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

Biswabandan 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 eviction 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].

The essence of an inclusive shared resource (e.g., an inclusive LLC that is always a superset of the private caches (L1 and L2)), becomes a security loophole when exploited carefully as an eviction of a cache block from the LLC, back-invalidates cache blocks in private caches. Moreover, future accesses by the victim incur cache misses at private caches. Note that inclusive LLCs are famous for simplifying the cache coherence layer as LLC becomes the directory and there is no need for additional hardware for maintaining cache coherence.

All the LLC eviction based cross-core side-channel attack strategies exploit this feature to attack the victim. Even for a non-inclusive LLC, a recent [8] successful eviction based cross-core attack, exploits the inclusive cache coherence directory to create inclusion-victims (back-invalidation hits). For the ease of understanding, unless specified, we focus on inclusive LLCs. We discuss a recent eviction attack on non-inclusive LLCs that uses inclusive coherence directory in Section III-E.

To understand how an inclusive shared resource helps in cross-core eviction attack, we look deep into these attacks: in Evict+Reload attack [1], the attacker evicts a specific cache block(s) of the victim at the shared inclusive LLC, and this eviction causes back-invalidation-hit(s) at the private caches of the victim. After a fixed interval, the attacker reloads the evicted address and if it gets a shorter access time (LLC hit), then the attacker can infer that the victim has accessed the cache block between the eviction and reload. Other strategies follow similar events to extract information about the victim's LLC accesses. All the eviction attacks use the notion of time precisely to schedule the events such as evict, flush, and reload.

The problem: In LLC side-channel attacks, the fundamental principle behind all the strategies is to observe LLC access time differences to infer about the data used by another core. What if an inclusive shared resource ensures equal access time for security-critical data even after a eviction? This problem is non-trivial.

Our goal is to propose a simple micro architecture technique that can completely mitigate cross-core LLC eviction based side-channel attacks with no performance and hardware overhead, and which does not demand intervention from software writer, instruction set architecture (ISA), compiler, runtime system, and operating system (OS). Prior proposals [9]–[14] that try to mitigate side-channel attacks at the LLC degrade

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00019

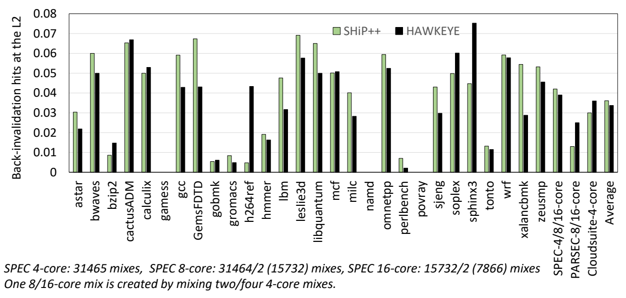

Fig. 1. Fraction of LLC evictions that result in back-invalidation-hits at the L2 for single-core and multi-core systems with 2MB/LLC per core.

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 a eviction 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 championship (CRC-2) held in ISCA '17. Figure 1 shows that on average, less than 4% LLC evictions cause back-invalidation-hits at the L2 for single-core and multi-core mixes involving SPEC CPU 2006 [22], PARSEC [23], and CloudSuite [24] benchmarks and out of which more than 98% of the back-invalidated blocks get reused. We use all the 4-core multi-programmed mixes possible (28 choose 4 with repetitions) for client based workloads and then combine them to create 8-core and 16-core mixes. For server workloads, we use the multi-threaded cloudSuite benchmarks and for scientific parallel applications, we use the multi-threaded PARSEC benchmarks. We observe a similar trend with other replacement policies that are mentioned in CRC-2 site. We do not report the back-invalidation-hits at the L1 as L2 is inclusive of L1. For this experiment, we use the gem5 [25] simulator with L1 D-cache of 32KB, L2 cache of 256 KB, and an inclusive LLC of 2MB/core, which is the industry standard in some of the recent commercial processors. Also, the recently concluded CRC-2 [26] has 2MB LLC/core.

Note that, the fraction of back-invalidations that hit at the L2 in the temporal locality aware (TLA) policy [17] is high because, 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 $\frac{256KB}{2MB}$ 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).

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][1], [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. Yarom 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/WRITEACK) 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.

[1]Note that this epoch is used in real machines.

