Fooling the Sense of Cross-core Last-level Cache Eviction based Attacker by Prefetching Common Sense

Biswa Banda Panda
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
biswap@cse.iitk.ac.in

Abstract—Cross-core last-level cache (LLC) eviction based side-channel attacks are becoming practical because of the inclusive nature of shared resources (e.g., an inclusive LLC), that creates back-invalidation-hits at the private caches. Most of the cross-core eviction based side-channel attack strategies exploit the same for a successful attack. The fundamental principle behind all the cross-core eviction attack strategies is that the attacker can observe LLC access time differences (in terms of latency differences between events such as hits/misses) to infer about the data used by the victim. We fool the attacker (by providing LLC hits to the addresses of interest) through a back-invalidation-hits triggered hardware prefetching technique (BITP). BITP is an L2 cache level hardware prefetcher that prefetches the back-invalidated block addresses and refills the LLC (along with the L2) before the attacker’s observation/access, efficiently nullifying inferences due to differences in access latencies.

We show that BITP can fool the attacker with various security metrics related to LLC side-channel. BITP provides zero probability of success in terms of attacker’s probability of success for Evict+Time, Evict+Reload, and Prime+Probe attacks. We also show the effectiveness of BITP in terms of performance by simulating SPEC CPU 2006, PARSEC, and CloudSuite benchmarks and find that, on average, BITP improves system performance marginally by 1.1%. Overall, BITP is a simple, practical, and yet powerful technique in mitigating various cross-core LLC eviction-based side-channel attacks. Compared to the state-of-the-art policies, BITP does not require support from software writer, operating system (OS), and runtime systems. Overall, BITP provides marginal improvement in system performance, providing security with no hardware and performance overhead, which makes BITP readily-implementable.

I. INTRODUCTION

Cross-core eviction based side-channel attacks at the last-level cache (LLC), observe the fundamental property of “latency differences between cache hits and misses” to infer about the cache blocks that are accessed by the victim (cryptographic) application [1]–[6]. An LLC eviction attack includes an attacker (spy) application running along with a victim application on a multi-core system. The attacker is a malicious application that tries to infer the secret data. As all the cores of a system usually share the LLC, an attacker tries to evict with the victim at the LLC set and fools the victim by employing different cross-core side-channel attack strategies. These strategies observe hits/misses to the cache block addresses of interest of the victim. There are various cache eviction attacks that are mounted on mobiles [7], desktops [5], and clouds [4].
system performance and system fairness and some of the proposals are very specific to a particular attack. Some of them [15], [16] demand changes at the OS level.

**Key Observations:** The following observations motivate our proposal.

(I) Back-invalidation-hits help attackers: In an evicted based cross-core side channel attack, the attacker’s premise is that the evicted LLC block is present in private caches. Therefore, in such a scenario, back-invalidations originating from the LLC, hit at the private cache of the victim and invalidates the corresponding blocks. This leads to private cache misses for future accesses to the same blocks by the victim and future accesses load the same block from the LLC.

(II) Cross-core Back-invalidation-hits are rare and benign: The fraction of back-invalidations that hit at the private caches is low if the per-core L2:L3 ratio is low and the back-invalidated blocks are “hot” (get reused) [17]. So, prefetching back-invalidated blocks will not degrade the system performance.

While the first observation is the essence of the attack scenario, the second observation requires empirical validation for strengthening the claim and quantifying the performance overhead. To establish “back-invalidation-hits help attackers”, we simulate a 2-core system mounting the Evict+Reload attack, where a spy runs on core-0 and a victim runs on core-1 with cryptographic applications such as GnuPG [18] and Poppler [19]. We find that all the block addresses of interest cause back-invalidation-hits at the L2. We describe the details about different attack strategies in Section II. We also experiment with other ciphers like AES-128 and RSA, and our conclusion remains the same.

To establish that “Cross-core back-invalidation-hits are rare and benign”, we quantify the fraction of back-invalidations that hit at L2. We perform this study with two of the best cache replacement policies: SHiP++ (an extended SHiP [20]) and a modified version of HAWKEYE [21] as per the 2nd cache replacement policy [17] held in ISCA 2010. In contrast, we observe a similar trend with other replacement policies known cross-core LLC eviction based side-channel attacks by the authors have used an LLC of 1MB/core, motivated by the 1st cache replacement championship (CRC-1) [27] held in ISCA 2010. In contrast, we consider 2MB/core LLC for the reasons already mentioned. Note that the expected back-invalidation-hit ratio is around $\frac{256KB}{3MB}$ for L2 and with a 1MB LLC/core, this ratio goes up to around $\frac{256KB}{1MB}$. We corroborate the findings of [17] for different L2/LLC ratios.

We also try LRU based replacement policies where the percentage of back-invalidation-hits is small because policies such as SHiP++ and HAWKEYE are more aggressive than LRU in evicting cache blocks of cache-averse applications (applications that do not get benefit with LLC). Note that, we use the gem5 [25] full-system simulator and simulate the SPEC CPU 2006 benchmarks for 250M instructions within their respective region of interest after a fast-forward of 200M instructions. We use the region of interests similar to the CRC-2 site. This experiment sets the tone for our proposal as the fraction of back-invalidations that hit at the L2 is marginal and a large fraction of them get reused.

