Address Translation Conscious Caching and Prefetching for High Performance Cache Hierarchy

Vasudha
Dept. of CSE
*Indian Institute of Technology Kanpur**
Qualcomm India
vasudha41@gmail.com

Biswabandan Panda
Dept. of CSE
Indian Institute of Technology Bombay
biswa@cse.iitb.ac.in

***Abstract*—Performance of Translation Lookaside Buffers (TLBs) and on-chip caches plays a crucial role in delivering high-performance for memory-intensive applications with irregular memory accesses. Our observations show that, on average, an L2 TLB (STLB) miss for address translation can stall the head of the reorder buffer (ROB) for a maximum of 50 cycles. The corresponding data request, also called as the replay load can stall the head of the ROB for more than 200 cycles. We show that current state-of-the-art mid-level (L2C) and last-level cache (LLC) replacement policies do not treat cache block with address translations and replay data access differently. As a result these policies fail to reduce ROB stalls because of translation and replay data access misses.**

To improve the performance further on top of high-performing cache replacement policies, we propose address translation and replay data access conscious cache replacement policies at L2C and LLC. Our enhancements help in reducing ROB stalls due to STLB misses by 28.76%. We also find that cache blocks storing replay loads are dead (no reuse after insertion), and cache replacement policies alone cannot mitigate the ROB stalls caused by replay data accesses. Hence, we propose an address translation hit triggered hardware prefetcher that brings replay data on an address translation hit at the L2C and LLC. This enhancement reduces ROB stalls due to replay data accesses by 18.5%. For a group of memory-intensive benchmarks with high STLB misses, our enhancements improve performance by 5.1% (reducing ROB stall cycles by 46.7%) and as high as 10.6%, on top of state-of-the-art cache replacement policies that are highly competitive. Our enhancements do not incur any additional storage overhead. However, we need additional flags from the page-table-walker into the cache hierarchy.

I. Introduction

The effectiveness of translation look-aside buffers (TLBs) and on-chip caches play an important role in overall system performance for applications with huge data footprints and irregular memory accesses. For memory-intensive applications with huge data footprints, low-performance L2 TLB (STLB) can severely degrade system performance. A translation miss at the STLB can induce a five-level page table walk may require five off-chip memory accesses. After a successful address translation, a processor sends the corresponding data request (known as the replay data load). If it misses at the on-chip cache hierarchy, requires one more memory access. So, overall a demand load can cause six DRAM accesses (five for address translations and one for the corresponding data). Since on-chip caches store both address translations and data blocks, effective management of the cache hierarchy keeping both address translations and data blocks is crucial.

*The work was done while the author was an MS student at IIT Kanpur.

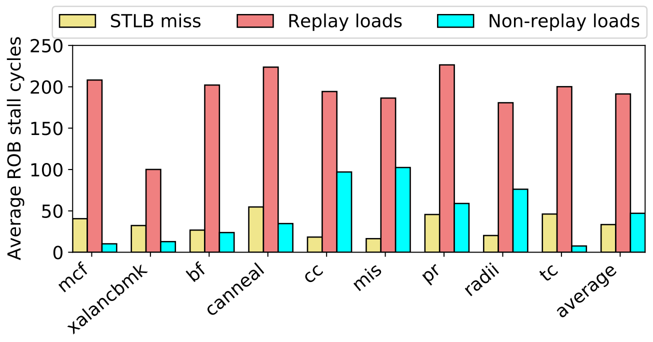

Fig. 1. ROB stalls due to STLB misses, its corresponding replay loads, and the remaining (non-replay) loads.

Replay and non-replay demand loads. We use the term *replay load* for those data requests (demand loads) whose corresponding address translations, miss at the STLB, and walk the page table. We call a demand load request a non-replay load if the corresponding address translation gets a hit at the STLB. These terms were proposed in one of the prior works [11].

The Problem. Modern deeper and wider out-of-order processors with a large reorder buffer (ROB) of 352 entries [1] can amortize i) translation misses at L1 TLB (DTLB) that get hit at STLB, and ii) their corresponding replay loads. However, these processors fail to amortize all the STLB misses and their corresponding replay loads. Fig. 1 shows that an STLB miss can stall at the head of the ROB for a maximum of 54 cycles and, on average, for 33 cycles. In comparison, the corresponding replay loads can stall the head of the ROB for a maximum of 226 cycles and on average, for 191 cycles. On the other hand, on average, non-replay loads stall the head of the ROB for 47 cycles. *So, a demand load request that misses at the STLB stalls the head of the ROB for a maximum of 280 (226+54) cycles and on average for 224 (191+33) cycles.*

For Fig. 1, we use irregular memory-intensive benchmarks from SPEC CPU2017 [4], PARSEC [3], and Ligra [2] suites,

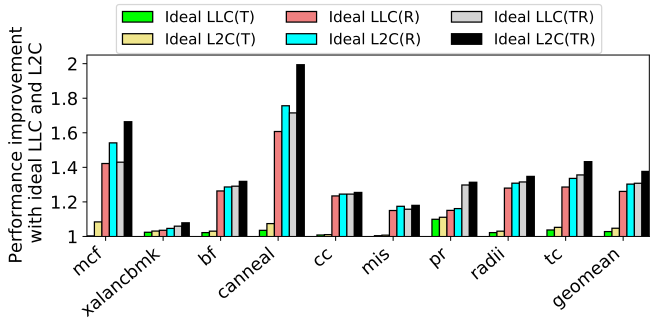

Fig. 2. Normalized performance with ideal L2C and LLC for leaf-level translations (T), replay loads (R) and for both of them (TR).

with average (maximum) misses per kilo instructions (MPKI) of 15.60 (35.69), and 31.25 (62.83), for STLB and LLC, respectively. We use a data-set of size 918MB for graph workloads, maximum of 4GB for SPEC workloads, and a maximum of 2.3GB for PARSEC workloads. Table I lists all simulated parameters (similar to Intel Sunny Cove) [1].

Opportunity. Fig. 2 shows the performance improvement that can be achieved if we have an ideal LLC and ideal L2C for leaf-level address translations, their corresponding replay loads, and for both address translations and replay loads. We use the term *leaf-level translations* for the last page table level that stores the required physical address of the page. An L1D cache is smaller in size, and non-replay loads dominate the cache due to their frequent reuse. Optimizing a small L1D cache with respect to non-frequent replay loads and address translations affects non-replay loads. Hence, we explore the scope of optimizations at L2C and LLC.

We simulate an ideal LLC for leaf-level address translations by providing a 100% hit rate to all those translations that missed at L2C. On a miss at the LLC, we respond with a hit latency of LLC. However, to model the impact of bandwidth on the ideal performance we send the miss request to miss-status-holding-registers(MSHRs). Similarly, we simulate it separately for replay loads and then for both address translations and replay loads. Similarly, we simulate the ideal L2C. On average, an ideal LLC for address translations and their corresponding replay loads provides 30.7% improvement on top of high-performing cache replacement policies at the L2C and LLC. An ideal L2C provides a performance improvement on top of the ideal LLC, leading to performance improvement of 37.6%. If we concentrate only on leaf-level translations at the L2C, then we get a performance improvement of 4.7%. If we consider ideal L2C only for the replay loads, we get a performance improvement of 30.2%, and for both leaf-level translations and replays, we get 37.6%.