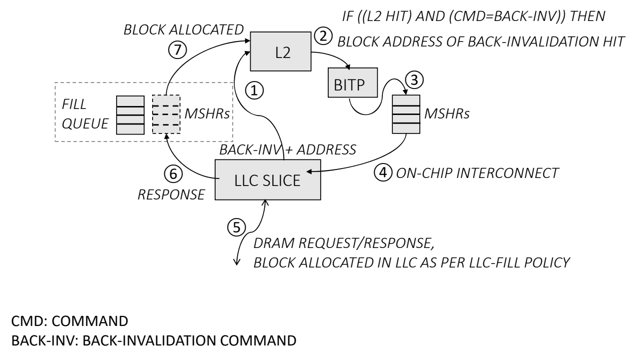
IF ((L2 HIT) AND (CMD=BACK-INV)) THEN
BLOCK ADDRESS OF BACK-INVALIDATION HIT
BLOCK ALLOCATED
L2
BITP
FILL QUEUE
MSHRs
MSHRs
BACK-INV + ADDRESS
ON-CHIP INTERCONNECT
LLC SLICE
RESPONSE
DRAM REQUEST/RESPONSE,
BLOCK ALLOCATED IN LLC AS PER LLC-FILL POLICY
CMD: COMMAND
BACK-INV: BACK-INVALIDATION COMMAND

Fig. 2. BITP Mechanism. MSHRS in ⑥ and ⑦ are the same as ③ and ④.

B. Metrics for Security Effectiveness

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.

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):
p1: 1.00, conventional mapping in which a DRAM address mapped to a particular cache set with probability 1.00 and it is known to the attacker. If it is not known to the attacker, then it will be less than 1.00.
p2: 1.00, for a successful attack, the attacker should be able to replace the cache block(s) of interest before the victim reloads. For a w-way cache, the attacker should access a particular set at-least w times for LRU based policy and $\bar{w}$ ($\bar{w}$ can be less than equal to w or greater than w) times for RRIP [34] based eviction policies.
p3: 1.00, this probability will change if we prevent replacement of the block of interest.
p4: 0.125, theoretically, the expected probability of getting a back-invalidation-hit with state-of-the-art replacement policies is $\frac{256KB}{2MB}$ = 0.125.

TABLE I
CONDITIONAL PROBABILITIES OF EVENTS OF INTEREST BASED ON PIFG.

	Events
p1	Memory block getting mapped into a cache set
p2	Cache block selected for replacement given the cache set
p3	Cache block selected by the replacement policy is evicted
p4	Evicted cache block leads to back-invalidation-hits at the L2
p5	Evicted block (that has caused back-invalidation-hit) when accessed again gets an LLC miss and the very next access gets a hit.
p6	LLC hit/miss getting mapped to the shorter/longer access time.

p5: 1.00, the attacker observes an LLC miss/hit in miss/hit type attacks.
p6: 1.00, a direct correlation between miss/hit with the LLC access time. So the PAS of the baseline system is **0.125** **(**$p1 \times p2 \times p3 \times p4 \times p5 \times p6$**)**. Next, we show the PAS for cross-core miss-type attacks, which is easy to understand followed by the hit-type attacks.

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 p5, 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 p1 to p4 (same as Evict+Time). p5 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 ($\frac{attacker=hit}{victim=accessed}$) 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 ($\frac{victim=accessed}{attacker=hit}$) till the point that a cache set is mapped to the memory block. A simple alternative is p5 can be exactly correlated with the TPR, which is 1.00 in the baseline. With BITP, p5 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 p5 are 0, which happens all the time with BITP.
(ii) However, 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. *So, in the worst case, the PAS for Evict+Reload is 0.125×PV. Based on our simulations on AES-128, GnuPG, and Poppler, we find PV varies from 0.04 to 0.26.*