**Our Idea:** We propose a simple per-core private hardware prefetcher at the L2 level and name it back-invalidation-triggered-prefetching (BITP). Note that, no other invalidation hits such as invalidation hits while maintaining cache coherence, trigger BITP. BITP prefetches the back-invalidated block addresses and it does not maintain any additional hardware structure to prefetch. To the best of our knowledge, this is the first proposal on hardware prefetching that mitigates well-known cross-core LLC eviction based side-channel attacks by exploiting the notion of back-invalidation-hits. Overall, our contributions are as follows:

- We quantify the back-invalidation-hits for cryptographic and standard applications (Figure 1) and propose BITP that brings the back-invalidated blocks into L2 and LLC. We provide the security effectiveness of BITP and show how BITP mitigates cross-core LLC eviction attacks. We also discuss few subtle issues of interest. (Section III).
- We show the effectiveness of BITP in terms of system performance (an average improvement of 1.1%) and provide security with no hardware overhead, and no support from ISA, compilers, runtime systems, and OS (Section IV).

![Fig. 1. Fraction of LLC evictions that result in back-invalidation-hits at the L2 for single-core and multi-core systems with 2MB/LLC per core.](image-url)
II. BACKGROUND

This section provides background on different cross-core eviction based side-channel attacks (miss type and hit type) at the LLC. It also provides a discussion about some of the recent LLC replacement policies. In miss type attacks, the attacker is interested in observing longer cache access time, because of cache misses (miss access can be either from the victim or the attacker). In contrast, in hit-based attacks, the attacker is interested in shorter access time (hits). All the attacks measure LLC access time. However, some attacks do it precisely per memory access (access based attacks), and some accumulate the timing information for the entire security-critical accesses (timing based attacks). Primarily, there are three different strategies such as (i) Evict+Reload (a variant of Flush+Reload attack where the Flush operation is replaced by the Evict operation), (ii) Evict+Time, and (iii) Prime+Probe. In flush based attacks such as Flush+Reload [5], the attacker uses clflush instruction to flush a cache block address from all the cache levels and later reloads the same block address. While reloading, if it gets a hit, then the attacker concludes that the victim has accessed the cache block.

Note that, based on the prior works suggest flush based attacks [5], [28]-[30] can be mitigated by preventing clflush instruction in user mode for read-only or executable OS pages (such as shared library code) [16]. It can be done through a system call (Linux OS already has a system call called cacheflush [31]). There are other possibilities like making clflush constant time, or the extreme case like Google NaCl [32] that disables clflush instruction. However, it can be noted that x86 still allows clflush from the user mode. We believe there are undisclosed reasons for which clflush is not privileged yet and it is an open problem to debate and discuss, which is beyond the scope of this paper. In this paper, we only concentrate on eviction based attacks.

Evict+Time: In this attack, the spy observes the execution time of the victim over a large number of intervals. First, the spy evicts cache blocks from a few set(s) at the LLC that causes back-invalidation-hits in the private L2 of the victim. Later, when the victim accesses the evicted block(s), it results in a longer access time. The spy observes the same.

Evict+Reload: In Evict+Reload attack, the spy core evicts a cache block from the LLC that results in a back-invalidation and invalidates the corresponding cache blocks in private L2 of the victim. After an interval (predetermined fixed value), the spy reloads the same address and if it gets a shorter access time (an LLC hit), then it concludes that the victim has accessed the same cache block.

Prime+Probe: In this attack, the attacker loads its cache blocks by evicting the blocks of the victim (the prime part). Then the victim executes its secure operation and in the process, gets LLC misses, evicts the blocks brought by the attacker. Next, the attacker probes its execution time by reloading its blocks, to see whether it gets longer access time because the victim has evicted the block (an LLC miss).

Out of all these attacks, Evict+Reload attack demands the notion of sharing of OS pages between the victim and the spy. The attack is more precise (operates at specific block addresses).

The notion of time: In all the cross-core eviction based attack strategies, the attacker uses monitor epochs of 5000 to 10,000 cycles [2], [16], [33], and [10]. In one epoch, the attacker evicts (or primes) 16 cache blocks (assuming a 16-way LLC) at the beginning of the epoch, waits, and reloads (or probes/observes) LLC block(s) of interest, just before the end of the epoch. Yarah and Falkner [5] show how to choose the time gap (length of the epoch) so that an attacker can attack successfully. The next section shows how BITP mitigates various cross-core LLC side-channel attacks.

Cache Replacement Policies: LLC replacement policies play an important role in setting up the eviction based attacks because to evict a block from a cache set, the attacker has to access the set multiple times to make sure the victim’s block is evicted from the LLC. As LRU based policies are not that effective for large LLCs, aggressive LLC replacement policies such as SHiP++ [20] and HAWKEYE [21] have been proposed that use re-reference interval prediction (RRIP) [34] chain based policies with re-reference prediction values (RRPVs). SHiP++ uses difference signatures like the program counter (PC) and memory region to infer about the reuse of the blocks belonging to that signature and HAWKEYE tries to provide an illusion of Belady’s optimal replacement policy. It looks at the past behavior of cache blocks based on a signature like PC and applies Belady’s policy on them.

