Dead Page and Dead Block Predictors: Cleaning TLBs and Caches Together

Chandrashis Mazumdar* Prachatos Mitra* Arkaprava Basu
Department of Computer Science and Automation
Indian Institute of Science
{chandrashism, prachatosm, arkapravab}@iisc.ac.in

***Abstract*—The last level TLB (LLT) and the last level cache (LLC) play a critical role in the overall performance of memory-intensive applications. While management of LLC content has received significant attention, the same may not be true for LLT.**

In this work, we first explore the well-known concept of dead blocks in caches for TLBs. We find that *dead pages* are fairly common in the LLT. Different from dead blocks in LLCs, dead pages in LLTs are most often dead-on-arrival, i.e., they produce zero hits in the TLB. We design a storage-efficient dead page predictor that works with a fraction of storage compared to typical dead block predictors. This is important since an LLT itself requires only a few KBs of storage compared to MBs in LLC. We then leverage the dead page information to guide a simple dead block predictor in LLC. This is driven by the observation that dead blocks are often concentrated within dead pages. In effect, we designed a dead page predictor and a correlating dead block predictor with a total storage overhead of only 11KB to bypass predicted dead pages and dead blocks in LLTs and LLCs, respectively. Together, these predictors help improve the IPC of a set of 14 memory-intensive workloads by 8.3%, on average.

***Index Terms*—TLB; virtual memory; last-level cache; replacement algorithm**

I. Introduction

A miss in the last level TLB (LLT) and/or in the last level cache (LLC) is slow. A miss in the last level TLB can require up to four memory accesses to *walk* the page table to find the missing address translation. A miss in LLC requires an off-chip memory access. Since misses in LLT and in LLC are expensive and cannot be hidden through memory-level parallelism of even large out-of-order cores, often, the performance of memory-intensive applications depends upon the frequency of LLT and/or LLC misses.

While the management and replacement strategies of LLC have received significant attention over the decades [1]–[6], the same cannot be said about the replacement strategies in LLT. The work on reducing misses in LLT has been primarily focused on using large pages that allow a single entry in LLT to map a larger contiguous amount of memory. However, large pages are not a panacea for reducing LLT misses. Operating systems and/or application modifications are necessary for deploying large pages. If the physical memory is fragmented, it may not be possible to use large pages. Since the availability of large pages is not guaranteed, the use of large pages can lead to significant run-to-run variations. The requirement of contiguous virtual address allocation and alignment to a large page boundary means that not all addresses can be mapped using large pages. Consequently, many commercially popular applications advise against the use of large pages [7]–[9]. We, thus, explore a complementary approach in this work to improve the management of LLT contents to improve its value to application performance.

In this context, we turn our attention to the well-explored concept of dead blocks in LLC. A dead block is an entry in a cache that will not experience further hits until its eviction. The entry is dead because it serves no purpose while occupying space in the cache. There has been a large body of work on predicting dead blocks in caches [1], [2], [10]–[17]. A dead block can be prioritized for replacement [1], [2], [10], [12]–[14] or bypassed [11], [15], [16]. This improves utilization of the cache capacity and reduces the miss rate.

An unexplored question is whether LLT can also harbor many dead entries? Our first contribution in this work is to characterize dead entries in the LLT empirically. We call a dead entry in TLB a *dead page*. We find that for many applications, 82% of LLT entries can be dead at any given point in time. We find that a large fraction of these dead pages – about 86% of them, on average, are *dead on arrival* or DOA pages. A DOA page experiences *no further hit* during its stay in the LLT after it is used for the demand request that brought it. Therefore, LLT misses would not increase if a DOA page was not allocated (i.e., bypassed). Bypassing a true DOA page can, however, avoid replacing an otherwise useful entry from LLT. This helps in reducing misses and in increasing the effectiveness of LLT.

Given the prevalence of DOA pages in LLT, we design a DOA page predictor for LLT. A predictor for LLT is more sensitive to additional storage than a predictor for LLC. The storage of a typical 1024-entry L2 TLB (LLT) would be $\sim$ 11.75KB. Compare this with multi-MB LLCs. Consequently, a practical predictor design for LLT can afford at most 1-2KB of storage. Thus, a key challenge is how to design an accurate and yet low-overhead DOA predictor for LLT.

Our DOA (dead) page predictor, called dpPred, uses a novel two-dimensional history table for better storage-efficiency. The table is indexed with a hash of the program counter (PC) in one of the dimensions while a hash of the virtual page number (VPN) is used for the other dimension. The PC of the instruction that brought an entry in the LLT is not immediately

*Authors contributed equally.

2378-203X/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00050

available during the entry's eviction. Thus, the hash of that PC needs to be stored in each LLT entry for updating the predictor's history table on eviction from LLT. On the other hand, a TLB entry already contains VPN; no additional storage is required. For example, in our default design, the predictor keeps a small 6-bit hash of PC of the instruction that brought an entry to the LLT. The 1024-entry history table is then indexed using the hash of PC and a 4-bit hash of the VPN. We empirically found that while PC is important for the predictor's accuracy, a DOA page predictor using a combination of PC and VPN achieves an accuracy similar to one using only PC, but with less storage.

The history table entries contain 3-bit saturating counters that act as confidence predictors. A counter is incremented on detecting a true DOA page. At the time of an LLT fill (i.e., on completion of a page walk), a DOA page is predicted if the counter value in the history table is above a threshold. A predicted DOA page is not placed in the LLT (bypassed).

We keep a small (e.g., two entries) *shadow table* that keeps recently bypassed entries and also acts as a victim buffer. A hit in the shadow table indicates misprediction. We use this as negative feedback and reset the predictor's entries corresponding to the given VPN. This helps in maintaining higher accuracy (on average, accuracy is 83.6%).

We then observe that LLT and LLC accesses are not entirely independent [18]. In hindsight, it is natural since address translation precedes data access. We empirically found that LLCs often contain more than 50% DOA blocks. While this is lower than that in LLT, it is still significant. Importantly, we found that DOA blocks are much more likely to be a part of a DOA page than a non-DOA page. Specifically, more than 70% of the DOA blocks are concentrated only on DOA pages.

We design a low-overhead DOA block predictor for LLC that leverages DOA page information. We name it the correlating dead block predictor (cbPred). The key idea is to use DOA page information as a filter for DOA block prediction. It enables two advantages. ① It helps cbPred achieve high accuracy (often $\geq$ 99%) as it attempts to predict DOA *only* for blocks that map onto DOA pages. We empirically found that a block belonging to a DOA page is more likely to be a DOA block than a block from a non-DOA page. ② Since we update the predictor for only a subset of blocks, even a relatively small history table ensures limited aliasing. Further, since cbPred uses output from a PC-directed dpPred for filtering, there is no need to use PC in cbPred itself. Therefore, LLC entries do not store (hash of) PC. Consequently, cbPred uses 6×-11× less storage compared to previous dead-block predictors. Like in LLT, on a predicted DOA block, we do not allocate it in the LLC (bypass).

dpPred and cbPred, working in tandem, improves IPC of 14 memory-intensive applications by 8.3%, on average (geometric mean). The LLT and LLC MPKI reduce by 9.65% and 4.24%, on average. Importantly, the cumulative state overhead of both the predictors is just around 11KB. Compare this to typical LLC dead block predictors that incur state overhead of 60-120 KB just for the LLC [2], [10].

In summary, we make the following contributions.

- We quantify and characterize the presence of dead entries (a.k.a. dead pages) in LLT.
- We design a storage-efficient dead page predictor for the LLT that uses novel concepts like mixed indexing using PC and address to keep storage overhead low.
- We then leverage dead page information of LLT to design a correlating dead block predictor in LLC.

II. Background on dead block predictors

Here, we discuss previous works on dead block predictors (DBPs) for caches. Since dead blocks, by definition, cannot produce hits in the cache, DBPs attempt to identify such blocks to bypass them (e.g., [11], [15], [16]) or prioritize them for victimization (e.g., [2], [10], [12]–[14]) or prefetch more useful blocks into the identified entries (e.g., [1]). This way, DBPs can help improve the usefulness of a cache.

DBPs differ based on different metrics and/or signatures used to identify dead blocks. For example, Lai *et al.* proposed to use a trace of instructions accessing a block to predict when a block becomes dead [1]. Alternatively, researchers have proposed using the number of cycles or the number of references for which a block is typically alive in a cache for predicting dead entries [19], [20]. Kharbutli *et al.* proposed a counter-based predictor that uses the number of accesses to a block or the number of accesses to a set between two accesses to a block (i.e., access interval) as the trigger for declaring when a block becomes dead [13].