Our goal. Our goal is to improve the performance of address translations and their corresponding replay loads in a multi-level cache hierarchy without incurring additional storage overhead.

Prior works. Prior works like TEMPO [11] helps in mitigating replay load latency by proposing a prefetcher at the DRAM controller that expedites replay load response. Some of the prior works [18] [17] improve the TLB and LLC coverage by bypassing dead pages and dead blocks and by a TLB aware cache partitioning. However, we find that these techniques do not mitigate the overall address translation and replay load penalty that stall the head of the ROB. OS-aware cache insertion policy [22] proposes a dynamic insertion policy on LRU replacement policy to reduce OS interference at the LLC. We find that with the current trend of large LLCs (2MB/core) and L2C of (512KB/core) and a large STLB of 1536 entries, there is marginal OS interference at the LLC. Instead, there is interference from the user-level accesses on the OS accesses. We corroborate the findings of [22] for a one-level TLB and two-level cache hierarchy with 64KB of L1D and 1MB of L2C as mentioned in [22] and find that with small caches page-table walk accesses can affect user level accesses at the LLC.

Our key observations. We find that state-of-the-art cache replacement policies [14] [21] [13] at the L2C and LLC are not conscious of address translations and replay loads and do not learn the reuse behavior of address translations, replay loads, and non-replay loads separately even though their reuse behaviors are different. For some of the benchmarks, state-of-the-art policies perform worse than LRU in terms of covering address translations and replay loads. One of the primary reasons for this trend is the signature (e.g., an instruction pointer) based reuse learning of cache blocks (more details in Section III). We observe that more than 95% of cache blocks that store data for replay accesses are *dead*;no reuse after a block is inserted into the cache (more details in Section III). Finally, we show that even state-of-the-art hardware data prefetchers [19] [16] [10] [12] fail to mitigate the ROB stalls because of a replay load access (more details in Section III). Based on these observations, we make the following key contributions.

- We propose minor enhancements that enhances cache block reuse training of cache blocks that store address translations and replay loads. We insert cache blocks that comprise address translations with low priority for eviction at L2C and LLC, so that these blocks will stay longer. To make the L2C replacement policy aware of the replays, we insert replay loads with high priority for eviction (as we find that the blocks containing replay loads are dead). This enhancement improves address translation on-chip cache hierarchy hit rate to 99%. Note that this enhancement incurs zero storage overhead (Section IV).
- To further mitigate the ROB stalls, we utilize the previous enhancements that strive for 100% address translation hit rate at the on-chip cache hierarchy and prefetch the corresponding replay loads. We trigger prefetching of replay loads once we get a hit for an address translation at the L2C or LLC, effectively reducing the ROB stall time because of replay loads (Section IV).
- Overall, our enhancements mitigate ROB stalls by 46.7% resulting in an average performance improvement of

5.1% and as high as 10.6%. In the case of a 2-way SMT processor where two threads that share the memory hierarchy, our enhancements are equally effective with an average performance improvement of 6.3% and as high as 12.6% (Section V).

II. Background

This section provides the necessary background on the address translation process and cache replacement policies.

A. Address translation

Page table. Processor demands load and store requests through virtual addresses that need to be translated to physical addresses in order to get the corresponding data. Modern processors, after Ice Lake processors [5] based on Intel Sunny Cove micro-architecture [1] uses 57 bit virtual address. This virtual to physical address translation is stored in a radix-tree structured five-level page table. An entry in the page table occupies 8B, also called page table entry (PTE).

Page table walker(PTW). It is a hardware structure that walks the page table by extracting the page-offset from the virtual address to obtain the physical address of the page. The virtual address is divided into 9-bit chunks, corresponding to every level. The CR3 register stores the pointer to the outermost page table, i.e., level five. For page size of 4KB, PTW adds the first chunk of the virtual address(VA[56:48]) to the base address of page table level five to get the page offset of the fifth level page. The corresponding PTE entry stores the page frame number of the next level, i.e., the fourth level. PTW then uses the next 9-bit chunk (VA[47:39]) to get the page offset of PTE. Page offset is retrieved from the last chunk of the virtual address (VA[11:0]).

Translation Look-aside Buffer(TLB). A TLB stores recent virtual to physical address translations. The first level TLB comprises Data TLB(DTLB) and Instruction TLB(ITLB). The second level TLB is a unified TLB. In the case of the DTLB/ITLB hit, the corresponding physical address is used as the translated address. On the other hand, in case of a miss, the address translation request is forwarded to 2nd level TLB (STLB). If it gets an STLB hit, the entry is filled in the first level TLB also. If it gets an STLB miss, then PTW is initiated for page walk.

Paging Structure Caches(PSCs). PSCs (four in number for a five-level page table) store the recently accessed PTEs of intermediate page table levels. PTW searches all the PSCs concurrently after an STLB miss. In case of more than one hit, the farthest level is considered as it minimizes the page table walk latency. In case of a PSC miss, the translation request goes through the cache hierarchy. All the data caches (L1D/L2C/LLC) store eight contiguous translations of all the page table levels (eight PTEs = 64 bytes) in a cache block, which is 64 bytes. If all these caches miss, then PTW accesses the DRAM.

B. LLC replacement policies

In the last one decade, many high-performing LLC replacement policies [14] [21] [13] have been proposed. We outline some of the recent and high-performing policies. A cache replacement policy consists of three sub-policies: (i) insertion, (ii) promotion, and (iii) eviction. An insertion policy decides the replacement priority of cache block at the time of a cache fill, a promotion policy decides the replacement priority on reuse, and an eviction policy decides on a miss, which block to use for eviction;

SRRIP. Static re-reference interval prediction (SRRIP) [14] stores an two-bit re-reference prediction value (RRPV) per block. On a miss to cache set, it *evicts* the block with a distant re-reference interval, RRPV of 3 (2^2-1) and *inserts* the incoming block with RRPV of 2. On a cache hit, SRRIP promotes the block to RRPV=0. Inserting blocks at RRPV=2 neither supports the new block to have an immediate (RRPV=0) nor distant (RRPV=3) re-reference interval. On a cache miss, for eviction, if no block with RRPV=3 is available within a cache set, then the RRPV of all the blocks is incremented by one.

DRRIP. Dynamic RRIP [14] uses two policies; a) SRRIP and b) bimodal RRIP(BRRIP). For thrashing access patterns, BRRIP inserts cache blocks mostly with 3. Only a few blocks are inserted with an RRPV of 2. DRRIP dynamically chooses the insertion policy out of the two for every set through a technique called set dueling [14].

SHiP. SHiP [21] uses SRRIP policy to select the victim block and promotes the block with RRPV=0 on a hit. However, it dynamically predicts the re-reference interval on every cache block insertion using signatures (e.g., instruction pointers (IPs)). It uses Signature History Counter Table (SHCT) that stores a counter per signature. If a cache block provides a hit, the counter is incremented. Else, if the cache block gets evicted without being referenced, the counter corresponding to that signature is decremented. If the signature corresponding to the upcoming block has counter value zero in the SHCT table, then the block is inserted with 3, else with 2.