III. BACK-INVALIDATION-HITS
TRIGGERED PREFETCHING (BITP)

A. BITP Mechanism

A self-contained Figure 2 shows the steps involved with the BITP mechanism. BITP only prefetches on back-invalidation hits and not on invalidations due to cache coherence. Note that in case of a baseline system without BITP, the LLC controller sends a normal invalidation command (INV) along with the evicted address to the private caches. With BITP, we need a mechanism to distinguish back-invalidations from normal invalidations. To accomplish this, the LLC controller sends a packet with BACK-INV command (similar to other commands like GET/PUT/INV/LOAD/STORE/PREFETCH/WRITEMACK) along with the evicted block address in the command+address bus. The private cache controllers would trigger BITP if there is a back-invalidation hit (by comparing the tag) and the command is BACK-INV and not INV. Also, depending on the implementation of cache coherence directory (e.g., a sliced directory for each slice at the LLC), the evicted LLC block address along with the BACK-INV command, should be communicated to the sliced directory first, which converts the address and the command into back-invalidation requests for private caches. So, overall, BITP demands marginal changes to existing structures and does not demand any additional hardware.

1Note that this epoch is used in real machines.
To compare different micro-architecture techniques in terms of information leakage, metrics such as true positive rate (TPR), which is the ratio of true critical accesses observed by the attacker and the number of critical accesses of the victim and Cache side-channel vulnerability (CSV) [35] (Pearson’s correlation coefficient between the victim and attacker traces at the LLC) are proposed. Recently, He and Lee proposed a nice and more generic model called Probabilistic information flow graph (PIFG) [36] to quantify the probability of attack success (PAS). A PAS value closer to 0 is better and secure. We apply PIFG [36] to include the events of interest for an inclusive LLC. We redefine PAS for an inclusive LLC for Evict+Time, Prime+Probe, and Evict+Reload attacks by adding one additional event of back-invalidation-hit. Overall, we show the effectiveness of BITP with the following metrics: (i) PAS, (ii) Relative LLC access time difference as observed by the attacker, (iii) TPR, and (iv) CSV. Out of these four metrics, PAS is a recent one, which we explain in details.

**B. Metrics for Security Effectiveness**

PAS [36]: Table I shows conditional probabilities of interest through which the information flows from the victim to the attacker, for all three cross-core eviction based attacks at the LLC. For a detailed overview on PIFG, please refer [36]. We quantify PAS for a baseline system with 32KB L1, 256KB L2, and 16-way 2MB LLC slice/core (similar to Intel’s slicing at the LLC [37]) by finding out the probabilities (Table I):

<table>
<thead>
<tr>
<th>Events</th>
<th>Probability (PAS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1: Memory block getting mapped into a cache set</td>
<td>1.00 (BITP provides hits), which results in a PAS of 0.</td>
</tr>
<tr>
<td>p2: Cache block selected for replacement given the cache set</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p3: Evicted cache block leads to back-invalidation-hits at the L2</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p4: Evicted block (that has caused back-invalidation-hit) when accessed again gets an LLC miss and the very next access gets a hit.</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p5: LLC hit/miss getting mapped to the shorter/longer access time.</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
</tbody>
</table>

**C. PAS of BITP**

1) **PAS for Evict+Time attack with BITP**: As Evict+Time is a miss type and timing based attack, the attacker will be successful if it observes longer access time to the block addresses of interest that are evicted by itself. The only conditional probability that changes with BITP is \( p_5 \), which becomes 0 as the probability of victim’s reload getting a miss is zero (BITP provides hits), which results in a PAS of 0.

2) **PAS for Evict+Reload Attack**: In contrast to Evict+Time attack, Evict+Reload is a hit type attack. An attacker goes through conditional probabilities of \( p_1 \) to \( p_4 \) (same as Evict+Time). \( p_5 \) corresponds to the attacker’s reload is a hit provided the victim has accessed the cache block between evict and reload. Note that, PIFG calculates the forward probability from the victim’s side and because of which it can not capture the effectiveness BITP as it does not consider the cases where the victim has not accessed and the attacker still gets the hits. A formal way of finding the PAS for this attack is to find the backward probability from the attacker’s point of view and till the point that a cache set is mapped to the memory block. A simple alternative is \( p_5 \) can be exactly correlated with the TPR, which is 1.00 in the baseline. With BITP, \( p_5 \) is PV (probability of the victim’s access at the LLC in between evict and reload). Note that, there are two possibilities:

(i) The victim gets a hit at the L2 thanks to BITP and there is no access to LLC. In this case PV and \( p_5 \) are 0, which happens all the time with BITP.