Liu *et al.* proposed using cache bursts, i.e., a contiguous stream of accesses to a given block, for predicting dead blocks [17]. SHiP is a signature-based dead-block predictor that uses hit counts whereby LLC blocks are predicted to have either a distant or an intermediate re-reference interval [2]. More recently, Faldu *et al.* proposed to use live distance, i.e., stack distance between allocation and eviction of a block, to help predict dead entries [14].

Since the storage requirement of a DBP is an important design consideration, Khan *et al.* proposed to track on a few sets of the cache and use a shadow tag structure for training the predictor [11]. Further, software support for profiling or compiler annotation to help predict dead blocks has also been explored in the past [21], [22].

While many researchers have explored dead block predictors for both L1 caches and LLC, there has been little work on dead pages. This work aims to fill that gap. In the process, we find that the nature of dead entries in LLT and LLC are different (detailed in Section IV-C), and thus, dead block predictors for caches are not immediately applicable to TLBs.

III. Experimental Methodology

Before discussing the analysis of dead pages and blocks, it is imperative to detail our experimental methodology. We performed experiments using the Sniper simulator. Sniper is a Pin-based x86 simulator in the Graphite framework [23]. We enhanced the simulator to emulate a realistic page table walk. Specifically, we allocate a four-level radix tree data structure

TABLE I: Parameters for experiments

CPU	2.66 GHz, OoO core
L1 D-TLB	64 entries, 4-way, 1 cycle.
L1 I-TLB	128 entries, 4-way, 1 cycle.
L2 TLB	1024 entries, 8-way, 8 cycles.
Page Walk Cache	3 levels, fully associative, Entries: 4 (L1), 8 (L2), 16 (L3), Lat. (cycles): 1 (L1), 1 (L2), 2 (L3)
L1 D-Cache	32KB, 8-way, 5 cycles.
L2 Cache	256KB, 8-way, 11 cycles.
L3 Cache	2MB per core, 16-way, 40 cycles, inclusive.
Main Memory	191 cycles.

TABLE II: Workloads

Workload	Description	Mem. footprint
cactusADM	Benchmark from SPEC 2006	780 MB
cc	Connected Components benchmark from GAPBS	680 MB
cg.B	Conjugate Gradient algorithm from NAS Parallel Benchmarks	300 MB
sssp	Single-Source Shortest Path benchmark from GAPBS	900 MB
lbm	Benchmark from SPEC 2017	450 MB
Triangle	Triangle counting from Ligra benchmark suite	450 MB
KCore	K-core decomposition from Ligra benchmark suite	450 MB
canneal	Routing cost optimization in chip design from PARSEC	350 MB
pr	PageRank from GAPBS	680 MB
graph500	Breadth-first-search and single-source-shortest path over undirected graphs	400 MB
bfs	Breadth-First Search from Ligra benchmark suite	450 MB
bc	Betweenness Centrality benchmark from GAPBS	680 MB
mis	Maximal Independent Set benchmark from Ligra benchmark suite	450 MB
mcf	Minimum cost Network Flow benchmark from SPEC 2006	450 MB

as the page table. The page table contents are cached on the processor caches as in the real hardware. A walk triggered by an LLT miss may need up to four memory accesses. Like real hardware, we use page walk caches (PWCs) to cache partial translations to reduce the number of accesses on a page walk to 1 to 3 memory accesses (on a hit to PWC). Therefore, the page walk latency is variable – it depends upon hits/misses to PWCs and whether the page table accesses hit in the data caches. When a page walk completes, it places the translation in both L1 and L2 TLB (LLT) in our design. Alternatively, it is possible to place the translation into L1 TLB only. An entry can then be placed in the LLT on its eviction from the L1. However, we did not find any significant performance difference between these two alternative designs.

Table I details the configuration of the simulated baseline processor. Table II lists the applications used in the evaluation. We chose these applications from various benchmark suites such as SPEC CPU 2006 [24], SPEC 2017 [25], Parsec [26], Ligra [26] and GAP benchmark suite [27]. We chose applications with relatively large memory footprint (listed in the table) and simulated entire applications.

TABLE III: Percentage of LLC DOA blocks that map on to a DOA page in LLT

Workload	LLC blocks (%)	Workload	LLC blocks (%)
cactusADM	72.22	canneal	64.15
cc	67.76	pr	33.33
cg.B	92.14	graph500	81.40
sssp	93.25	bfs	81.00
lbm	99.98	bc	62.38
Triangle	73.33	mis	62.23
KCore	68.18	mcf	66.18

IV. Quantifying dead pages in LLT and its relation with dead blocks in LLC

We first quantify *dead pages* in the TLB. We then explore the correlation between dead pages and dead blocks in LLC.

A. Dead pages in TLB

We focus on the LLT (here, L2 TLB) since an LLT miss triggers a slow page table walk that could affect performance of applications. In contrast, the latency of an L1 TLB miss that hits in the L2 TLB is often hidden by out-of-order cores. The total heights of stacked-bars in Figure 1 show the estimated fraction of dead LLT entries at a given time, on average. We estimate the fraction by sampling entries over time. Across all workloads, on average, 81.66% of LLT entries are dead at any given time. Let's momentarily ignore stacks in the bars.

To better characterize dead pages, we classify LLT entries into ① dead-on-arrival (DOA), ② mostly dead (dead time $>$ live time) but experienced at least one TLB hit, and ③ active/mostly live (dead time $<$ live time). We classify entries at the time of their eviction from LLT, unlike the sampling of entries in Figure 1. DOA entries produce zero hits during their stay in LLT after servicing the demand request. The mostly dead entries spend the majority of their time in TLB as a dead entry. Only the last category of entries is truly valuable to retain in LLT.

Figure 2 quantifies and categorizes dead pages at the time of eviction. The total height of each stacked bar represents the percentage of all entries that had more dead time than live time at the time of their eviction from LLT. Each bar is then divided into DOA entries and mostly-dead entries. The lower stack represents DOA entries. First, as expected, a majority of the entries at the time of their eviction spent more time as a dead entry in the LLT. Further, on average, more than 85% of these entries are DOA. Figure 1 shows a similar breakdown of the DOA pages among the dead entries, but as a sampled view of LLT contents averaged across many snapshots. This measurement also shows the prevalence of DOAs. This leads to two key observations that guide the rest of the work. ① There are many dead entries in the LLT that can be leveraged to improve TLB performance. ② DOAs dominates among the dead entries, and thus, any technique to leverage dead pages in LLT should focus on DOA pages.

B. Correlation between dead pages and dead blocks

TLB and cache accesses are not entirely independent [18]. A TLB lookup precedes a cache lookup. Therefore, an interesting

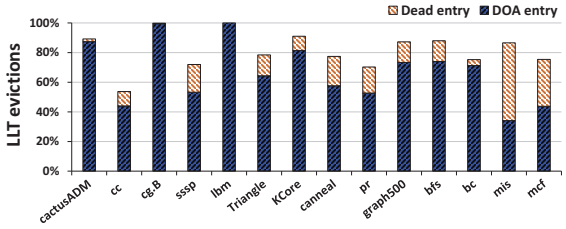


Fig. 1: Fraction of LLT entries dead or DOA at any time

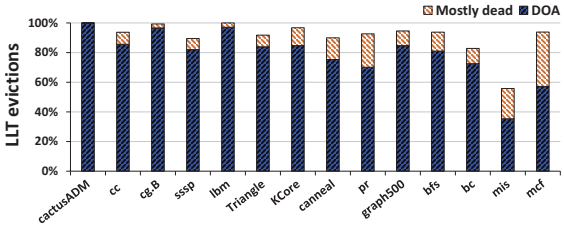


Fig. 2: Classification of dead pages in LLT

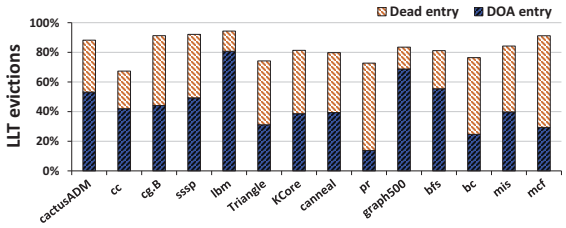


Fig. 3: Fraction of LLC entries dead or DOA at any time

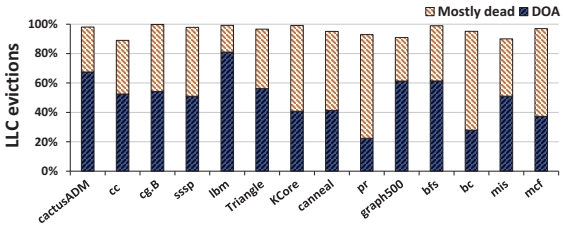


Fig. 4: Classification of dead blocks in LLC

question is whether there exists any correlation between the dead pages and the dead blocks in LLC. If yes, one could leverage the dead page information to identify dead blocks in the cache accurately. We focus on the correlation between the LLC and the LLT since LLC misses are costly.