Hawkeye. Hawkeye [13] learns a long history of accesses to align its future replacement decisions with Belady's optimal policy. It uses two main components, OPTgen and Hawkeye predictor. OPTgen uses the cache occupancy vector and usage interval of a block to determine whether it will get hit on future accesses. Hawkeye uses a three-bit RRPV and assigns RRPV=0 for cache-friendly and RRPV=7 for cache-averse blocks upon insertion. It selects the block with RRPV=7 as a victim. If no such block is available, it selects the block with the highest RRPV value.

In this paper, we evaluate DRRIP at the L2C, and SHiP and Hawkeye at the LLC.

III. Motivating Observations

Performance of cache hierarchy in terms of address translations. Fig. 3 shows from which level of the cache hierarchy, leaf-level translations get their responses after an STLB miss. On average, out of all the STLB misses, 23% get serviced at L1D, 55.6% at L2C, and 15.1% get serviced at LLC. The remaining 6.3%, miss at LLC. Delay in address translation also delays further corresponding replay loads, and more than 80% replay loads get miss at LLC.

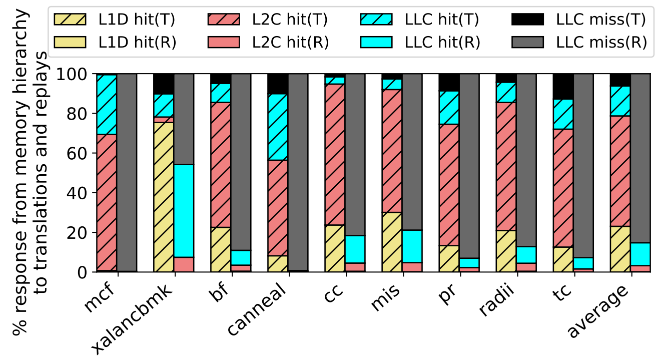

Fig. 3. Distribution of memory-hierarchy responses to leaf-level translations because of STLB misses(T) that stall the head of the ROB and their corresponding replay loads(R).LLC miss(T) and LLC miss(R) correspond to responses for translations and their replays from the DRAM.

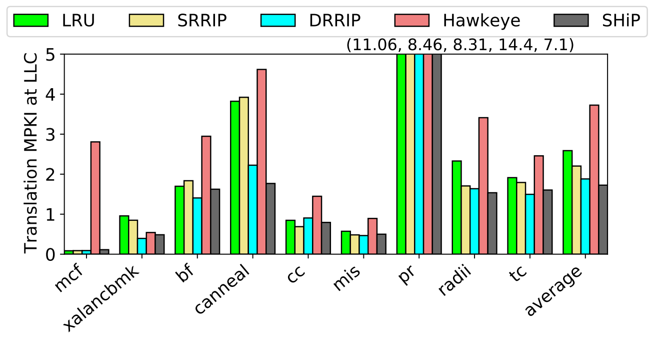

Fig. 4. Leaf-level translation MPKI at the LLC with various replacement policies.

LLC replacement policies fail to keep all the cache blocks that comprise address translations. Fig. 4 shows the leaf-level translation MPKI at LLC with various LLC replacement policies. On average, the MPKI for SRRIP, DRRIP, and SHiP decreases as compared to LRU by 14.72%, 27.45%, and 33.3%, respectively. On the other hand, leaf-level translation MPKI increases by 44.1% with Hawkeye. Both SHiP and Hawkeye use IP based training for predicting the usefulness of cache block (cache-friendly or cache-averse). Hawkeye uses reuse distance for training. SHiP observes if the cache block gets a hit before getting evicted to train it as cache-friendly or cache-averse. The original proposals do not consider cache blocks with address translations and train both the data blocks and address translation blocks impartially, leading to noise in learning and predicting the reuse behavior of cache blocks.

Why do LLC replacement policies fail in keeping address translation blocks? There are two primary reasons: (i) not all the PTEs have the same or similar reuse behavior, and classifying all the eight PTEs based on one leaf-level translation is misleading. Ideally, the reuse behavior of all eight PTEs should be trained independently, and (ii) the IP that is used as a signature for learning the reuse behavior of data blocks is also used for training of translation blocks. This leads to noise in training as the reuse behavior of data blocks and translation blocks are different even though they belong to the same IP.

For example, if an instruction X with IP$_X$ generates a load to virtual address VA and gets a TLB miss at the STLB. It further initiates a PTW, and let's say it misses at the L1, L2C, and LLC, and finally inserts a PTE at LLC with IP$_X$ as the signature. If the demand loads are cache-averse in nature, then SHiP and Hawkeye will assume the address translation blocks brought by IP$_X$ will also be cache-averse.

Since, non-replay demand loads are frequently reused as compared to leaf-level translations at LLC, Hawkeye classifies non-replay loads as cache-friendly and leaf-level translations as cache-averse. Hence, the translation MPKI with Hawkeye increases significantly as compared to other policies. On the other hand, SHiP focuses on whether a cache block got used before eviction; the IP-based training of SHiP is more accurate than Hawkeye. This property does not let non-replay memory references get prioritized so aggressively as in Hawkeye. However, the translation MPKI at LLC with SHiP is still greater than one. This trend shows that there is a need for better signatures for dealing with address translation blocks.

Recall distance of address translations. Fig. 5 shows the recall distance of address translation blocks at the LLC and L2C, respectively. We define recall distance as the distance in terms of a number of unique accesses that arrive in the same cache set. For example, for the LLC, we calculate the number of unique accesses to a set after a block gets evicted from the LLC and before the next request to the same block arrives at the LLC. Please note that this is different from one of the popular metrics called reuse distance. As we can see, around 30% of the address translation blocks have a recall distance within ten, and more than 70% of the blocks have a recall distance within 50. *This shows that if we can keep the address translation blocks for 10 more accesses in their respective cache sets then they will get LLC and L2C hits.*

Cache blocks storing replay loads are dead in the LLC. Fig. 6 shows LLC MPKI for replay loads with various replacement policies. As we can see, there is no effect of replacement policies on replay demand accesses. Fig. 7 shows the recall distance of replay loads at the LLC and L2C, respectively. As we can see, more than 60% of the blocks have a recall distance of more than 50 unique accesses to their respective sets, which explains why all the replacement policies fail to keep these kinds of blocks. This motivates for proposals [18] [15] that bypass dead blocks at the LLC. Though LLC bypassing helps in providing more cache space to non-replay loads, but it does not help in mitigating the ROB stalls because of replay loads.

Data prefetchers fail to bring replay loads into the cache hierarchy. Fig. 8 shows that state-of-the-art spatial prefetchers [19] [10] [16] fail in prefetching the replay loads. One of the primary reasons for the same is even though these prefetchers allow cross-page training, they do not perform cross-page prefetching as some of them like SPP and Bingo are employed at the L2C. Instruction Pointer Classifier-based Spatial Hardware Prefetching (IPCP) is an L1D prefetcher and

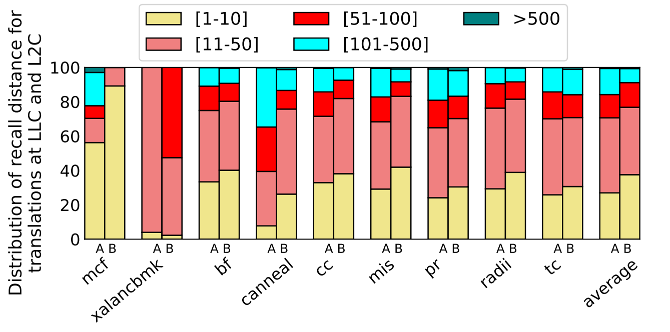

Fig. 5. Recall distance of leaf-level translations at LLC (A) and L2C (B).