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 = square (r)
6:     r = modulo (r, m)
7:     if ($e_i$==1) then
8:         r = multiply (r, b)
9:         r = modulo (r, m)
10:    end if
11: end for
12: return r
```

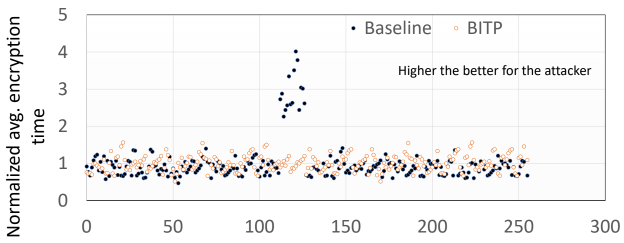

Fig. 3. Normalized average encryption time observed by the attacker for different plain-text values on AES-128. Higher avg. encryption time leads to a successful Evict+Time attack.

there is one sequence of p1 to p6 for the attacker (evicting victim's blocks). There is another sequence of p1 to p6 for the victim (evicting attacker's blocks), which leads to a PAS of **0.125×0.125 = 0.0156** in the baseline system. With BITP, the PAS becomes zero, because the victim does not evict the blocks of the attacker. So in the Probe stage, the attacker gets hits at its L2 (no LLC misses during the victim's accesses, no evictions, and no back-invalidations to the blocks of the attacker). So BITP prevents information leakage and also makes it difficult to mount Prime+Probe attack at the LLC.

Note that, BITP prefetches on all back-invalidation-hits, irrespective of the source of the request:attacker/victim and it is not dependent on the core-id of the request. Although PAS is a generic metric, PAS does not account for timing characteristics and does not adequately capture the nuances of side-channel attacks. In the next section, we show the relative LLC latency differences as observed by the attacker.

D. Access Time Difference with BITP

In this section, we evaluate the security effectiveness for cross-core LLC eviction based Evict+Time, Evict+Reload, and Prime+Probe attacks by running AES-128, GnuPG, and Poppler. In case of Evict+Time and Evict+Reload attacks, to find out the cache set that contains the critical cache block addresses, the spy *mmaps* the virtual addresses of interest. In case of Prime+Probe attack, it creates an eviction-set(s) before the prime process. In miss/hit type attacks, the attacker will be successful if it gets longer/shorter LLC access (execution) time.

1) Evict+Time Attack on AES: In Evict+Time attack, the attacker attacks AES-128 where it evicts an AES cache block containing table entries and then call a routine to encrypt with

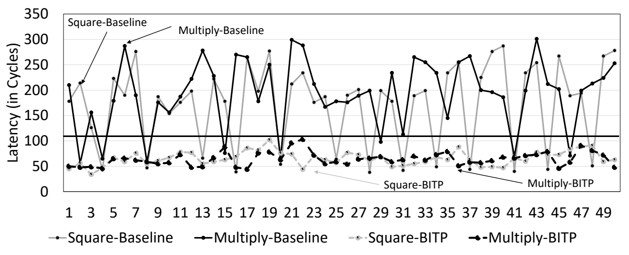

Fig. 4. LLC access time observed by the attacker while attacking GnuPG. Y-axis shows LLC access time. X-axis: attack epochs.

random plain-text and measures the encryption time. Note that, as Evict+Time is a miss-type attack, the attacker will be successful if it gets LLC misses for certain plain-text values, increasing the average encryption time. However, with BITP, the average encryption time does not change much (thanks to LLC hits) and the attack is unsuccessful. Figure 3 shows the average encryption time for each plaintext values normalized to encryption time averaged across all the plaintext values. We can see, plaintext values from 112 to 126 show LLC misses (higher avg. encryption time), which helps the attacker in the baseline.