Before we analyze the correlation between dead pages and blocks, we quantify the existence of dead blocks in the LLC. Figure 3 reports the fraction of entries (sampled) in the LLC that are typically dead and the DOA fraction at any given time. We observe that, on average, 83% blocks in the LLC can be dead at any point in time. This is in line with previous studies [11].

Figure 4 shows what fraction of blocks are DOA and what fraction are mostly-dead at the time of eviction. We additionally note that a significant fraction of dead entries is DOA, similar to a previous study [28].

The analysis in the previous subsection demonstrated that dead entries in LLT are predominantly DOA while here we find that LLC also contains a significant fraction of DOA blocks. Thus, an obvious question is if DOA blocks in LLC relate to DOA pages in LLT. If so, then the detection of a DOA in one structure can help predict DOA in the other structure. To quantify the opportunity for such an optimization, we measure the likelihood of a DOA block in LLC being part of a DOA page in the LLT.

Table III shows the fraction of all DOA cache blocks that map on to a DOA page in the LLT. On average, we find that 72.7% of all DOA cache blocks fall on a DOA page, while only 27.3% of the DOA blocks fall on a non-DOA page. Therefore, a DOA block is more likely to be part of a DOA page. We will later show how we leverage this propensity for building a low-overhead but accurate ($> 98\%$ accuracy) DOA block predictor for LLC with the aid of a DOA page predictor.

C. Discussion

To appreciate the intuition behind the prevalence of DOAs, we must note that the LLTs/LLCs observe the access streams filtered by the upper levels (closer to the core) of TLBs/caches. Therefore, an LLT or an LLC entry does *not* witness *immediate* reuse after the entry/data is brought on demand. If the reuse distance of an entry, *after* its immediate reuse, is large enough, then LLTs/LLCs will fail to register any hit, resulting in a DOA. We found that 50.4% of all LLC blocks are DOA, on average. This is similar to numbers reported by others [28].

However, LLTs have a larger fraction of DOA entries than LLC (78.9% of LLT entries are DOA). This is primarily because an LLC hosts orders of magnitude more entries (blocks) compared to the number of entries in an LLT. Consequently, the length of the average stay of an entry in LLT is $4\times$-$5\times$ shorter than an LLC entry. This increases the likelihood of DOA entries in LLT since the reuse distance needs to be shorter to produce a hit in LLT.

A curious reader could ask if the dead block predictors designed for LLC could work well for LLT too? While in Section VI-A we quantitatively show how LLC dead block predictors can be ineffective for LLT, here we present a high-level intuition on why so. The dead block predictors have primarily focused on non-DOA entries, which are more common in LLC [1], [13]. However, LLTs have much more DOAs. Thus, those predictors do not work well for LLT. Secondly, compared to a cache block entry, a TLB entry size is much smaller. For example, a typical cache block size, including tag and data, would be around 70 bytes, while a TLB entry is around 12 bytes. Therefore, the amount of metadata that can be reasonably added to TLB entries for prediction is far limited compared to a cache block.

One can also ask whether dead pages can be predicted using dead block information. This is hard for two reasons. First, since many cache blocks (e.g., 64) map onto a single page, an entry in TLB is *likely* to be dead only if a substantial percentage of cache blocks belonging to the page is dead (often $>$70%). This would necessitate tracking all predicted dead blocks that map onto a given page to help predict a dead page. Second, as mentioned earlier, an entry typically stays much longer in LLC than an entry in LLT. It is not uncommon

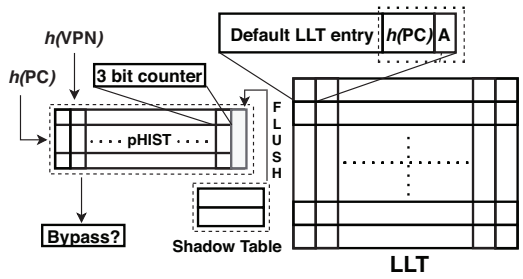


Fig. 5: Design of LLT's dead page predictor (dpPred)

that an LLC block may not be dead, but the page that it maps is dead in the TLB. Therefore, one would often fail to detect a dead page if guided only by dead blocks.

V. Design and implementation of predictors

We describe the dead page predictor for LLT and then the LLC dead block predictor that leverages the knowledge of dead pages.

A. Dead page predictor

We focus on building a storage-efficient DOA predictor for LLT since the analysis in Section IV demonstrated that majority of dead pages are DOA. We call this the dpPred or Dead-Page Predictor. When dpPred predicts a DOA, we do not allocate the corresponding entry in the LLT (i.e., bypass) to avoid replacing useful entries.

Figure 5 depicts a high-level diagram of the proposed design. To help identify a true DOA page, we first add a single bit (Accessed) to each entry. This bit is unset by default and is set only on a TLB hit to that entry. A DOA page is identified if the Accessed bit is unset upon the eviction of an entry. This information is then used to update the predictor's history table (described shortly) accordingly.

Each TLB entry also stores a hash of the program counter (6 bits long by default) of the memory instruction that brought the entry in the TLB. The hash is computed by dividing the PC into subblocks and XOR-ing them.

Next, we keep a simple *two-dimensional* direct-mapped table of 3-bit saturating counters as the history table (default, 1024 entries), as shown in Figure 5. We call this table the page history table or pHIST. This history table is indexed by the hash of PC in one dimension and the hash of VPN (here, 4 bits) for the other dimension (Figure 5). We use VPN alongside PC, to avoid storing many bits in each TLB entry. Since VPN is naturally available during evictions and lookups, no extra storage overhead is required. We empirically found that the performance achieved using a mix of PC and VPN is similar to solely using PC (Section VI). However, the former (proposed design) reduces storage overheads on the TLB.

The pHIST is looked up during the eviction of a TLB entry to increment the corresponding saturating counter based on whether an entry is a DOA, and during TLB fill to predict if the incoming translation is a DOA.

Finally, we introduced a small shadow table to correct mispredictions and also work as a victim buffer. The shadow table stores the VPN of recently predicted DOAs, along with the corresponding translation (2 entries by default). A match in the shadow table on a TLB miss signifies a mispredicted DOA. The column of entries corresponding to the (hash of) given VPN is flushed from the pHIST on such incidents to forget the mispredicted DOA (right upper side of Figure 5). In short, the shadow table provides negative feedback to the predictor on mispredictions to help improve accuracy.

Operation: Now that we have described the key components of the predictor and its design philosophy, we summarize steps taken by the predictor on LLT lookups, during LLT fill, and during the eviction of an entry from the LLT.

Figure 6a shows the flowchart of operation on an LLT lookup. On a hit to an LLT entry, the Accessed bit is set. On a miss, however, the shadow table is looked up to ascertain previous mispredictions. On a match in the shadow table, the translation is returned from the shadow table's victim entry. The translation is also placed in the LLT, and the shadow table entry itself is removed. Finally, to provide negative feedback to the predictor, entries (column) corresponding to the hash of the VPN are cleared from the pHIST table.

On an LLT miss, before sending the request downstream, the hash of the PC that triggered the miss is stored in the LLT's MSHR. This avoids the need to attach the PC to the page walk request. When the walk finishes (i.e., during the LLT fill), the hash of the PC will be needed for prediction.

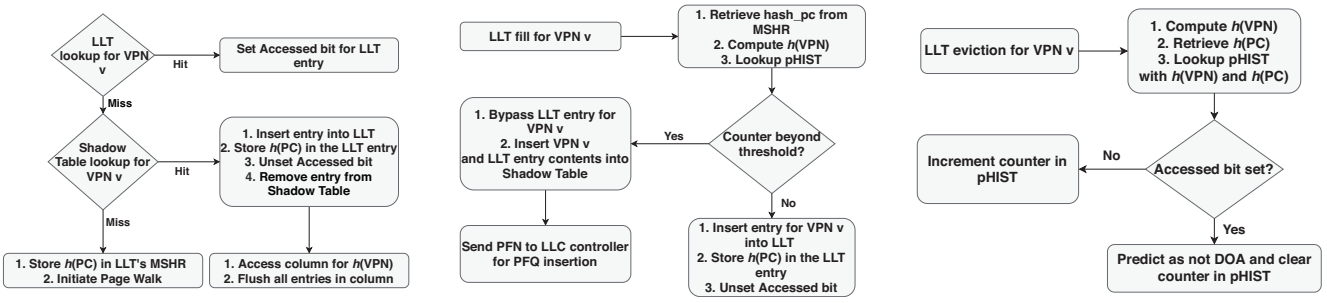
Figure 6b shows the operation on an LLT fill, i.e., when a page walk request completes for an earlier LLT miss. On an LLT fill, the pHIST table is looked up using the combination of the hash of the PC (from the MSHR) and the hash of the VPN. If the counter value in the history table entry is more than a threshold value (here, 6 by default), the VPN is predicted to be a DOA. Consequently, the translation is not placed in the LLT and instead placed in the shadow table's victim entry. Otherwise, the usual allocation in LLT follows.