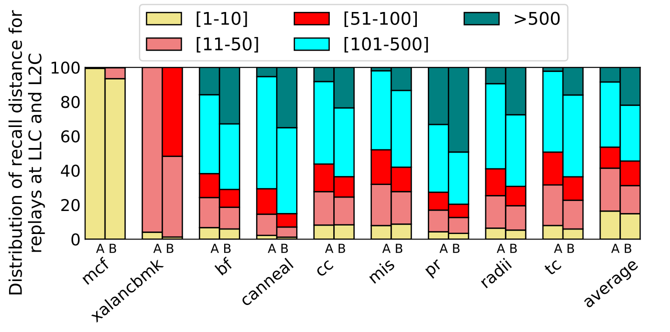

Fig. 7. Recall distance for replays at the LLC (A) and L2C (B).

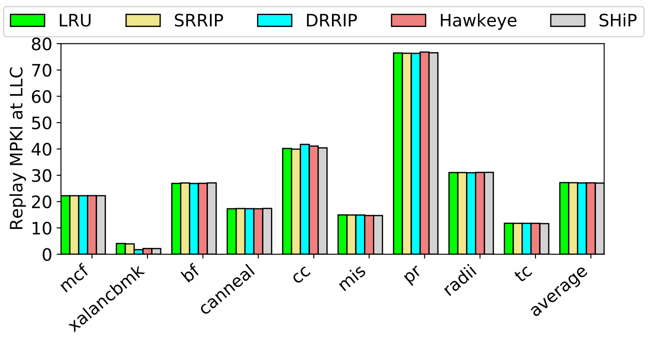

Fig. 6. Replay load MPKI at the LLC with various replacement policies.

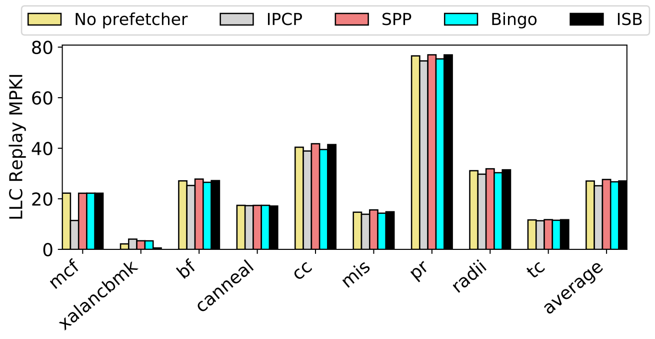

Fig. 8. LLC Replay MPKI with and without perfetchers.

can perform cross-page prefetching. However, even a cross-page IPCP prefetcher fails to hide the ROB stalls because of a replay load request. The primary reason for the same is late prefetching. IPCP with cross-page prefetching sends a prefetch request to STLB, and on an STLB miss, the prefetch request doesn't proceed till the STLB fills, delaying the prefetch response. We find that in benchmarks like `mcf` and `bf`, IPCP improves the ROB stall due to replay loads by 5%. However, the average improvement in L2C and LLC replay load MPKI is less than 1%. On the other hand, temporal prefetchers like Irregular Stream Buffer (ISB) [12] performs significantly better for some of the benchmarks like `xalancbmk`, and overall improve ROB stalls because of replay loads by 20%. We do not show performance with other temporal prefetchers like IMP [23], as On average, ISB performs better than IMP [23] with an average performance improvement of 2%. Overall, on average, state-of-the-art prefetchers fail to hide the latency of replay load requests.

IV. Improved Caching for Address Translations and Replay Loads

At a high level, we propose enhancements to cache replacement policies for improving translation miss coverage at the L2C and LLC. This helps in retaining address translations for a longer period at the L2C and LLC so that a significant fraction of STLB misses get their responses from L2C and LLC. Next, we propose an address translation hit triggered data prefetcher that prefethes replay loads on an L2C or LLC hit to an address translation.

Address translation conscious signatures. SHiP and Hawkeye use IP as a signature for training and predicting the reuse of LLC blocks. As discussed in Section III, SHiP and Hawkeye do not differentiate reuse learning of data blocks from translation blocks causing noise in reuse training, resulting in premature eviction of address translation blocks. An ideal and non-trivial solution for this problem is to learn the reuse behavior of each PTE present within a cache block and group PTEs into two signatures: cache-friendly IPs and cache averse IPs. We propose a simple enhancement that differentiates an IP signature based on whether the IP is filling a PTE, replay load, or non-replay load into an LLC. We augment an additional flag and redefine the signatures (IPs) that are used for learning the reuse behavior of address translations and demand replay loads as follows:

$$signature_{translations} = IP << IsTranslation$$

$$signature_{replayloads} = IP << IsReplay + IsTranslation$$

For non-replay loads, we do not change the signature and use the signature as suggested in SHiP and Hawkeye. With the new signatures, we make sure reuse learning of replay loads, non-replay loads, and address translations are independent of each other, mitigating noise in the training. Note that for L2C, we do not need these changes as L2C uses DRRIP, a policy that does not use any signature.

L2C insertion policy. Fig. 9 shows our enhancements for the L2 replacement policy. At L2C, we insert replay loads with the highest priority (RRPV=3) for eviction as replay loads

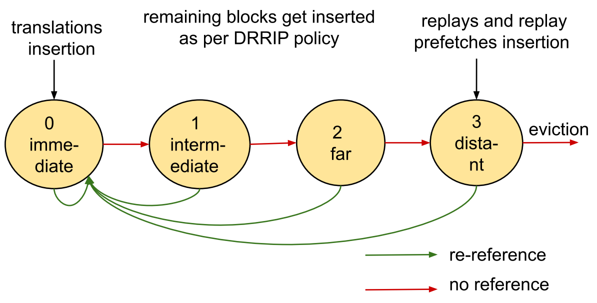

Fig. 9. RRPV transition with the L2C replacement policy (T-DRRIP)

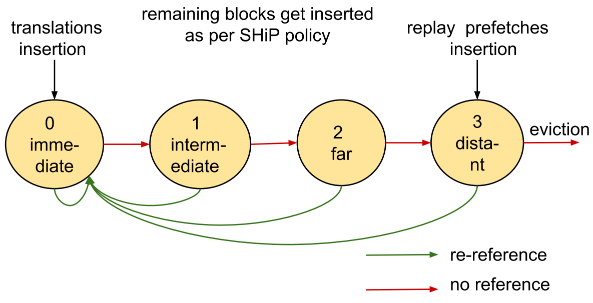

Fig. 11. RRPV transition with the LLC replacement policy (T-SHiP)

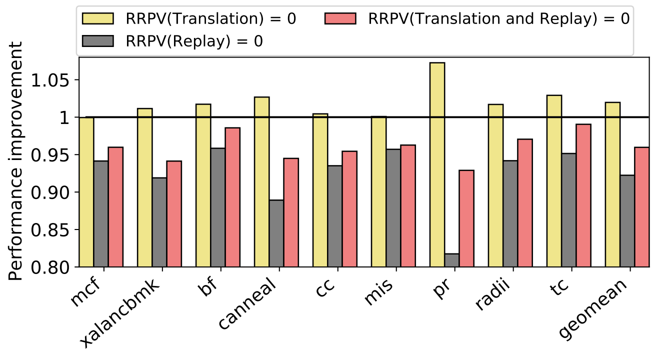

Fig. 10. Performance degradation with RRPV=0 insertion for replay loads for DRRIP at L2C and SHiP at the LLC.