(ii) It is still possible that the victim gets a miss at the L2 and accesses LLC with probability PV and in this case, BITP provides LLC hits all the time to the attacker. A PAS value closer to 0 is better and secure. We apply PIFG [36] to include the events of interest for a baseline system with 32KB L1, 256KB L2, and 16-way 2MB LLC slice/core (similar to Intel’s slicing at the LLC [37]) by finding out the probabilities (Table I):

<table>
<thead>
<tr>
<th>Events</th>
<th>Probability (PAS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1: Memory block getting mapped into a cache set</td>
<td>1.00 (BITP provides hits), which results in a PAS of 0.</td>
</tr>
<tr>
<td>p2: Cache block selected for replacement given the cache set</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p3: Evicted cache block leads to back-invalidation-hits at the L2</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p4: Evicted block (that has caused back-invalidation-hit) when accessed again gets an LLC miss and the very next access gets a hit.</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
<tr>
<td>p5: LLC hit/miss getting mapped to the shorter/longer access time.</td>
<td>0.125 (p1 X p2 X p3 X p4 X p5 X p6)</td>
</tr>
</tbody>
</table>

3) **PAS for Prime+Probe Attack with BITP**: In Prime+Probe attack, first, the attacker evicts blocks of interest of the victim (step 1) and then the victim misses and evicts the blocks of interest of the attacker (step 2). Later, when the attacker probes, it gets an LLC miss (longer access time). In terms of PAS,
ALGORITHM 1: Square Multiply Exponentiation

1: **Input:** base \( b \), modulo \( m \), and exponent \( e \) (\( e_{n-1} \) to \( e_0 \))
2: **Output:** \( b^{e} \mod m \)
3: \( r = 1 \)
4: **for all** \( i \), from \( n-1 \) to 0 **do**
5: \( r = \text{square} (r) \)
6: \( r = \text{modulo} (r, m) \)
7: **if** \( (e_i == 1) \) **then**
8: \( r = \text{multiply} (r, b) \)
9: \( r = \text{modulo} (r, m) \)
10: **end if**
11: **end for**
12: return \( r \)

The attacker measures the access time to infer about LLC hits and misses, and with BITP, the attacker should get the addresses corresponding to the square and multiply functions (addresses of interest).

The attacker makes the access time to infer about LLC hits and misses, and with BITP, the attacker should get the time closer to the hit latency of the LLC. Figure 4 shows the LLC access time for the critical accesses (square and multiply functions) as observed by the attacker within a representative window of 50 epochs (epoch \# from 1000 to 1050). For illustration purpose, we pick a small window. However, we observe a similar trend for the rest of the epochs. From Figure 4, in the baseline, the attacker can differentiate the LLC hits and misses by observing the access latencies and can infer the secret key bit (if multiply follows the square in one epoch, then the bit is one else zero). However, with BITP, the attacker gets access time closer to LLC hits to the addresses of interest as BITP prefetches them before the attacker reloads. Note that, there is no latency difference between an LLC hit to a demand cache.
3) **Prime+Reprime+Probe attack on Poppler:** This section explains the effect of BITP on a pdf rendering library called Poppler [19]. We run the attacker on core-0 and **pdf2ps** on core-1. The approach used in [41] that attacks four different functions of **pdftops** motivates our attack. We show the LLC hits for each function with the baseline system and with BITP in Figure 5. More details about the attack are available in [41] where the authors describe how they probe the addresses of interest (four functions) to identify more than 100 pdf files. The attacker uses we trigger BITP on every eviction at the extended coherence (back-invalidation hits) at the L2. Similar to the inclusive LLC, blocks at the LLC are not present in its private L2. The attacker ensures that the reprime process keeps the prime data in the LLC, which makes a solid case for BITP.

**Probability distribution of LLC access time:** Figure 6 shows the probability distribution function of LLC access time averaged across all the cross-core LLC-eviction attacks. Note that, with BITP, the attacker gets access time closer to LLC hits. LLC misses have two components: DRAM row-buffer hits and row-buffer conflicts. Table II summarizes the effectiveness of BITP in terms of four different metrics related to all the three cross-core cache side-channel attacks. Note that, with PIFG model (a mathematical model), PAS of BITP is zero. However, the range of values of TPR (0.04 to 0.11) and CSV (0.07 to 0.13) are non-zero (though closer to zero) because of noise that comes from the experiments.

**E. Eviction Attacks in Non-inclusive LLCs**

A recent work that will appear in SP ’19 [8] shows cross-core eviction attacks in non-inclusive caches. The premise of the attack is ”shared inclusive, and extended sliced cache coherence directory”. The authors exploit the inclusive directory to create cross-core evictions that create inclusion victims (back-invalidation hits) at the L2. Similar to the inclusive LLC, we trigger BITP on every eviction at the extended coherence directory that creates back-invalidation hits. So, fundamentally, any attack that creates back-invalidation-hits will be mitigated by our approach (irrespective of inclusive and non-inclusive LLC), which makes a solid case for BITP.

**F. BITP with Intelligent Attackers**

We discuss some other ways of launching a cross-core LLC eviction attack and how BITP handles them.