Figure 6c shows the process of updating the predictor's history table on eviction of an LLT entry. We compute the hash of the VPN and couple it with the hash of the PC stored in the LLT entry to look up the two-dimensional pHIST table. If the Accessed bit in the LLT entry is set, then it was not a DOA, and thus the counter in the pHIST table entry is cleared. Otherwise, the counter is incremented by one.

B. Correlating dead block predictor

The analysis in Section IV demonstrated that ① typically there are significant number of DOA blocks in the LLC, and ② most of these DOA blocks fall on the DOA pages of the LLT (e.g., on average, 73% of the DOA blocks concentrated on DOA pages). We thus set out to design a low-storage correlating LLC DOA block predictor that utilizes information about DOA pages. We name this predictor the correlating dead block predictor or cbPred in short.

The key idea in cbPred is to update the DOA block predictor and attempt to predict DOA in LLC *only if* the block's address falls on a DOA page. This has two important benefits. First, it incurs very low storage overhead (∼10KB



(a) Predictor workflow on an LLT lookup (b) Predictor workflow on an LLT fill (c) Predictor workflow on an LLT eviction

Fig. 6: Operation of the LLT's dead page predictor (dpPred)

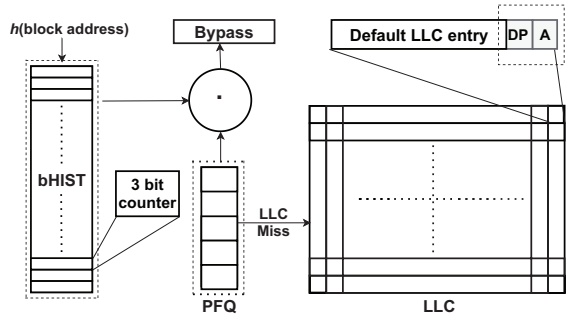


Fig. 7: Design of correlating dead block predictor (cbPred)

compared to 50-100s KBs for typical dead block predictors). Since the predictor is updated only by a limited subset of all cache block addresses, a relatively small number of entries in the history table can keep the number of collisions in the predictor's history table low. Further, unlike many dead block predictors (e.g., SHiP-PC) that store PC along with each cache block, there is no such requirement here. The PC is indirectly used to filter out the candidates for DOA block since the dpPred that guides cbPred, uses (hash of) PC. Second, the predictor becomes highly accurate, with prediction accuracy over 99% in almost all cases (Section VI). The screening of candidates using DOA pages ensures that the predictor attempts to predict only for blocks that are likely to be a DOA.

Figure 7 depicts the high-level components of cbPred. To realize the above idea, we propose to introduce a small structure to keep physical page numbers of recently predicted DOA pages at the LLC (at every slice in a banked LLC configuration). We call this structure the PFN filter queue or PFQ. We found that an 8-entry PFQ is sufficient since typical cache block accesses fall in recently accessed pages. Entries in PFQ are replaced in a simple FIFO order.

When the dpPred in the LLT predicts a DOA page, the corresponding PFN is sent to all LLC slices. In the worst case, such communication happens only as frequently as page walks. Further, only a fraction of those walks (TLB fill) would be predicted to be DOA. This communication is also off the critical path of execution. On receiving PFN corresponding to a DOA page, the LLC controller inserts the entry in the PFQ.

The update of LLC's dead block predictor (cbPred) and the prediction for a DOA block happens *only if* the block falls on a DOA page, i.e., PFQ has a matching entry. This is key to the predictor's high accuracy at a low storage overhead.

As shown in Figure 7, we add two bits to each cache block entry to identify blocks that belong to a predicted DOA page and identify a true DOA block. A dead page bit (DP) is set for cache blocks that map on to a predicted DOA page. The bit is unset by default and is set when a cache block is allocated in LLC whose PFN is in PFQ. Further, an Accessed bit captures if a cache block has produced a hit during its stay in the LLC. This bit helps identify true DOA blocks and is used for updating the predictor's history table.

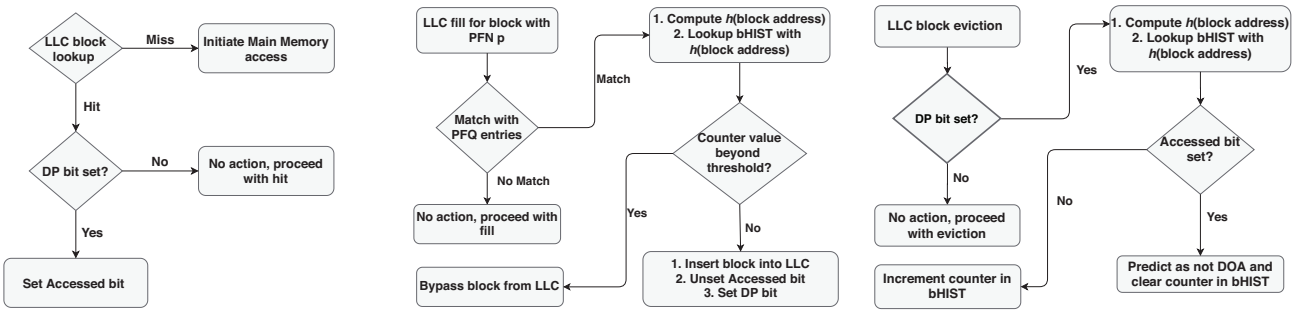
As typical to most predictors, we introduce a direct-mapped table with saturating counters, called the block history table or bHIST. In the default configuration, there are 4096 entries in the bHIST for a 2MB LLC. Counters in bHIST are updated only if a cache block being evicted has its DP bit set (i.e., the block maps onto a DOA page).

Operation: Now that we have described the basic philosophy behind our proposed DOA cache block predictor and its key hardware components, we detail important steps taken in the event of a TLB fill, eviction of an entry from LLC, on a lookup to LLC, and on an LLC fill.

On a TLB fill request, if the page is predicted DOA, the corresponding PFN is sent to all LLC slices (in a banked LLC). On receiving such a message, the LLC controller inserts the PFN in the PFQ.

Figure 8a shows the flowchart of operations on an LLC lookup. On an LLC hit, its DP bit is first checked. If the DP bit is set, then the Accessed bit is also set. On a miss, a main memory access is initiated as usual.

During an LLC fill (Figure 8b), the PFN of the incoming cache block is matched against all the entries in PFQ in parallel. No action is taken if there is *no* match. On a match, the cache block address is folded and XOR-ed to create a 12 bit hash to lookup the bHIST table. If the counter value in the corresponding entry in the bHIST table is more than a threshold (here, 6), then the incoming block is *not* allocated in the LLC (i.e., bypassed). If the counter value is below or equal to the threshold, then it signifies that while the block falls on a DOA page, the predictor is not yet confident to predict the block is a DOA block. In this case, we allocate the incoming block in the LLC, but we also set the DP bit.



(a) Predictor workflow on an LLC lookup (b) Predictor workflow on an LLC fill (c) Predictor workflow on an LLC eviction

Fig. 8: Operation of correlating dead block predictor in LLC (cbPred)

During eviction of an LLC block, the DP bit in the block is checked (Figure 8c). No action is taken if DP is unset. If the DP bit is set, but not the Accessed bit, then the block's address is hashed into bHIST table to increment the counter. If both DP and Accessed bits are set, it indicates that the cache block is not a DOA and the corresponding counter in bHIST table is cleared.

C. Impact on hit and miss latency

LLT hit latency is not impacted by dpPred. On a hit, the translation is returned to the L1 TLB before the Accessed bit is set. On an LLT miss, only the *2-entry* Shadow Table lookup happens in the critical path before page walk starts, which has a negligible impact on the miss latency. The Shadow Table reduces the number of walks by working as a victim buffer. The steps mentioned in Figure 6b during LLT fill are not in the critical path either. They are performed after the translation is returned to the L1 TLB.