2) Evict+Reload attack on GnuPG: GnuPG [18] is an open-source implementation of OpenPGP standard. We use the GnuPG 1.4.13 version that uses modular exponentiation in the form of square-and-multiply [38] Algorithm (refer Algorithm 1), where each occurrence of square-modulo-multiply-modulo corresponds to the exponent bit one whereas each occurrence of square-modulo not followed by a multiply operation corresponds to a zero. A complete trace of the square-and-multiply can help in recovering the exponent as mentioned in [5], [39], [40]. Note that modular exponentiation is the main computation in many other public-key ciphers like RSA and ElGamal. We run a simulated 2-core system with an attacker on core-0 and GnuPG on core-1. The attacker creates 10,000 epochs where each epoch is of 6,000 cycles (this gap is required for a successful attack. Note that this number may change with the cache and DRAM organization and it should be determined empirically). In each epoch, the attacker evicts and reloads the addresses corresponding to the square and multiply functions (*addresses of interest*).

The attacker measures 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 # 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*

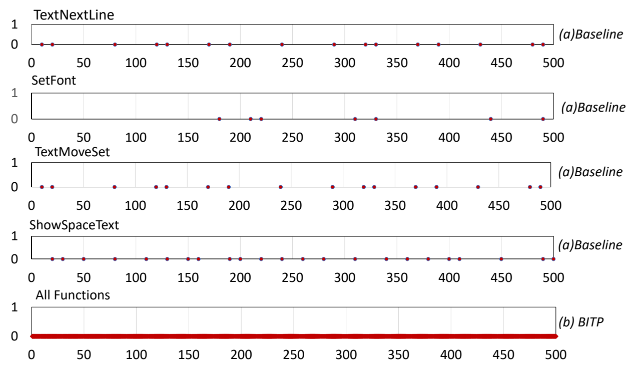

Fig. 5. Shorter/longer access time observed by the attacker in terms of `0s`/`1s` while attacking critical functions of Poppler. We do not show the "1s" explicitly. X-axis: attack epochs.

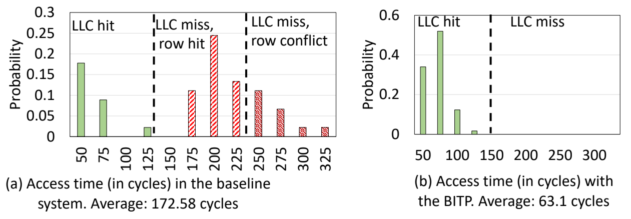

Fig. 6. Probability distribution function of LLC access time of critical accesses across all the attacks with the (a) baseline and (b) BITP.

block and an LLC hit to a prefetched cache block.

3) Prime+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 `pdftops` on core-1. The approach used in [41] that attacks four different functions of pdf2ps 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 10,000 epochs each of 8,000 cycles, where the attacker primes and probes the addresses corresponding to four functions that are accessed by the victim core. Typically, an LLC hit takes in between 40 to 126 cycles depending on which LLC slice (bank) has the requested address and an LLC miss takes around 150 to 325 cycles. So if an attacker sets a threshold of 130 cycles for an LLC hit and anything above 130 cycles as an LLC miss, then BITP makes sure that the attacker always gets LLC hit, access latency of fewer than 90 cycles. Figure 5 shows the shorter access times (less than 130 cycles) with "0" and more than 130 cycles as longer access time with "1" as observed with the baseline for all the four functions. It also shows the effectiveness of BITP that results in LLC shorter access times for all the functions all the time. Note that the shorter access time includes L2 hits too.

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.

TABLE II
SUMMARY OF SECURITY RESULTS ACROSS ALL ATTACKS.

	Baseline	BITP
PAS	0.125	0
Avg. access time	172 cycles	63 cycles
Range of TPR	0.77 to 0.91	0.04 to 0.11
Range of CSV	0.81 to 0.93	0.07 to 0.13

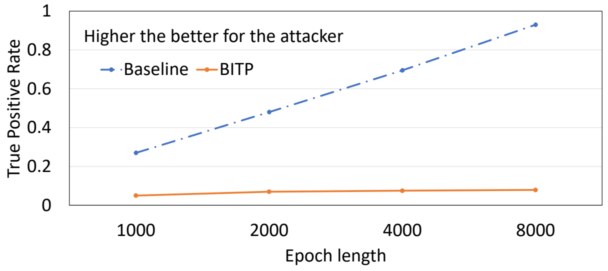

Fig. 7. True positive rate (TPR) versus epoch length.

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 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.

Invalidate+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-invalidation 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-invalidation hit and prefetch response: This time gap should be less than the gap between back-invalidation-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 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-invalidates 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