**Prime+Reprime+Probe attack [42]:** An attacker can launch a Prime+Probe attack, where the attacker primes the LLC and reprimes the LLC to make sure the primed cache blocks at the LLC are not present in its private L2. The attacker ensures that the reprime process keeps the prime data to a new cache set in the LLC, but to the same cache set in the L2s. In case of systems that use huge pages (OS page size of 1MB and 1GB) it is relatively easy because 20 to 30 bits (for 1MB and 1GB pages) of page offset will not change during the page translation. Depending on the cache indexing, the attacker can evict the cache blocks only from the L1/L2 caches while leaving them in the LLC. In this case, there will not be back-invalidation-hits at the attacker’s L2. However, back-invalidation-hits will be there at the victim’s L2 in both

---

**Fig. 5. Shorter/longer access time observed by the attacker in terms of 0s/1s while attacking critical functions of Poppler. We do not show the “1s” explicitly.**

**Fig. 6. Probability distribution function of LLC access time of critical accesses across all the attacks with the (a) baseline and (b) BITP.**

**Fig. 7. True positive rate (TPR) versus epoch length.**

---

**TABLE II**

<table>
<thead>
<tr>
<th>Summary of security results across all attacks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS</td>
</tr>
<tr>
<td>Avg. access time</td>
</tr>
<tr>
<td>Range of TPR</td>
</tr>
<tr>
<td>Range of CSV</td>
</tr>
</tbody>
</table>
the Prime and Reprime steps, which trigger BITP from the victim’s L2 and makes sure that the victim gets L2 and LLC hits, no eviction of attacker’s blocks. So, in the Probe step, the attacker gets LLC hits because its blocks are not evicted by the victim.

**Invalidation+Evict+Reload attack**: There are a few ciphers that updates (writes) and reads the data. In this case, the attacker may first invalidate the victim block(s) in the victim’s L2 through cache coherence, and then evict its blocks from its private caches and then from the LLC. In such cases, the LLC eviction will not cause back-validation hits. However, this methodology is non-deterministic and impractical as explained below. After the invalidation (STORE access from the attacker) of the victim’s private block, the attacker would have the same block in the M state (assuming MOESI protocol) at its L1.

To ensure that the block is not present in the entire cache hierarchy, the attacker does the following: (i) evicts the block from L1 (4 accesses for a 4-way L1, being dirty, block would enter the L1 write-back queue), block gets updated at the L2 if needed, (ii) evicts the updated block from L2 (8 accesses for an 8-way L2), block would enter L2 write-back queue, and the block gets updated at the LLC if needed, and finally (iii) evicts the block from the LLC (16 accesses for a 16-way LLC). Note that there would be a gap between L1 eviction and L2 update from the L1 write-back queue, and same at the LLC level. We find this attack is impractical because in the best case, the attacker has to perform 29 to 50 accesses to different levels of caches. We perform the same on Intel Skylake and Intel Haswell machines.

### G. Revisiting The Notion of Time

**Sensitivity to the epoch length**: In Section II, we have discussed the length of the epochs (5000 to 10000 cycles). We also test shorter (starting from zero cycles) and longer monitor intervals for the effectiveness of BITP. As mentioned in Section III-F, the victim accesses the LLC after an interval, which is of 2000 cycles. So, fundamentally, the attack will not be effective even in the baseline case if the attacker uses an epoch of less than 2000 cycles. For example, in a Prime+Probe attack, the attacker has to evict 16 cache blocks of a given cache set then the victim accesses and evicts few blocks, and then again the attacker has to access the block addresses of interest. Figure 7 shows the effect of epoch length on TPR across all the eviction attacks and we find an epoch length between 6000 cycles to 10,000 cycles is the best for most of the cross-core eviction based LLC attacks. This shows the effectiveness of BITP as small epoch length will make an attack weak and with a good enough epoch length, BITP makes the attack weak.

**Time gap between back-validation hit and prefetch response**: This time gap should be less than the gap between back-validation-hit and the victim’s access. We find, there is an average gap of just more than 2000 cycles between the Evict or Prime step and the victim’s access. That is why the attacker chooses a long epoch. If the attacker chooses a small epoch of say 2000 to 4000 cycles, then most of the time the attacker will miss at the LLC. We find two insights: (i) It is sufficient to prefetch anytime in between Evict and the Reload of the attacker for all the miss-type attacks. (ii) However, for the hit-type attacks, the prefetcher should prefetch before the victim accesses, and makes sure that by the time the victim accesses it finds hits at the L2 (no access to LLC from the attacker corresponds to no information leakage at the LLC).

Based on our simulations, in the best/worst case, a prefetch response takes 72/323 processor cycles (averaged among 4-core, 8-core, and 16-core simulations with one, two, and four DRAM controllers).

**Motivated attacker**: Note that if a motivated attacker reloads during the prefetch response interval (for a fixed epoch as shown in Figure 7), then it will be unsuccessful. If an attacker knows about BITP and tries to reload just after the eviction then it would be successful with a TPR of less than 0.002 (TPR of the baseline system is 0.9). To make our case even stronger, we run all the attacks on real machines (on Intel Skylake and Intel Haswell) where once we finish evictions of all the blocks, we reload immediately creating a multi-threaded attacker, and find even a lower TPR. We find two scenarios dominating this experiment: (i) Attacker reloads before victim’s access and (ii) an overlap between attacker’s reload and victim’s access.

