and such changes are not appealing for commercial adoption. We had argued, vociferously and naively at that time, that such a zero-overhead solution would be virtually impossible, as we need some extra state for doing the selection.

II. The Problem

The basic problem solved by this paper is about cache replacement policies. However, the context in which we looked at the problem was new. Until then, much of the work on cache replacement (for almost 25 years) had looked only at L1 caches, and the consensus at that time was that LRU was pretty good (or good enough), such that new cache replacement policies were not warranted. However, cache sizes were increasing and multi-level cache hierarchies had become common by the early 2000s. Furthermore, cache replacement policies were similar for L1 and L2.

Our work noted that there is a huge locality difference between L1 access stream and L2 (LLC) access stream, as the L1 caches filter away the temporal locality. If a line has high locality, it stays in L1 and offers lots of hits, but these hits do not count toward L2 hit-rate. Furthermore, spatial patterns (such as reading consecutive words) result in high L1 hit rates, but do not contribute to L2 hits, as the L2 linesize tends to be identical to L1, so spatial hits are not visible to L1. This filtering of locality by L1 results in poor reuse of L2 lines.

We defined a term *Dead on Arrival (DoA)* lines as the lines that are brought into the L2 and evicted without being accessed again. We showed that, with an LRU policy, an overwhelming majority (almost 70%) of the lines installed in the L2 cache are DoA lines. Thus, much of the L2 space is used for lines that do not ever contribute to cache hits. By analyzing the code for memory-intensive workloads (such as mcf, art, and health), we showed that the main cause of DoA lines is that the working set of frequently used lines is larger than the cache size, which causes thrashing and poor reuse.

The goal of our paper was to develop replacement policies that are tailored to exploit the filtered locality at the L2 cache and to have a simple and practical solution that can be adopted in commercial implementation.

III. The Contributions

There were several cache replacement policies (exploiting line-level reuse, PC-based, frequency-based, adaptivity between LRU and Random, lookahead-based, or genetic learning based) explored during the course of this work. However, most of these solutions were either too complex and/or storage-intensive, or the performance benefit was limited. By some of our rough estimates at that time, there were more than a million simulation runs performed in the course of this study.

Key Contribution-1: Insertion Policy: The key insight in this work was that when the working set is greater than the cache size, simply preserving some part of the working set is sufficient to improve the cache hit rate (the effective hit-rate is quite close to optimal for a circular reference pattern). Preservation of the working set can be provided by simply changing the place where the incoming line is placed (note that the conventional LRU replacement policy places the line at the MRU position). The first main contribution of the work was introducing the notion of *insertion policy* as a means to improve cache performance. Two policies, LRU Insertion Policy (LIP, which places incoming lines to LRU position) and Bimodal Insertion Policy (BIP, which places almost all of the incoming lines in LRU but randomly places a small fraction of lines in MRU position). Both LIP and BIP are thrashing resistant, with BIP more adaptable to a changing working set.

Key Contribution-2: Set Dueling: Our initial hunch was to use dynamic set sampling (using a separate auxiliary tag directory) to choose between BIP and LRU. However, this would suffer from the same implementation constraints that Andy Glew had mentioned (the extra directory needs to be designed, verified, tested, and incorporated in the floorplan). The key insight that enabled the virtually zero overhead solution was to let the sets *duel* for the two policies, on a small number of sampled sets, and use that winner as the policy for the remaining follower sets. This scheme requires just a single saturating counter (two bytes) to track which of the two policies incurs more misses and the most-significant bit (MSB) indicates the winner. While cache miss rate varies across sets, even for the same policy, we showed (with both an analytical model and experimentation) that once we have 32-64 sampled sets, chosen randomly, then the scheme is able to identify the stronger policy, as long as there is a non-negligible difference in miss rate between the two policies (the cases where the difference is negligible is moot for practical scenarios as both policies would give similar performance). Simon Steely is responsible for the cool name of *Set Dueling*, much better than the rather boring term *In-Cache Dynamic Set Sampling (IDSS)* we had in the submitted version.

Our proposed design of *Dynamic Insertion Policy (DIP)* used Set Dueling to choose between BIP and LRU. Our results showed that DIP significantly bridged the gap between LRU and OPT. As BIP requires negligible changes (simply skip the update to MRU for the incoming line), and Set Dueling required a single counter, the simplicity of DIP made it appealing for commercial adoption.

IV. The Initial Reception and Impact

When the results of DIP first arrived, we knew right away that these ideas would get adopted in industry and academia. The ideas started to influence industrial designs within a few months of publication. At Intel, Aamer shared the ideas with the Ivy Bridge design team and evaluated DIP directly within the Ivy Bridge performance model. There were some challenges along the way, specifically associated with studying a dynamic policy in a detailed product level performance model that only studied a small snippet of total application execution. Eventually, Ivy Bridge adopted the "Adaptive Fill Policy" [2] and academics reverse engineered Ivy Bridge's cache to identify "leader" sets and "follower" sets [5].

Moin graduated and joined IBM Research in Aug 2007. He soon became aware of a problem in upcoming Power-7 processors, which had private 4MB L3 per core but no means to share the aggregated 32MB capacity across the 8 cores. He suggested the use of Set Dueling to learn the sharing decision at runtime (which cores should spill and which should receive). Set-Dueling based capacity sharing was immediately adopted in Power-7 processors [1]. A variation of this scheme, *Adaptive Spill Receive (ASR)* [4], was published at HPCA-2009.

The DIP proposal converges on a single binary decision (BIP or LRU) globally for the entire cache. Greater performance can be obtained if each core/thread could have an independent binary decision (BIP or LRU). During Fall 2007, the Intel team, lead by Aamer, investigated such a generalization of DIP for multi-core processors and developed *Thread Aware DIP (TADIP)* [3]. TADIP required just one counter per core and performed similar to or better than UCP.

V. The Legacy

While DIP and variations found adoption immediately, future research in cache replacement (such as RRIP, SHiP, etc.) developed even more effective schemes, albeit at higher hardware overheads. Set Dueling continues to be used in both academic and industrial designs for dynamic cache decisions. Owing to the simplicity, the umbrella of solutions DIP, TADIP, and ASR are often included in graduate architecture courses to teach cache management. As of now, the paper has 942 citations (second highest of all ISCA-2007 papers).

One of the main learning from this paper was that simplicity is possible, however, it takes a lot of effort and understanding to develop a simple and effective solution. This journey is worth it though, as simple and effective ideas tend to move the needle, get adopted, and stand the test of time.

References

[1] "Under the hood: Of power7 processor." [URL]: http://gibsonnet.net/aix/ibm/systems_power_software_i_perfmgmt_underthehood.pdf

[2] S. Jahagirdar *et al.*, "Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge," in *HC-2012*.

[3] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer, "Adaptive insertion policies for managing shared caches," in *PACT-2008*.

[4] M. K. Qureshi, "Adaptive spill-receive for robust high-performance caching in cmps," in *HPCA-2009*.

[5] H. Wong, "Intel Ivy Bridge cache replacement policy," Jan 2013. [URL]: http://blog.stuffedcow.net/2013/01/ivb-cache-replacement/