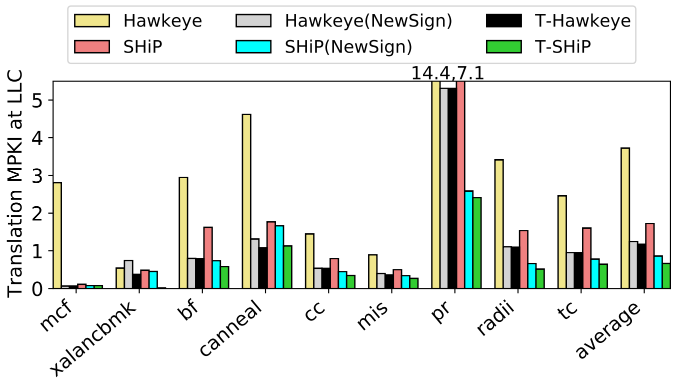

Fig. 12. Leaf-level translation MPKI at LLC. NewSign: New enhanced IP signature.

are dead and do not get any reuse. However, we insert leaf-level address translations with RRPV=0, the lowest priority for eviction so that translations will stay at the L2C for a longer period of time. As the L2C is dominated by non-replays and more than 96% replays get misses at LLC, there is no performance penalty in bringing these replay data blocks with the highest priority for eviction. Note that just inserting address translations with RRPV=0 does not improve performance unless their corresponding replay loads are also inserted with RRPV=3. Fig. 10 shows the performance degradation when both the address translations and replay loads are inserted with RRPV=0. This is because the replay loads affect the replacement priorities (if they get inserted with RRPV=2) of cache blocks with translations as RRIP based policies increment RRPV values of all the blocks within a set when it fails to find a block for eviction (RRPV=3). We call this policy, address translation conscious DRRIP(**T-DRRIP**).

LLC insertion policy. Fig. 11 shows our enhancements for insertion through the LLC replacement policy. At the LLC, we enhance SHiP [21] and insert leaf-level address translations with RRPV=0. We do not enhance the LLC replacement policy for replays as SHiP is already effective in handling these dead data blocks with new IP signatures. We call this policy as address translation conscious SHiP (**T-SHiP**). We, similarly enhance Hawkeye and call it T-Hawkeye. As we can see SHiP outperforms Hawkeye even in the baseline without our enhancements, we use SHiP as the baseline LLC policy as a strong baseline.

Note that the promotion policy and eviction policy of **T-DRRIP** and **T-SHiP** remains the same as in DRRIP [14] and SHiP [21], respectively. So, in summary, we propose simple extensions in the form of better signatures with L2C and LLC management policies that are aware of address translations and replay loads. Fig. 12 shows the improvement in address translation MPKI at the LLC, with these enhancements. Note that we do not need this enhancement for the private L2C as it uses DRRIP that does not use any IP based signatures.

Address translation initiated replay-load prefetcher. With T-DRRIP and T-SHiP, translations stay at the cache hierarchy for more time and enjoy more cache hits. We propose an address-translation triggered hardware prefetcher (**ATP**) that prefetches a replay load as soon as its corresponding translation hits at on-chip caches, thanks to translation aware cache management policies, T-DRRIP and T-SHiP. However, to implement this, we modify the PTW with an additional bit (IsLeafLevel) that indicates whether the PTW is walking through the leaf-level of the page table. Note that the leaf-level of the page table provides the required physical page address. The PTW also carries the upper six bits of the page offset for the final level page-table walk. When the PTW gets a hit at the on chip L2C or LLC, it triggers hardware prefetching for the

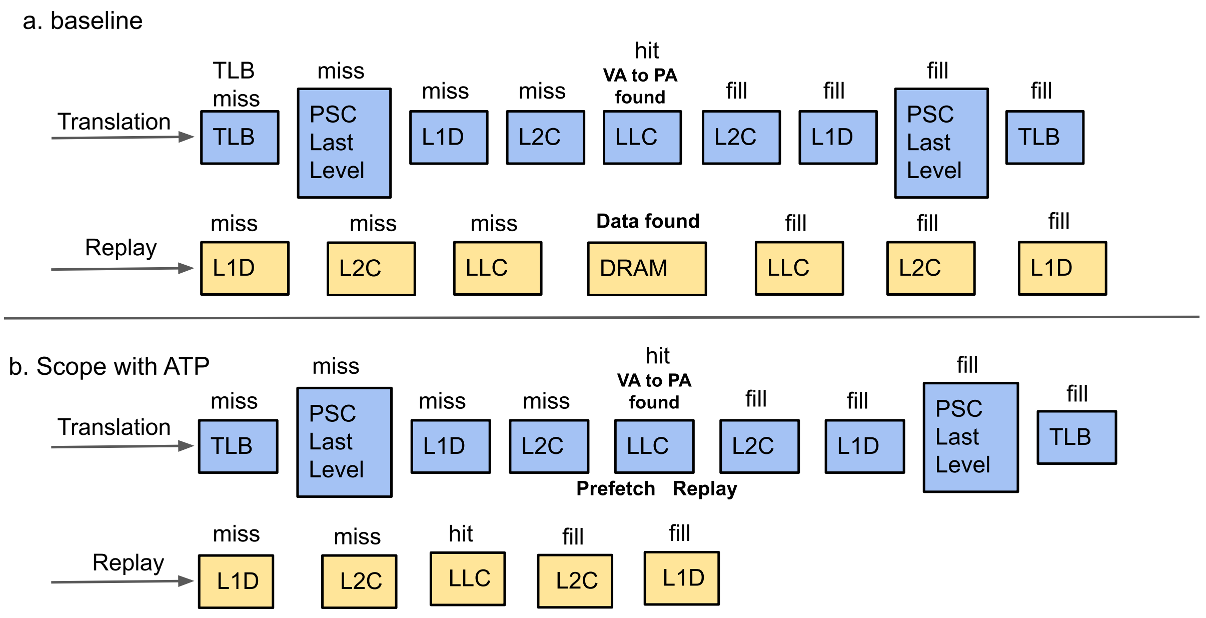

Fig. 13. Scope of address translation initiated replay-load prefetcher (ATP). In general, ATP converts one replay load LLC miss into an LLC hit, which is not the case with the baseline with improved caching like T-SHiP. However, there are replay loads for which ATP hides a fraction of LLC miss latency and not the complete LLC miss latency.