### H. Security Comparison with Recent Works

SHARP prevents cross-core eviction of blocks that create back-invalidation hits and hence prevents cross-core side-channel attacks at the LLC by sending queries to L2 and probing the coherence directory. SHARP does not allow a spy to perform cross-core eviction if the eviction results in inclusion victims (back-invalidation-hits) at the L2 of the victim. To realize that, before evicting a cache block from an LLC set, SHARP-4 sends up to four queries (4 block addresses based on the replacement priority order, for example, LRU to LRU-3 positions if the LLC uses LRU replacement policy) one by one to the L2 cache. The moment it finds that a query does not create an inclusion victim then it evicts the block from the LLC. In the worst case, if all the four queries fail to provide a block that prevents inclusion victims; it uses the coherence vector to find out if the rest of the blocks that are present in the set will cause inclusion victim. In the rare case, SHARP evicts a block randomly and if the # random evictions cross a threshold, then it raises an interrupt to the OS.

**RIC** [15] is a relaxed inclusive cache hierarchy that prevents back-invalidations of thread-private data and read-only data. RIC takes the help of system software (OS) to identify the read-only pages and it augments an additional bit (relaxed inclusion bit) per cache block to identify the read-only block. During an eviction, if the relaxed inclusion bit of the block is set, then RIC does not back-invalidate private caches. RIC is simpler (in terms of design aspects) compared to SHARP.

**Security**: Both SHARP and RIC provide the same level of protection as BITP. SHARP and RIC fool the attacker.
by preventing back-invalidation-hits, (in terms of PIFG, by making p4 zero), resulting in PAS of zero. We compare SHARP and RIC with BITP in terms of PAS, LLC access time, TPR, and CSV, and find that all these techniques are equally effective. Table III shows the subtle issues that are involved with SHARP and RIC and why BITP scores better equally effective. Table III shows the subtle issues that are involved with SHARP and RIC and why BITP scores better.

**IV. Performance Evaluation**

**A. Simulation Methodology**

We use the x86 based gem5 [25] simulator to simulate single-core SPEC CPU 2006 [22] benchmarks and multi-core (4-core to 16-core) multi-programmed mixes. To simulate the CloudSuite [24] benchmarks, we use the CRC-2 framework that provides traces of CloudSuite benchmarks. Table IV shows the parameters used in our simulated system. Note that for multi-core mixes, the shared resources are scaled to prevent resource constraints. For simulating CloudSuite benchmarks, we use a modified version of ChampSim [44] interfaced with DRAMSim2 [45]. We simulate the region of interest for 250M instructions with a warm-up of 200M instructions. For CloudSuite benchmarks, we use the 100M traces as provided by the CRC-2. For PARSEC, we use the sim-medium input set and simulate the region-of-interest. For multi-programmed mixes, we continue our simulation till the slowest application finishes its 250M instructions (same methodology as prior works such as [17]). However, we report the results only for the region of interest of each application. Table V classifies all the SPEC CPU 2006 and CloudSuite benchmarks into three categories: (i) L2 fitting (working set fits in L2), (ii) LLC fitting (working set fits in LLC), and (iii) LLC thrashing (working set thrashes LLC) as used in [17].

**Metrics:** We use the L2+LLC misses per kilo instruction (MPKI) to measure the reduction or increase in the L2+LLC misses. For single-core simulations, we use speedup as the metric, i.e., \( \frac{\text{Exectime}_{\text{baseline}}}{\text{Exectime}_{\text{technique}}} \). For multi-programmed mixes, we use harmonic mean of speedups (fair-speedup (FS)) [46].

\[
FS = \frac{1}{\sum_{i=1}^{N} \frac{1}{\text{IPC}_{i}^{\text{alone}}}}
\]

where \( \text{IPC}_{i}^{\text{alone}} \) is the IPC of core \( i \) when it runs alone with other \( N-1 \) applications and \( \text{IPC}_{i}^{\text{together}} \) is the IPC of core \( i \) when it runs alone on a \( N \)-core multi-core system.

**B. Single-core and Multi-core Results**

**Single-core results:** Though, BITP is effective in mitigating cross-core side-channel attacks at the LLC for multi-cores, it is essential to report its effect on single-core simulations as single-core performance should not be compromised for cross-core security. With BITP, improvement/degradation in...
performance depends on LLC MPKI, the fraction of LLC evictions that cause back-invalidation-hits and their reuse.

(i) L2 fitting applications do not contribute to LLC accesses, and hence LLC misses and back-invalidations. The fraction of back-invalidations-hits are close to zero. So the performance improvement is negligible.

(ii) LLC fitting applications evict cache blocks rarely at the LLC, which causes LLC back-invalidations and their corresponding hits, also rare (< 1%), causing a negligible impact on the IPC.

(iii) LLC thrashing applications miss significantly at the LLC, which causes significant back-invalidations. However, again the back-invalidation-hits are marginal (less than 7% in most of the benchmarks). So BITP brings prefetched blocks for 7% of total LLC evictions improving performance by 2.19% only. In summary, for single-core simulations, BITP has no impact on system performance for L2 fitting and LLC fitting benchmarks. It improves performance by an average 2.19% for LLC thrashing applications, only.