LLC hit latency is not impacted by cbPred since the DP and the Accessed bits may be examined after returning the block to the higher-level cache. On an LLC miss, there is no action taken. On an LLC fill, similar to LLT, the block is returned to the L2 cache before the PFQ is looked up. Hence, LLC miss latency is unaffected by cbPred.

D. Storage overhead analysis

The total storage overhead due to both predictors is below 11KB. A detailed breakdown of the overhead is as follows.

Dead page predictor: The dpPred's storage overhead has three components – additional storage added to the LLT, the pHIST table and the shadow table. We added a total of 7 bits of metadata (6 bits hash of the PC and Accessed bit) to each TLB entry. For a 1024-entry TLB this adds 7Kbs or 896 bytes.

The pHIST table has 1024 entries, with each entry holding a 3-bit wide saturating counter. This adds another 3Kbs or 384 bytes of storage. Finally, the two-entry shadow table with each entry around 13 bytes long adds a total of 26 bytes. In total, the storage overhead is 1306 bytes.

This amounts to less than 11% storage overhead for a 1024-entry TLB. We assume the default TLB entry to be 94 bits long, and thus, the LLT would require about 11.75KB storage in the baseline. A TLB entry contains 29 bits VPN tag (assuming 48 bits virtual address), 39 bits PFN (assuming 51 bits physical address), 12 bits ASID, 4 bits MPK protection key [29], metadata bits including protection and supervisor/user.

Correlating dead block predictor: The correlating dead block predictor adds storage overhead due to metadata in the LLC, bHIST table, and the PFQ. We add two bits for each LLC entry. Considering a 2MB LLC, this adds to about 8KB of storage. A 4096 entry bHIST with 3-bit long entries adds 1.5KB of storage. Finally, an 8 entry PFQ with each of its entries holding 39 bits of PFN adds 39 bytes of storage. Therefore, the total overhead of the dead block predictor is around 9.54 KB or about 0.47% state overhead for a 2MB LLC. If we put both the dead page and dead block predictors together, then the total state overhead is around 10.81KB or 0.53% of the LLT and LLC storage budget.

VI. Results

We evaluate to answer the following questions. ① How much improvement does the dead page predictor (dpPred) provide? Could traditional dead block predictors applied to LLT achieve similar performance? ② How much further IPC improvement is possible with the correlating dead block predictor (cbPred), and how does it compare against traditional dead block predictors for LLC? ③ How does the storage overhead of each compare? ④ How sensitive are the dead page and correlating dead block predictors to various hardware configurations?

A. Dead page predictor

Figure 9 shows the normalized IPC with our DOA dead page predictor (dpPred). To demonstrate the need for dpPred we also report the IPC if AIP, and SHiP are applied to the LLT (denoted by AIP-TLB and SHiP-TLB, respectively). We use PC as the signature for SHiP and configure SHiP–TLB to use similar storage as dpPred, indexing with an 8-bit hash of the PC. Since the baseline replacement policy is LRU, we adapt SHiP to mark entries predicted to have distant re-reference as LRU. For AIP, since it needs 21 bits with every TLB entry, we use the default 256×256 two-dimensional history table.

Finally, to demonstrate the value that dpPred brings, we show the IPC if the L2 TLB was extended with slightly more

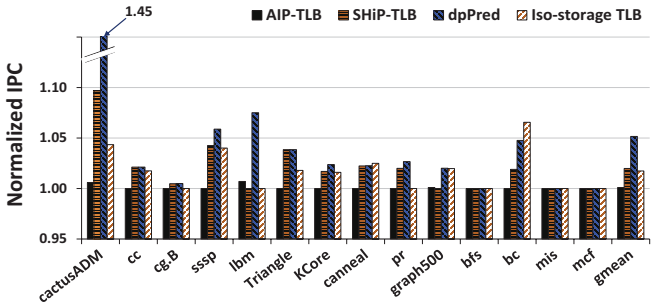


Fig. 9: Normalized IPC for TLB dead page predictors

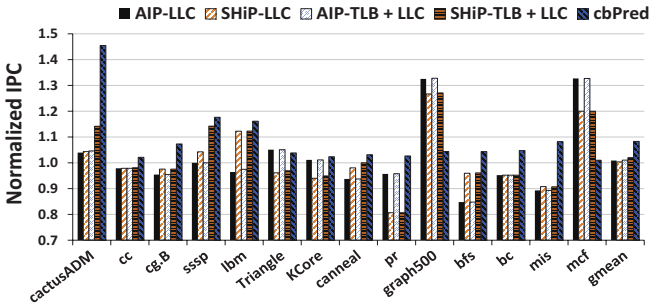


Fig. 10: Normalized IPC for LLC dead block predictors or LLC and TLB combined predictors

TABLE IV: LLT MPKI reductions by dead page predictors

Benchmark	AIP–TLB (%)	SHiP–TLB (%)	dpPred (%)	Iso–TLB (%)	Oracle (%)
cactusADM	0.6	7.3	37.8	2.8	55.2
cc	0.0	6.4	7.8	6.0	12.8
cg.B	0.0	8.0	16.0	0.0	18.3
sssp	0.0	6.8	9.4	6.0	32.1
lbm	1.0	0.0	30.2	0.0	46.5
Triangle	0.0	5.5	8.1	3.6	14.1
KCore	0.0	4.1	4.6	2.8	13.3
canneal	0.0	2.9	3.4	5.0	15.4
pr	0.0	4.3	4.4	0.0	15.2
graph500	0.2	1.3	3.8	3.5	18.5
bfs	0.0	0.0	0.0	0.0	10.0
bc	0.0	4.2	8.6	9.7	33.6
mis	0.0	0.0	0.0	0.0	16.7
mcf	0.0	0.0	1.0	0.0	9.0

TABLE V: LLC MPKI reductions by dead block predictors

Benchmark	AIP–LLC (%)	SHiP–LLC (%)	cbPred (%)
cactusADM	12.46	13.84	1.84
cc	-6.56	-6.56	-1.60
cg.B	-4.49	-2.63	5.90
sssp	0.19	14.29	17.82
lbm	-2.76	13.99	17.74
Triangle	7.15	-7.74	0.65
KCore	1.74	-8.82	-0.45
canneal	-15.54	-4.46	0.00
pr	-5.00	-21.45	-0.39
graph500	38.79	22.87	4.25
bfs	-22.35	-5.54	4.45
bc	-11.49	-11.38	-0.17
mis	-12.76	-10.67	7.45
mcf	23.59	16.00	1.81

amount of storage than dpPred's storage overhead for iso-storage comparison. Therefore, there are a total of four bars in the cluster for each application.

First, we observe that in eleven out of fourteen applications, dpPred improves IPC. The application cactusADM's performance improves the most, by about 45%. The average IPC improvement across all applications is about 5.2%. Dead block predictors like AIP, when applied to the LLT, provide almost no performance improvement. While SHiP does improve performance, the improvement is much smaller compared to dpPred. The reason SHiP-TLB was able to provide relatively better performance than AIP-TLB is that SHiP primarily predicts DOA while AIP focuses on non-DOAs. In Section IV we found that LLT's dead entries are dominated by DOAs.

Further, when we increased the number of ways in the LLT to account for extra storage of dpPred (iso-storage-TLB), the performance improved for several applications. However, our proposed dpPred provides much better IPC improvement.

Table IV shows the reduction in LLT MPKI under different predictors, with increased storage for baseline design, and for an *approximation* of an oracle dead-page predictor. The oracle could provide a sense of further scope of improvement. However, creating a true oracle predictor requires the full knowledge of the future. This is impractical to simulate due to gigantic size of memory access trace one would require. Instead, we *approximate* oracle by tracking if a true DOA entry replaced a non-DOA entry. This would effectively be an oracle predictor with a lookahead of 1 for each evicted entry.

We find that dpPred reduced MPKI by 9.65%, on average. Only the oracle betters it with an average MPKI reduction of 22.19%. The gap between dpPred and the oracle is due to two primary reasons. First, the same page that was a DOA during one of its stay in LLT may not be a DOA next time it is brought to LLT. Whether a page is DOA during its stay in LLT also depends upon other pages that are accessed during that time. Understandably, this impacts predictability of DOA without oracle knowledge. Second, not all DOA pages lead to the same number of misses. Since the oracle predictor has access to future information, it may identify additional DOA entries that reduce a higher number of misses in the future, while some correctly identified DOA entries by dpPred may not cause any additional future misses. This shows that while dpPred is more effective than other practical predictors, there is still scope for further improvement.