TABLE I
SIMULATED PARAMETERS.

Core	Out-of-order, hashed perceptron branch predictor [20], 4GHz with 6 issue width, 4 retire width, 352 entry ROB
TLBs	64 entry 4-way at L1 DTLB/ITLB (1 cycle), 2048 entries 16-way entry L2 STLB (8 cycles)
MMU Caches	2 entry (PSCL5), 4 entry (PSCL4), 8 entry (PSCL3), 32 entry (PSCL2), searched parallely, one cycle
L1	32KB 8-way L1I (4 cycles), 48KB 12-way L1D (5 cycles)
L2	512KB 8-way associative (10 cycles), DRRIP [14]
LLC	2MB/slice 16-way (20 cycles), SHiP [21]
DRAM	1 channel/4-cores, DDR5, 6400 MT/sec

replay with the highest priority for eviction (e.g., RRPV=3 for DRRIP). Fig. 13 illustrates the scope of ATP. In case of a translation miss at the LLC, we trigger **TEMPO** [11] prefetching that triggered prefetching for replay loads at the DRAM controller.

We do not trigger prefetching on an L1D translation hit as the scope (time gap between translation hit and data request to L1D is small) for replay load prefetching is minimum. Note that through this prefetching, we improve the replay load miss latency (not miss rate) as the prefetched block is on the way from DRAM before the replay load request comes to L2C/LLC. Ideally, ATP can hide one off-chip DRAM access latency.

ATP and **TEMPO** are beneficial for such cases in prefetching a replay and also, accelerating other prefetch requests. We show the performance of our enhancements combined with state-of-the-art data prefetchers in section V.

V. Evaluation

In this section, we evaluate our enhancements based on performance improvements and reduction in ROB stalls because of address translations and replay load misses. We also compare our enhancements with some of the recent proposals that try to mitigate address translation latency. Finally, we evaluate our results for a Simultaneous multi-threading (SMT) core and multi-core environments. We use ChampSim simulator that is used in the recent ISCA championships for cache replacement, data, and instruction prefetching [7] [6] [8] [9]. Table I shows our simulated parameters. For each benchmark, we simulate their region of interests of 10B instructions after a warmup of 100M instructions. Table II provides the details of the benchmarks that we use in our study.

A. Performance

Fig. 14 shows the performance improvement (in terms of reduction in execution time) with our enhancements normalized to the baseline. On average, more than 98% of the leaf-level address translations now hit at the on-chip cache hierarchy. On average, we get a performance improvement of 5.1% and as high as 10.6%. T-DRRIP provides an average performance improvement of 0.5% that improves to 2.9%, 4.8%, and 5.1% with T-SHiP, ATP, and TEMPO [11], respectively. Note that these improvements are on top of state-of-the-art cache management policies that are highly competitive, and a 5% improvement on top of these policies is a significant performance boost. Also, with our on-chip enhancements, the effectiveness of TEMPO is marginal (a performance boost from 4.8% to 5.1%) as, on average, only 2% of address

TABLE II
BENCHMARKS AND THEIR CORRESPONDING L2 AND LLC MPKIS FOR LEAF-LEVEL TRANSLATIONS(PTL1), REPLAY LOADS, AND NON-REPLAY LOADS. NOTE THAT THE MEMORY FOOTPRINT OF SIMULATED REGIONS ARE IN THE RANGE OF 200MB TO 400MB.

Benchmark	Suite	Data-set size	MPKI Category	STLB	L2C Replay	L2C Non replay	L2C PTL1	LLC Replay	LLC Non replay	LLC PTL1
xalancbmk	SPEC CPU2017	500MB	Low	4.78	4.37	17.27	1.04	2.16	7.81	0.48
tc	Ligra	918MB	Medium	12.54	12.35	10.88	3.51	11.64	8.59	1.6
canneal	PARSEC	2.3GB	Medium	17.54	17.51	4.15	7.65	17.41	4.07	1.76
mis	Ligra	918MB	Medium	18.64	17.76	63.68	1.49	14.7	39.07	0.49
mcf	SPEC CPU2017	4GB	Medium	22.35	22.27	8.21	6.84	22.24	4.5	0.11
bf	Ligra	918MB	High	33.31	29.37	42.06	4.82	27.10	34.18	1.62
radii	Ligra	918MB	High	35.69	34.08	44.91	5.18	31.11	31.86	1.54
cc	Ligra	918MB	High	49.5	47.25	4.94	66.15	40.40	42.54	0.79
pr	Ligra	918MB	High	82.29	80.43	44.65	20.98	76.53	35.63	7.1

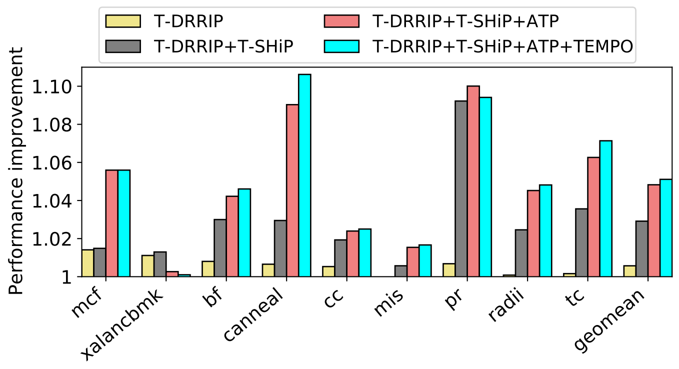

Fig. 14. Normalized performance with our enhancements.

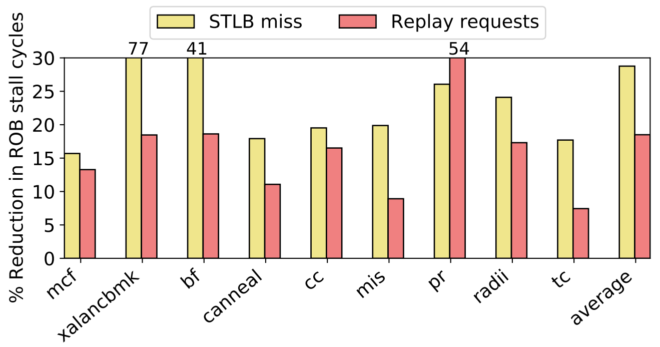

Fig. 16. Reduction in ROB stall cycles due to STLB miss and replay requests.

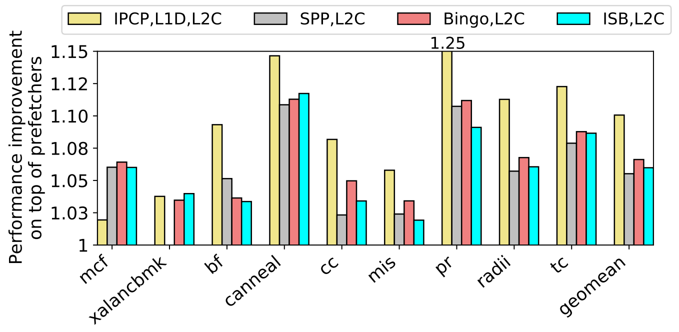

Fig. 15. Normalized performance in the presence of prefetchers.

translations go to the DRAM. For `pr`, we observe that the address-translation MPKI at the LLC increases slightly (1.65 becomes 1.98) with TEMPO as TEMPO's prefetched blocks cause evictions of translation blocks at the LLC. We observe a similar trend with `xalancbmk` with one difference which is the non-replay LOAD MPKI also increases from 8.3 to 8.5.