Multi-core results: For multi-core evaluation, We create 120 representative 4-core mixes (15 from each type as mentioned in Table VI). We pick 15 mixes from each type to get a cohesive picture of BITP. We also create 50 and 25 8-core and 16-core representative mixes, respectively.

Based on the single-core performance results, it is expected that multi-programmed mixes that contain L2-fitting applications along with LLC fitting or LLC thrashing applications would get performance benefit because L2-fitting applications’ blocks will be back-invalidated by LLC-fitting applications. We observe and validate this expected trend.

Figure 8 shows the effect of BITP on LLC+L2 misses and fair-speedup. On average (across all 4-core mixes (31,465 mixes, 28 choose 4 with repetitions)), the effect of BITP is marginal on LLC misses (average reduction of 2% and 3% with SHiP++ and HAWKEYE) and fair-speedup (average improvement of 1.1% with SHiP++ and HAWKEYE). There are few mix types like 0.5L2F-0.5LLCF, 0.25LLCT-0.75L2F, and 0.5L2F-0.5LLCT, where BITP improves the system performance by 3%, in which one application evicts blocks of other applications, aggressively. With BITP, these evictions cause back-invalidation-hits that cause prefetching of the corresponding blocks, resulting in subsequent L2/LLC hits. For example, in one of the mixes, with HAWKEYE, BITP improves performance close to 3%, where LLC thrashing applications are lbm and mcf (more than 99% of cache blocks of zero reuse). So with BITP, performance of lbm and mcf does not increase. However, the cross-core evictions caused by lbm and mcf that have resulted in back-invalidation-hits, get allocated again for benchmarks like h264ref and sjeng.

Similarly, there are mixes that contain LLC fitting and LLC thrashing applications only, where the effectiveness of BITP is marginal. In few mixes, where the reuse of back-invalidated blocks is low, it increases LLC misses by polluting the LLC (bringing in cache blocks that get no further cache hits) causing performance degradation of 0.03%. Overall, BITP improves performance marginally.

Moving from 4-core to 8/16-core systems: BITP scales well with large core count as there is no hardware overhead. Also, the effectiveness remains the same (average performance improvement of less than 1%) even with 8-core and 16-core mixes as well. Figure 9 shows performance improvement with 8-core and 16-core multi-programmed mixes. Apart from the effect of reuse of back-invalidated blocks, few mixes get affected because, with BITP, the miss access pattern is different at the DRAM compared to the baseline, which causes a marginal increase/decrease in the LLC MPKI. Overall, on average, BITP does not affect system performance and scales well, which makes BITP a simple, scalable, yet effective choice for mitigating cross-core side-channel attacks at the LLC.

CloudSuite and PARSEC benchmarks: The effectiveness of BITP remains the same (average performance improvement of less than 0.5%) with CloudSuite benchmarks too. Figure 10 shows the reduction in LLC misses and improvement in the execution time. We do not report fair-speedup for these benchmarks as these are system workloads and multi-threaded in nature with synchronization primitives that affect the actual instruction count. As expected, the applications that get penalized because of back-invalidation-hits get the maximum improvement. Figure 11 shows the effectiveness of BITP on for 8 and 16-threaded parallel applications from the PARSEC benchmark suite. The trend remains the same for PARSEC
Lower the better
0.975
0.98
0.985
0.99
0.995
1
1.005
1.01
1.015
1.02
1.025
SHiP+BITP over SHiP
HAWKEYE+BITP over HAWKEYE
(a) L2+LLC MPKI reduction normalized to the baseline
Higher the better
0.99
0.995
1
1.005
1.01
1.015
1.02
1.025
SHiP+BITP over SHiP
HAWKEYE+BITP over HAWKEYE
(b) Speedup normalized to the baseline
Fig. 10. Normalized L2+LLC MPKI and speedup for 4-core CloudSuite benchmarks. _px: phase x. SHiP:SHiP++

Fig. 11. Normalized speedup averaged across 8-threaded/16-threaded PARSEC applications.

also (avg. improvement of just 1.09%) as in some applications it improve the execution time by bringing back the shared data into the cache hierarchy.

Energy Consumption: Figure 12 shows the normalized energy consumption with BITP. We use CACTI 6.5 [47], DRAM Micron power model [48], and Orion 2.0 [49] for modeling energy related to caches, DRAM, and interconnect, respectively. Compared to the baseline, there is a slight variation with the maximum overhead of 1.9% for PARSEC benchmarks, which is because of interconnect traffic that comes from back-validations of S state blocks at the LLC, which leads to multiple back-invalidation hits and multiple prefetch requests. However, Beyond LLC, all these requests merged into one prefetch request.

Sensitivity studies: So far, we use L2:L3 ratio of 1:8. When we bridge the difference between L2 and LLC capacity (LLC is just 256KB or 512KB per core, with L2:L3 ratio of 1.00 and 0.5, which is opposite of the current trend as the shared resources are more constrained), the fraction of back-invalidations that hit at L2 become significant. As expected, with BITP, the performance improves significantly (average improvement of more than 20% and 10%, respectively). Figure 13 (a) shows the improvement in performance with different L2:L3 ratios averaged across all multi-core mixes, which corroborates the conclusions of TLA [17]. So, for smaller LLCs, BITP improves performance significantly and at the same time prevents information leakage.