B. Correlating dead block predictor

Next, we present IPC improvement when we add the correlating dead block predictor (cbPred) on the LLC. Note that cbPred works *only* when it is coupled with dpPred.

Figure 10 shows the normalized IPC (over baseline) for various LLC dead block predictors and combinations of the dead page and dead block predictors. To put cbPred's performance in perspective, we also present IPC improvement due to AIP

and SHiP as well as when these two are applied to both L2 TLB and LLC together. First, we observe that cbPred, along with dpPred improved performance of all 14 applications. The average (geometric mean) IPC improvement is 8.3%. The performance improvement is significantly more than any other configuration we evaluated.

There are a couple of applications, graph500 and mcf where cbPred's improvements fall short of its peers. We note that both of these applications demonstrate considerable random access patterns that our predictors failed to leverage. Nonetheless cbPred, along with dpPred, still improves performance over the baseline. Importantly, they improve performance for *all* applications. The same is not true for its peers. For example, for the page rank (pr) application both AIP and SHiP suffered non-negligible performance loss. SHiP also suffered performance loss for mis and bc.

Table V shows the reduction in LLC MPKI due to AIP-LLC, SHiP-LLC and cbPred. We observe that six applications witness significant MPKI reductions due to cbPred, explaining the corresponding IPC uplifts. The reduction is most significant for sssp at about 18%. cbPred fails to reduce LLC misses for applications where there are not many DOAs in the LLC. Recall that cbPred attempts to predict only DOAs. However, where cbPred fails to reduce misses, it at least does not increase misses significantly. This is because cbPred attempts to predict DOAs for cache blocks that fall in dead pages and thus are likely to have DOA blocks (Section IV). The same is not true for other dead block predictors such as AIP and SHiP. For example, LLC misses increases significantly under SHiP for applications such as pr, bc, mis.

In summary, our proposed dpPred, along with cbPred, provides the most consistent and significant performance improvements and MPKI reductions in LLT and LLC.

C. Accuracy and coverage of predictors

We now report the accuracy and coverage of the proposed predictors and how they are affected by the design optimizations we made. Accuracy is defined as the fraction of correct predictions among all predictions made. Coverage is defined as the fraction of correct predictions over the total number of true (oracle) DOAs. In an ideal world, one would like both high accuracy and coverage.

Table VI shows the accuracy and coverage of dpPred and a constrained version of it called dpPred-SH. In dpPred-SH, the two-entry shadow table is disabled. Thus, the difference between dpPred and dpPred-SH shows the impact of the shadow table on the accuracy and coverage of the predictors.

First, we observe that dpPred predicts DOA pages with more than 80% accuracy and at least 50% coverage in more than half of the applications studied. However, for applications such as mcf, mis, canneal, Triangle, coverages are poor. Both mcf and mis have a relatively low fraction of DOAs among the dead pages (Figure 2) and consequently, dpPred fails to train enough to predict. In the case of canneal and Triangle, the statically set threshold on the saturating counter (default, 6) for prediction turns out to be too conservative. This is because the DOA VPNs are not repetitive enough for these applications due to their streaming like access patterns.

The accuracy of dpPred is high across all applications except for mcf and canneal where collisions in the history table and the lower predictability of the DOAs itself are to be blamed. In this context, we note the difference in accuracy and coverage between dpPred and dpPred-SH. The negative feedback provided by the shadow table helps improve accuracy at the cost of possibly lowering the coverage. This is visible in the case of mcf, and mis, where the accuracy improved by a large margin but the coverage reduced significantly. Better accuracy is important to ensure that no application suffers performance loss due to excessive (wrongful) bypassing caused by mispredicted DOAs.

To put our predictor's accuracy and coverage in perspective, we bring attention to SHiP-TLB's accuracy and coverage. Note that SHiP-TLB performed relatively better among the alternatives considered. We observe that our proposed dpPred provides significantly better accuracy across all the applications. For example, in applications such as cactusADM, Triangle or KCore, there is a large difference in accuracy enabled by dpPred compared to SHiP-TLB. While dpPred also provides better coverage compared to SHiP, it is not universally true. In cases like Triangle or graph500, the shadow table in dpPred significantly reduced the coverage in order to avoid mispredictions and wrong bypasses.

Next, we turn our attention to the correlating dead block predictor. Table VII shows the accuracy and coverage with cbPred, cbPred-PFQ and SHiP. cbPred-PFQ is the constrained version of cbPred that disallows the use of PFQ. The difference between cbPred and cbPred-PFQ shows how PFQ helps improve the accuracy.

First, we note the very high accuracy of cbPred for all applications – the predictor is accurate at least 98% of the time. The accuracy is consistently and significantly more than that of the alternatives such as SHiP-LLC. A key reason behind the high accuracy is the pre-filtering using knowledge of dead pages via the PFQ. For example, accuracy for Triangle improves from 84% to 100% when employing PFQ. All applications witness almost perfect accuracy with pre-filtering via PFQ. However, filtering of requests does reduce coverage as one would expect. But even with moderate to low coverage with very high accuracy, almost all of the bypasses for predicted DOAs are useful and are substantial in number. The benefit of highly accurate bypasses is visible in the performance improvements it enables (Figure 10).

SHiP-LLC provides comparatively better coverage than cbPred. However, in terms of accuracy cbPred still wins across all applications and by a significant margin. Please note that while SHiP-LLC is a specifically designed dead block predictor for LLC, cbPred makes use of already available dead page information to create a useful dead block predictor at a small fraction of its storage cost.

D. Storage overhead comparison

A key requirement for a dead page predictor is to be effective while being storage-efficient since TLBs themselves require only about 11.75KB of storage. This philosophy is then further extended to build a low-storage correlating dead block predictor. The total storage cost, including dpPred and cbPred, is 10.81KB (Section V-D), assuming a 1024-entry L2 TLB and a 2MB LLC. Compare this to the storage overheads of 124KB and 66KB for AIP and SHiP respectively, for the same LLT and LLC sizes. This shows that our proposed predictors, working together, can achieve better performance than many alternatives with 1/11th - 1/6th of the typical storage overhead.

E. Sensitivity analysis

We now explore how our proposed predictors behave in response to alteration of various configurations.

Figure 11a shows the impact of changing LLT size on the IPC improvement enabled by dpPred alone. As expected, IPC improvement is typically a bit muted when the LLT size is increased to 1536 entries due to less pressure on the TLB. However, cactusADM and lbm are exceptions. These applications thrash smaller LLTs, and the usefulness of entries increases with a larger LLT size. dpPred is then able to leverage the TLB better. In general, we see that dpPred remains useful across different sizes of LLT.

In Figure 11b, we explore the impact of the choice of the indexing function and the size of the pHIST table of dpPred. We first focus on a 1024-entry pHIST table that can be either indexed entirely by a 10-bit hash of PC or by a combination of PC and VPN (default). The advantage of the latter is the less storage overhead since unlike the PC, VPN is naturally part of TLB entries and adds no extra overheads. We observe that the performance remains almost the same irrespective of whether we use a hash of 10 bit PC or a combination of 6-bit PC hash and 4-bit VPN hash. This demonstrates that by combining PC and VPN in a novel two-dimensional history table design, we are able to lower the storage cost of the predictor without impairing performance.

When we double the pHIST table size and use one more bit in the hash of VPN, there is a slight improvement in performance. Overall, we demonstrate that the use of a 6-bit hash of PC, along with a hash of 4-bit VPN provides a good balance between performance and storage cost.

Figure 11c shows the impact of changing the shadow table size. A bigger shadow table reduces coverage while potentially increasing accuracy. Both high coverage and high accuracy are desirable for performance. We observe that increasing shadow table size from 2 to 4 degrades performance slightly due to a reduction in coverage. Therefore, we use 2-entry shadow table for the default configuration.

Next, we turn attention to the dead block predictor – cbPred. Figure 11d shows the impact on performance if the PFQ size is changed. We found that increasing PFQ size from 8 to 64 did not impact performance in any noticeable way. Therefore, we used 8-entry PFQ as default.