Performance in the presence of hardware prefetchers. Our enhancements are slightly more effective in the presence of a baseline with state-of-the-art data prefetchers IPCP [19], Bingo [10], SPP [16], and ISB [12] with performance improvements of 11.2%, 7.5%, 6.4% and 7.2%, respectively (Fig. 15). This happens because, on average, these prefetchers are not effective in predicting the irregular memory access patterns. For example, with IPCP at L1D and L2C, `mcf` shows 4% degradation compared to a baseline with no prefetching. Other prefetchers improve performance in some of the benchmarks and degrade in the rest, overall providing performance worse than the baseline with no prefetching. We also find that with the presence of ATP prefetcher, conventional data prefetchers prefetch requests for accesses following the replay load (which itself is prefetched by ATP) a bit earlier than the baseline improving the timeliness and hence performance. Note that our enhancements do not affect the performance of applications that do not see significant STLB misses. Our ATP prefetcher is 100% accurate as it is not speculative.

Reduction in ROB stalls. Fig. 16 shows that the performance gain that we achieve is due to the reduction in STLB miss caused ROB stall cycles. Our enhancements successfully reduce ROB stalls due to STLB misses by 28.76% and replay requests by 18.5%, on average. Improved caching of address translations helps in reducing ROB stall cycles due to STLB misses. Further, ATP and TEMPO [11] both help in the reduction of ROB stall cycles due to replay load requests. For `xalancbmk`, the average ROB stall cycles get reduced by 77%, as most of the address translations hit in the on-chip cache hierarchy with a translation MPKI at LLC of only 0.003.

SMT results. Next, we evaluate our work on 2-way SMT environment. For creating mixes of two threads, we divide the benchmarks into three categories with respect to STLB MPKI. Table II shows that if the STLB MPKI of a benchmark is within 10, then it is classified under *Low* MPKI category. If STLB MPKI is in between 11 to 25, then the benchmark is

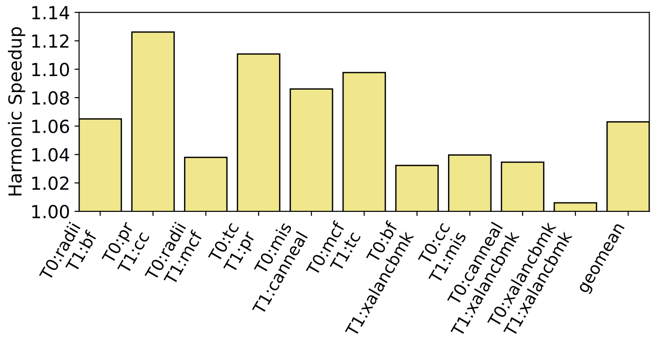

Fig. 17. Normalized speedup with our enhancements for a 2-way SMT core.

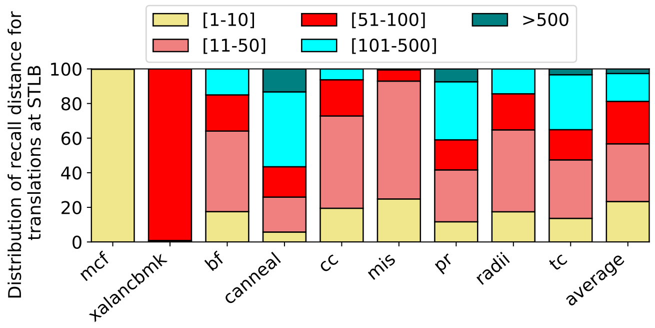

Fig. 18. Recall distance of translations at STLB.

classified as *Medium* MPKI, else *High* MPKI category. We have created 2-threaded mixes by considering all the combinations, for eg. *High-Low*, *High-Medium*, etc. Fig. 17 shows the performance improvement with 2-way SMT mixes. We use harmonic speedup of individual thread to show the total gain in performance for a mix. For every mixes, we list two benchmarks as T0 and T1 that are used in one mix. On average, we find performance gain of 6.3%. Performance gain trend for those mixes which comprises of atleast one benchmark with low or medium STLB MPKI is lower, in comparison to other mixes. For example, canneal-xalancbmk and xalancbmk-xalancbmk yield an improvement of 3.5% and 0.5%, respectively. On the other hand, for radii-bf, pr-cc, and tc-pr the performance gain is 6.5%, 12.6% and 11.1%, respectively.

Multi-core results. We evaluate our enhancements with 25 8-core multi-programmed (homogeneous and heterogeneous) mixes. On average, our enhancements provide an average performance improvement of more than 4%. One of the primary reasons for this trend is with heterogeneous mixes, benchmarks that used to get high translation and replay load misses stay at the LLC, relatively for more amount of time, if the co-running applications do not thrash LLC.

B. Comparison with recent works

STLB management policies fail to reduce ROB stalls because of STLB misses. Chandrashis et al. propose a dead block predictor(cbPred) [18] at LLC using dead page predictor(dpPred) [18] at STLB. The main goal of this paper is to bypass dead translations at STLB and dead data blocks at LLC. The proposal incurs 11KB of additional storage per core. Overall, cache blocks that store replay loads are dead, and bypassing dead blocks does not expedite the ROB stalls because of a replay load access. Figure 18 shows that on average, more than 40% of the TLB entries have a recall distance of more than 50 (dead TLB entries). So, bypassing dead TLB entries do not expedite the costly address translation misses that have a high recall distance. Also, this proposal uses conventional SHiP at the LLC, which is unaware of address translations and replay loads. Our enhancements increase the lifetime of translations at data caches (L2C/LLC) and prefetch replay loads. When we compare our enhancements with CbPred based on DpPred [18], we find that on average, our enhancements further improve average performance by 3.1%. Note that DpPred simulates a conservative core with 128 ROB entries, and its effectiveness drops when we have an aggressive core that uses 352 entries.

CSALT [17] dynamically partitions cache for data blocks and translations based on their hit rates. CSALT also proposes TLB optimizations that are orthogonal to our proposed enhancements. Also, CSALT does not consider the effect of hardware prefetching. On average, CSALT partitioning improves performance by 1% on top of the enhanced SHiP and DRRIP at the L2. We corroborate the findings of CSALT in terms of performance improvement over a weaker baseline that uses LRU policy at the L2 and LLC.

C. Sensitivity Analysis

STLB sensitivity study. Fig. 19 shows the normalized performance of our idea with respect to their corresponding baselines with different STLB sizes. We observe similar performance gains with different TLB sizes as the recall distance of most of the address translations that stall the head of the ROB is high. In mcf, after 2048 entries, there is no further performance gain as all the translations fit in STLB and STLB MPKI drops to 0.39 with 4096 entries. The performance gain reduces with an increase in STLB size, as the scope of retaining leaf-level translations at data caches reduces due to a decrease in STLB MPKI.