RRPV Increments: Figure 13 (b) shows the fraction of evictions that result in RRPV increments. Note that this scenario is not critical in LRU based policies as SHARP evicts blocks from LRU position to LRU-3 position. However, with all the RRPB based chains, all the blocks within a cache set get affected if they share the same RRPV values. This increases LLC misses and with a detailed DRAM model (in contrast to a fixed latency of SHARP), degrades performance significantly.

Also, to prevent cross-core evictions resulting in inclusion
Based on 4-/8-/16-core simulations, we find that ELATED Wordtsc. Based on Table VII and +1.1%. AND -1.7% -3.12% -0.02%
BITP: A SHARP PERFORMANCE COMPARISON
ONCLUSION
+4.2% -5.1% +4.5% +1.2%
RIC

Figure 13 shows performance improvement in the same magnitude with BITP for per core L2:L3 ratios of 1 and 1:2. Based on the simulation results, we corroborate the findings of RIC [15] for small LLCs shared by a large number of cores and we conclude the same for BITP too. Based on the experiments, we find both RIC and BITP are effective and the magnitude of the effectiveness is the same. Note that both RIC and SHARP use a fixed latency DRAM model that also contributes to improving the margin of performance improvement. In case of an LRU replacement policy at the LLC, the effectiveness of SHARP, RIC, and BITP are similar. However, LRU policy is less effective for the multi-core system when compared with SHiP++ and HAWKEYE.

Summary: Based on 4-/8-/16-core simulations, we find that SHARP degrades system performance (in terms of fair-speedup) for 42% of mixes by more than 4.7%. Table VII summarizes the performance results for multi-core systems. A recent paper discusses some of the subtle issues that are not discussed in the original SHARP paper [50]. Based on Table VII and Table III, we can conclude that BITP eliminates information leakage with no hardware overhead, performance overhead, and additional support from the OS, compiler, run-time systems. Apart from performance reasons, as already mentioned, with SHARP, processor core generates an interrupt to notify the OS about the suspicious activity. BITP is simpler and free from additional hardware/software intervention.

V. RELATED WORK

This section discusses side-channel mitigation techniques apart from SHARP [16] and RIC [15]. Several cache partitioning techniques [10], [13], [51], [52] have been proposed to mitigate side-channel attacks. However, all these attacks affect system performance and fairness significantly. CATaLyst [10] partitions the LLC into insecure and secure partitions. Also, within the secure partition, it prevents replacement of cache blocks that store the secure data. CATaLyst demands changes to the programming language and run-time. Random fill cache [12] is a technique that is proposed to mitigate reuse-based side-channel attack at the L1. Another technique that thwarts side-channel attack is by random L1 cache mapping instead of standard cache mapping technique.

Timewarp [11] and fuzzy timing [53] are some of the run-time system techniques that try to fudge the timing information (mostly rdtsc) by adding noise. These techniques demand changes at the ISA level, applications level, and for some changes, it needs virtualization support, which is a substantial modification for an architect. These fuzzing techniques do not mitigate attacks that use new techniques (e.g., performance counters) to keep track of micro-architecture events. For applications that need to use rdtsc, these techniques demand changes at the system administrator level. Compared to all these techniques, BITP is simpler and yet efficient in mitigating LLC side-channel attacks.

There are few other mitigation techniques, such as CC-Hunter [54] and replayconfusion [55] that detect covert channels and not side-channels. HexPads [56] is a technique that uses heuristics (based on the LLC access counts of the attacker) in mitigating side-channel attacks. However, this technique is affected by false-positives. TLA [17] can be argued to be secure. However, an attacker can control its access patterns (hence temporal locality) to nullify the effects of TLA, making it a baseline inclusive cache. Also, TLA does not assure that it will not create back-invalidation-hits, which motivates the SHARP [16] proposal.

VI. CONCLUSION

This paper proposed back-invalidation-hits triggered prefetching (BITP) that mitigates various cross-core eviction based side-channel attack strategies at the last-level cache (LLC) by prefetching the addresses of interest. We showed the effectiveness of BITP by simulating attacks on AES-128, GnuPG and Poppler and quantified the probability of attack success and other relevant metrics for security effectiveness. We also showed the effect of BITP on system performance and fairness with the use of fair-speedup metric. We conclude that BITP does not compromise on system performance and system fairness for security and makes a case for secure but inclusive LLC. Overall, BITP is a prefetching framework that is simple (no additional hardware structures, no intervention from software writer and OS) yet effective to mitigate LLC cross-core side-channel attacks.

VII. ACKNOWLEDGEMENT

We would like to thank all the anonymous reviewers for their helpful comments and suggestions. We would also like to thank members of CARS research group, Mainak Chaudhuri, Debadatta Mishra, Clementine Maurice, Zecheng He, and Andre Seznec for their feedback on the initial draft. This work is supported by the SRC grant SRC-2853.001.


[48] “Calculating memory system power for ddr3.”