TABLE VI: Accuracy, coverage for dead page predictors

Benchmark	dpPred		dpPred –SH		SHiP-TLB	
	Acc (%)	Cov (%)	Acc (%)	Cov (%)	Acc (%)	Cov (%)
cactusADM	100	98	99	98	70	99
cc	72	70	70	74	67	68
cg.B	83	80	82	80	75	82
sssp	86	78	92	83	88	86
lbm	100	100	100	100	100	65
Triangle	84	23	78	36	55	42
KCore	90	71	88	75	69	81
canneal	72	13	72	13	62	25
pr	82	49	80	50	79	52
graph500	87	21	87	61	70	27
bfs	87	41	74	50	66	59
bc	74	49	49	56	54	47
mis	81	25	68	37	45	22
mcf	67	10	40	21	41	11

Figure 11e shows the impact of LLC size on the performance of the predictors. The height of each bar is normalized to the baseline with the given LLC size. As the size of the LLC per-core is increased from 2MB to 3MB, benefits from the predictors reduce slightly. However, it still remains substantial at 7.03%, on average. Since a larger LLC makes misses less frequent, the scope for improvement reduces.

Figure 11f shows the effectiveness of proposed predictors when using an advanced replacement policy in LLT and LLC. We choose SRRIP replacement policy for this purpose [6]. There are four bars for each application in the figure. The height of each bar is normalized to the IPC of the baseline with LRU replacement policy. The first bar shows normalized IPC with LLT using SRRIP, while LLC continues with LRU. We observe little value in using SRRIP in LLT only. The second bar shows IPC uplifts with dpPred for an LLT that uses SRRIP. We observe that IPC improves by $\tilde{5}$% on average, on top of LLT using SRRIP.

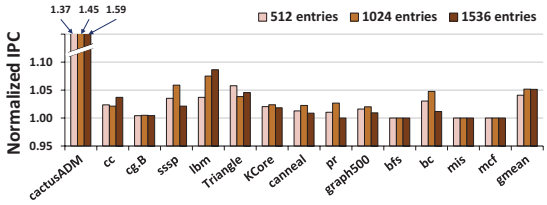
Therefore, dpPred remains effective even in the presence of an advanced replacement policy.

The third bar shows IPC improvement when SRRIP is applied to both LLT and LLC, over LRU. We observe that SRRIP brings significant performance benefit over LRU, when applied to LLC. However, importantly dpPred and cbPred, working in tandem, bring further IPC improvement (6.29%, on average). In short, we find that our predictors retain usefulness even when advanced replacement policies are deployed.

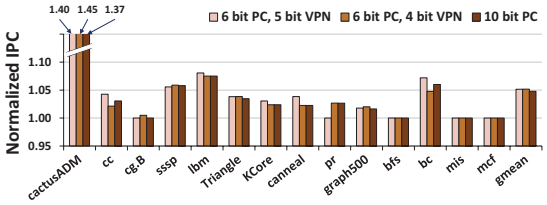
F. Why not use large pages?

A legitimate question could be, why not use large pages to reduce translation overheads and avoid changes to TLB? To this, we wish to point out that large pages are no panacea. Large pages are not transparent to the software. It requires modifications in the operating system and/or to applications.

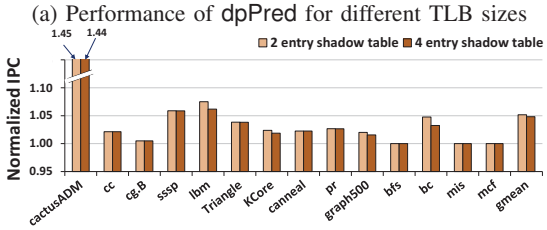
Large pages can significantly increase memory bloat, where an application memory footprint artificially increases due to excessive internal fragmentation [30], [31]. A commonly used technique to employ large pages is Linux's Transparent Huge Pages or THP that maps memory with large pages as a best-effort service but without requiring application modifications.



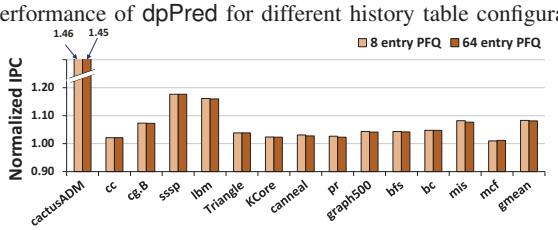
(a) Performance of dpPred for different TLB sizes



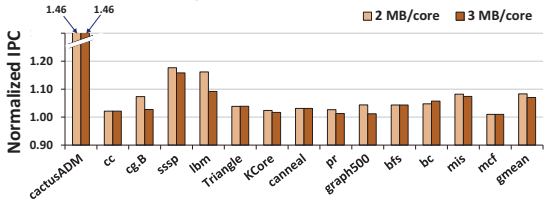
(b) Performance of dpPred for different history table configurations



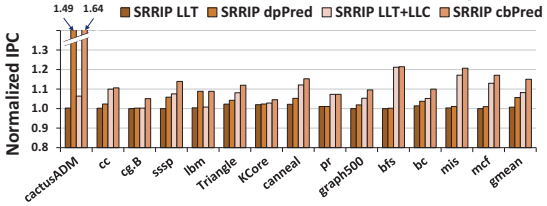
(c) Performance of dpPred for different shadow table sizes



(d) Performance of cbPred for different PFQ sizes



(e) Performance with dpPred and cbPred for different LLC sizes



(f) Performance of cbPred and dpPred when using SRRIP

Fig. 11: Sensitivity studies for our predictor designs

TABLE VII: Accuracy, coverage for dead block predictors

Benchmark	cbPred		cbPred –PF		SHiP-LLC	
	Acc (%)	Cov (%)	Acc (%)	Cov (%)	Acc (%)	Cov (%)
cactusADM	100	66	94	71	94	73
cc	99	40	86	61	89	66
cg.B	100	90	92	92	99	98
sssp	99	24	93	72	96	70
lbm	100	44	90	98	95	99
Triangle	100	43	84	46	93	83
KCore	100	34	95	80	92	96
canneal	100	14	87	67	87	74
pr	99	10	89	35	86	62
graph500	100	28	91	46	96	78
bfs	100	46	93	50	88	64
bc	98	27	90	32	89	71
mis	100	47	86	21	85	50
mcf	100	11	93	54	97	70

Unfortunately, THP is known to introduce latency jitters and performance variance [32]. As a consequence, several commercial software, particularly in-memory databases, advise disabling of THP [7]–[9], [32]–[36].

Even when THP is enabled, only a part of application memory would be mapped using large pages at any point in time. Rest will use default 4KB pages. Further, larges pages are not used by page caches in the file system due to increased overheads of writing back to storage. In short, improving performance under default 4KB pages helps a wide section of applications, even those that use large pages.

Furthermore, proposed predictors are useful to TLBs for large pages too. Unfortunately, quantitatively validating that would require executing applications with a much larger memory footprint, which leads to unrealistic simulation times.

VII. Related Work

We discussed several dead cache block predictors that guide our design in Section II. Here, we discuss other related TLB and cache works.

A closely related but concurrent work to ours is CHiRP [37]. It uses bits from multiple features to design a signature-based TLB replacement policy. Our proposed dead page predictor, dpPred, on the other hand, uses a bypassing approach to optimize the LLT predictor with ½ - 1/7th of the storage overhead incurred by CHiRP. Importantly, we leverage the dead page information in the correlating dead block predictor (cbPred) to efficiently predict dead blocks in LLC. CHiRP does not attempt to predict dead blocks.

Many have explored the use of segments to selectively bypass page walks to reduce translation overheads [38]–[42]. They require changes to both hardware and the OS. TLB prefetching can also reduce TLB misses. Kandiraju *et al.* [43] described three prefetching algorithms for TLBs, among which distance-based prefetching gives the best performance for most workloads. However, prefetching does not perform well across all applications, which suggests that further hardware or software improvements are required [44]. Bhattacharjee et al. proposed inter-core cooperative TLB prefetchers for multi-threaded workloads to share TLB entries [45]. Pham et al. proposed to exploit intermediate contiguity to map multiple pages using a single TLB entry [46]. Bhattacharjee later proposed shared PWCs and efficient page table designs to increase PWC hits [47]. Recently, Guvenilir et al. proposed to leverage unused bits in page table entries to customize

how much of memory each TLB entry maps [48]. Barr et al. proposed SpecTLB to predict address mappings to hide TLB miss latency. MIX TLBs showed how to efficiently support multiple page sizes within a single TLB [49]. Different from these, we extend the concept of dead blocks to TLBs to enhance its usefulness.