L2C sensitivity study. Fig. 20 shows the performance gain with various L2C sizes. We observe that with an increase in L2C size after 512KB(baseline), the average gain in performance is either almost similar(768KB) or decreases(1MB). Note that the access latency of 1MB is higher than the 512KB L2C. With an increase in L2C size, the baseline can retain more translations so, the contribution of T-DRRIP at L2C towards the performance gain reduces. However, we observe marginal performance gain in xalancbmk with an increase in L2C size. For xalancbmk, the contribution of T-DRRIP towards total performance gain is high as even with an 1MB L2C, translation misses are non-negligible. Hence, it is intuitive that the scope of improvement at L2C for this benchmark is high. On the other hand, in canneal the scope of the performance of T-DRRIP is low, hence no further increase in the magnitude of performance. mcf also shows high scope for T-DRRIP, but since the translations fit in L2C as

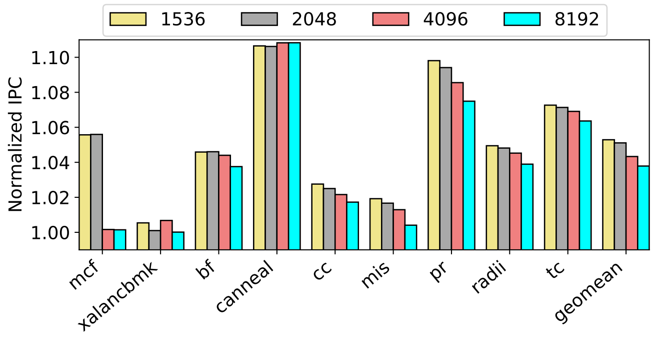

Fig. 19. STLB sensitivity with various STLB entries.

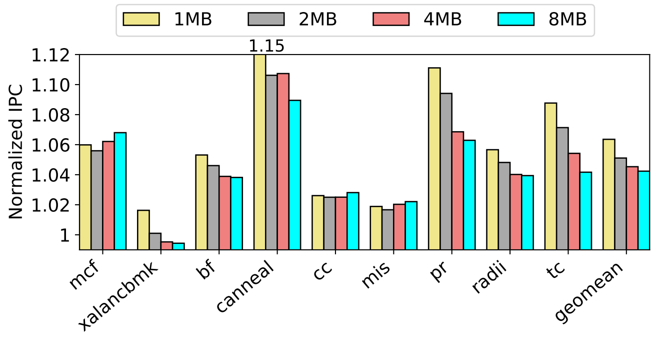

Fig. 21. LLC sensitivity with various LLC sizes.

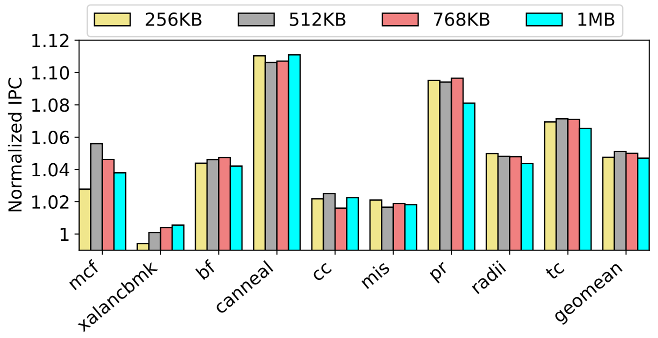

Fig. 20. L2 sensitivity with various L2 sizes.

it grows bigger, the magnitude of performance gain decreases.

LLC sensitivity study. Fig. 21 shows similar sensitivity trend as L2C at LLC, with various cache size. The performance gain decreases with an increase in LLC size from 6.3% at 1MB to 4.2% at 8MB LLC. The influence of ATP and T-SHiP is bigger at LLC than L2C as the number of prefetch requests generated at LLC is higher than L2C. As the LLC size increases, the LLC prefetch hit rate also increases and hence, the gain in performance decreases. In `mcf`, LLC prefetch MPKI does not decrease significantly as the data blocks are yet not fitting in the LLC size. With a large LLC size, the effectiveness of ATP prefetcher increases. Hence, the performance keeps on increasing.

VI. Conclusion

In this paper, We find that an STLB miss can stall the head of the ROB for a maximum of 50 cycles and replay loads for more than 200 cycles. We further discussed that the state-of-the-art cache management policies at the L2 and LLC are ineffective in reducing ROB stall cycles because of an STLB miss and replay loads. We proposed simple enhancements in the form of T-DRRIP and T-SHiP for address translation aware cache hierarchy management. Our enhancements reduce the number of ROB stall cycles by reduced by 28.76%. For mitigating latency because of replay loads, we propose ATP that reduces ROB stall cycles by additional 18.5% by prefetching replay loads into the cache hierarchy. Overall, these enhancements contribute to an average performance improvement of 5.1%.

VII. Acknowledgements

The authors would like to thank Mainak Chaudhuri, Shankar Balachandran, and Anant Nori for their valuable feedback on the earlier draft. This work is supported by the SRC grant SRC-2922.001.

References

[1] "Examining intel's ice lake processors: Taking a bite of the sunny cove microarchitecture," Available at https://www.anandtech.com/show/14514/examining-intels-ice-lake-microarchitecture-and-sunny-cove.

[2] "Ligra benchmarks," Available at https://github.com/jshun/ligra.

[3] "Parsec benchmarks," Available at https://parsec.cs.princeton.edu.

[4] "Spec benchmarks," Available at https://www.spec.org/cpu2017/.

[5] "Sunny cove microarchitecture," https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove.

[6] "The 2nd data prefetching championship (dpc-2)," Jun. 2015. [Online]. Available: https://comparch-conf.gatech.edu/dpc2/

[7] "The 2nd cache replacement championship (crc-2)," Jun. 2017. [Online]. Available: https://crc2.ece.tamu.edu/

[8] "The 3rd data prefetching championship (dpc-3)," Jun. 2019. [Online]. Available: https://dpc3.compas.cs.stonybrook.edu/

[9] "The 1st instruction prefetching championship (ipc1)," May 2020. [Online]. Available: https://research.ece.ncsu.edu/ipc/

[10] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 399–411.

[11] A. Bhattacharjee, "Translation-triggered prefetching," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 63–76.

[12] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 247–259.

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

[14] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 60–71, 2010.

[15] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 175–186.

[16] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[17] Y. Marathe, N. Gulur, J. H. Ryoo, S. Song, and L. K. John, "Csalt: Context switch aware large tlb," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 449–462.

[18] C. Mazumdar, P. Mitra, and A. Basu, "Dead page and dead block predictors: Cleaning tlbs and caches together," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 507–519.

[19] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 118–131.

[20] D. Tarjan and K. Skadron, "Merging path and gshare indexing in perceptron branch prediction," *ACM transactions on architecture and code optimization (TACO)*, vol. 2, no. 3, pp. 280–300, 2005.

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

[22] C. Wu and M. Martonosi, "Characterization and dynamic mitigation of intra-application cache interference," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2011, 10-12 April, 2011, Austin, TX, USA*. IEEE Computer Society, 2011, pp. 2–11. [Online]. Available: https://doi.org/10.1109/ISPASS.2011.5762710

[23] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "Imp: Indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 178–190.