There have been numerous research proposals for better management of LLC contents over the years. It is not possible to do justice to all of them, but we mention a few important ones here. Dynamic insertion policy (DIP) used set dueling to dynamically adapt between LRU or MRU policy to get the best of both worlds [5]. Qureshi et al. also made a case for a cache replacement policy to be aware of memory-level parallelism as the impact of all cache misses are not the same [4]. Re-reference Interval Prediction classified blocks based on their re-reference intervals and replaced the block that is predicted to be re-referenced furthest in the future [6]. The HawkEye predictor from 2017 Cache Replacement Championship (CRC) learned from Belady's OPT algorithm by training on past accesses and utilizing it for future replacement decisions [3]. Xiang *et al.* proposed using less-reused cache blocks to take bypassing decisions, apart from zero-reuse blocks [50]. Several research works also proposed bypassing of entries in caches using either reuse distance [51] or using probabilistic bypassing [15]. In our work, however, we focus on LLTs and not caches.

VIII. Conclusion

We explored dead pages in the last level TLB. We observed that a large fraction of LLT entries are dead at any point in time and a good portion of those are dead-on-arrival. We, therefore, designed a storage-efficient predictor for DOA pages in LLT. We also discovered that DOA blocks in LLC and DOA pages in LLTs are correlated. We then leverage this correlation to design a dead-page predictor guided dead-block predictor in the LLC. Together, these predictors improved IPC by over 8%, on average, with only 10.81KB of extra storage (or 0.53% of storage overhead over LLT and LLC).

IX. Acknowledgement

We thank anonymous reviewers of HPCA 2021 for their thoughtful review of this work. We thank Aditya Kamath for his feedback on a draft of this article. This work is supported by Pratiksha Trust, Bangalore, and by a research grant from VMware Inc..

References

[1] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," *SIGARCH Comput. Archit. News*, vol. 29, no. 2, pp. 144–154, May 2001. [Online]. Available: http://doi.acm.org/10.1145/384285.379259

[2] C. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 430–441.

[3] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 78–89, 2016.

[4] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for mlp-aware cache replacement," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ser. ISCA '06. USA: IEEE Computer Society, 2006, p. 167–178. [Online]. Available: https://doi.org/10.1109/ISCA.2006.5

[5] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 381–391. [Online]. Available: https://doi.org/10.1145/1250662.1250709

[6] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 60–71. [Online]. Available: https://doi.org/10.1145/1815961.1815971

[7] "Recommendation to disable huge pages for MongoDB," https://docs.mongodb.org/manual/tutorial/transparent-huge-pages/.

[8] "Recommendation to disable huge pages for NuoDB," http://www.nuodb.com/techblog/linux-transparent-huge-pages-jemalloc-and-nuodb.

[9] "Recommendation to disable huge pages for Redis," http://redis.io/topics/latency.

[10] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 222–233. [Online]. Available: https://doi.org/10.1109/MICRO.2008.4771793

[11] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 175–186. [Online]. Available: https://doi.org/10.1109/MICRO.2010.24

[12] A.-C. Lai and B. Falsafi, "Selective, accurate, and timely self-invalidation using last-touch prediction," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ser. ISCA '00. New York, NY, USA: ACM, 2000, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/339647.339669

[13] M. Kharbutli and Y. Solihin, "Counter-based cache replacement algorithms," in *2005 International Conference on Computer Design*, 2005, pp. 61–68.

[14] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *26th International Conference on Parallel Architectures and Compilation Techniques, PACT 2017, Portland, OR, USA, September 9-13, 2017*, 2017, pp. 180–193. [Online]. Available: https://doi.org/10.1109/PACT.2017.32

[15] S. Gupta, H. Gao, and H. Zhou, "Adaptive cache bypassing for inclusive last level caches," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, pp. 1243–1253.

[16] M. Kharbutli, M. Jarrah, and Y. Jararweh, "Scip: Selective cache insertion and bypassing to improve the performance of last-level caches," in *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 12 2013, pp. 1–6.

[17] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41. USA: IEEE Computer Society, 2008, p. 222–233. [Online]. Available: https://doi.org/10.1109/MICRO.2008.4771793

[18] A. Bhattacharjee, "Translation-triggered prefetching," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 63–76. [Online]. Available: https://doi.org/10.1145/3037697.3037705

[19] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 209–220, May 2002. [Online]. Available: http://doi.acm.org/10.1145/545214.545239

[20] J. Abella, A. González, X. Vera, and M. F. P. O'Boyle, "Iatac: A smart predictor to turn-off l2 cache lines," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, pp. 55–77, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061267.1061271

[21] Z. Wang, K. S. McKinley, A. L. Rosenberg, and C. C. Weems, "Using the compiler to improve cache replacement decisions," in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '02. USA: IEEE Computer Society, 2002, p. 199. [Online]. Available: https://doi.org/10.1109/PACT.2002.1106018

[22] J. B. Sartor, S. Venkiteswaran, K. S. McKinley, and Z. Wang, "Cooperative caching with keep-me and evict-me," in *Proceedings of the 9th Annual Workshop on Interaction Between Compilers and Computer Architectures*, ser. INTERACT '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 46–57. [Online]. Available: https://doi.org/10.1109/INTERACT.2005.7

[23] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2014.

[24] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[25] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

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

[27] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2015.

[28] P. Faldu and B. Grot, "Llc dead block prediction considered not useful," in *13th Workshop on Duplicating, Deconstructing and Debunking, WDDD 2016*, Jun. 2016. [Online]. Available: http://www.eecg.toronto.edu/~enright/wddd/

[29] kernel.org, *Memory Protection Keys*, https://www.kernel.org/doc/html/latest/core-api/protection-keys.html.

[30] A. Panwar, S. Bansal, and K. Gopinath, "Hawkeye: Efficient fine-grained os support for huge pages," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 347–360. [Online]. Available: https://doi.org/10.1145/3297858.3304064

[31] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and efficient huge page management with ingens," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 705–721.

[32] "The black magic of systematically reducing linux os jitter," http://highscalability.com/blog/2015/4/8/the-black-magic-of-systematically-reducing-linux-os-jitter.html.

[33] "Arch Linux becomes unresponsive from khugepaged," http://unix.stackexchange.com/questions/161858/arch-linux-becomes-unresponsive-from-khugepaged.

[34] "Recommendation to disable huge pages for VoltDB," https://docs.voltdb.com/AdminGuide/adminmemmgt.php.

[35] "Why TokuDB Hates Transparent HugePages," http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Hadoop_Tuning_Guide-Version5.pdf.

[36] "Tales from the Field: Taming Transparent Huge Pages on Linux," https://www.perforce.com/blog/151016/tales-field-taming-transparent-huge-pages-linux.

[37] S. Mirbagher-Ajorpaz, E. Garza, G. Pokam, and D. A. Jiménez, "Chirp: Control-flow history reuse prediction," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 131–145.

[38] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 237–248. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485943

[39] A. Basu, "Revisiting virtual memory," 2013, http://research.cs.wisc.edu/multifacet/theses/arka_basu_phd.pdf.

[40] A. Basu and M. M. Hill, Mark D. amd Swift, "Virtual memory management system with reduced latency," 2015, https://patents.google.com/patent/US9158704B2/en.

[41] J. Gandhi, A. Basu, M. D. Hill, and M. M. Swift, "Efficient memory virtualization: Reducing dimensionality of nested page walks," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 178–189. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.37

[42] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant memory mappings for fast access to large memories," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 66–78. [Online]. Available: http://doi.acm.org/10.1145/2749469.2749471

[43] G. B. Kandiraju and A. Sivasubramaniam, "Going the distance for tlb prefetching: An application-driven study," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 195–206, May 2002. [Online]. Available: http://doi.acm.org/10.1145/545214.545237

[44] G. Kandiraju and A. Sivasubramaniam, "Characterizing the d-tlb behavior of spec cpu2000 benchmarks," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 129–139, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/511399.511351

[45] A. Bhattacharjee and M. Martonosi, "Inter-core cooperative TLB for chip multiprocessors," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 359–370. [Online]. Available: http://doi.acm.org/10.1145/1736020.1736060

[46] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced large-reach TLBs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2012, pp. 258–269.

[47] A. Bhattacharjee, "Large-reach memory management unit caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 383–394. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540741

[48] F. Guvenilir and Y. N. Patt, "Tailored page sizes," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 900–912.

[49] G. Cox and A. Bhattacharjee, "Efficient address translation for architectures with multiple page sizes," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017, pp. 435–448. [Online]. Available: http://doi.acm.org/10.1145/3037697.3037704

[50] L. Xiang, T. Chen, Q. Shi, and W. Hu, "Less reused filter: Improving l2 cache performance via filtering less reused lines," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 68–79. [Online]. Available: https://doi.org/10.1145/1542275.1542290

[51] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. USA: IEEE Computer Society, 2012, p. 389–400. [Online]. Available: https://doi.org/10.1109/MICRO.2012.43