TABLE III
SHARP [16], RIC [15], AND BITP: A COMPARISON.

	BITP	RIC	SHARP
Needs OS support?	**No**	Yes, to identify read-only pages and thread private data, to flush the stale data in L2s, and to prevent thread migration	Yes (interrupt handling)
Affects LLC eviction priority chain?	**No**	No	Yes
Needs coherence directory support?	**No**	No	Yes (for probing)
Does thread migration affect?	**No**	No (if OS flushes the stale private data) and Yes otherwise	Yes (SHARP uses the core-id information to prevent cross-core back-invalidations but allows intra-core back-invalidations)
Hardware Overhead	**Zero**	RI bit/tag, 64KB for 32MB LLC	alarm-counter (12-bit) per core

TABLE IV
PARAMETERS OF THE SIMULATED SYSTEM.

Processor	1/2/4/8/16-cores, 3.7 GHz, out of order
L1 D/I, L2	32 KB (4 way), 256KB (8 way, inclusive)
Shared L3	2MB$\times$ cores, #slices=#cores, 16 way, inclusive
MSHRs	16, 16, 16/128/256 MSHRs at L1, L2, L3 with 1/8/16 cores
Cache line size	64B in L1, L2 and L3
Replacement policy	SHiP++ [20] and HAWKEYE [21]
L2 prefetchers	Best Offset [43]
On-chip interconnect	Ring
DRAM controller	1/2/4 controllers for 1/8/16-cores, Open Row, 64 read/write queues, FR-FCFS, drain-when-full
DRAM bus	split-transaction, 800 MHz, BL=8
DRAM	DDR3 1600 MHz (11-11-11) Max bandwidth/channel - 12.8 GB/sec

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 over SHARP and RIC. BITP does not demand OS intervention and it does not incur additional hardware. Apart from these important points, BITP does not affect the LLC replacement policy chain. SHARP affects the replacement priority, which causes performance degradation with SHiP++ and HAWKEYE based policies (details in Section IV-C).

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].

TABLE V
CLASSIFICATION OF BENCHMARKS.

Benchmarks	Type
h264ref, perlbench, povray, sjeng, gamess, namd	L2 fitting
astar, bzip2, calculix, hmmer, xalancbmk, namd classification, cloud9, bodytrack, dedup, x264, ferret, freqmine, swaptions, blackscholes, raytrace, fluid., vips	LLC fitting
mcf, libquantum, sphinx3, omnetpp, gobmk, GemsFDTD, bwaves, gcc, lbm, leslie3d, milc, zeusmp, catus., tonto, wrf, soplex, cassandra, nutch, streaming, streamcl., facesim, canneal	LLC thrashing

TABLE VI
REPRESENTATIVE WORKLOAD MIX TYPES.

Mix type	
1	All L2 fitting (L2F)
2	All LLC fitting (LLCF)
3	All LLC thrashing (LLCT)
4	50% L2F + 50% LLCF (0.5L2F-0.5LLCF)
5	50% L2F + 50% LLCT (0.5L2F-0.5LLCT)
6	50% LLCF + 50% LLCT (0.5LLCF-0.5LLCT)
7	25% LLCT + 75% Ł2F (0.25LLCT-0.75L2F)
8	Random mix

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., $\frac{Exectime_{baseline}}{Exectime_{technique}}$. For multi-programmed mixes, we use harmonic mean of speedups (fair-speedup (FS)) [46]. FS = $\frac{N}{\sum_{i=0}^{N-1} \frac{IPC_i^{alone}}{IPC_i^{together}}}$, where IPC$_i^{together}$ is the IPC of core i when it runs along with other N-1 applications nd IPC$_i^{alone}$ 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*

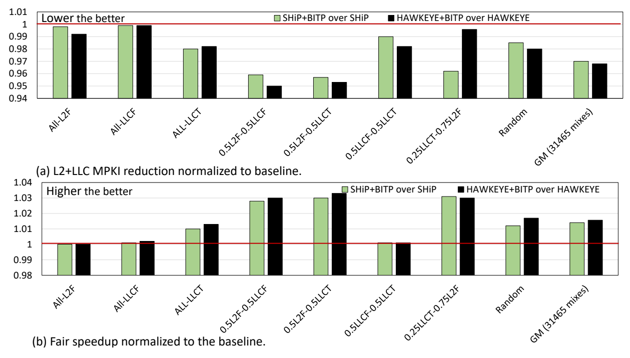

Fig. 8. Normalized L2+LLC MPKI and performance for 4-core multi-programmed mixes. SHiP:SHiP++ and GM: geometric mean.

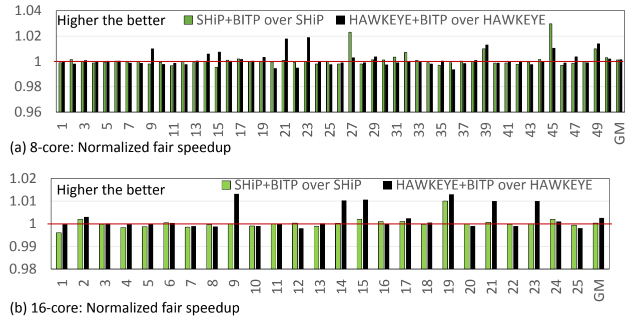

Fig. 9. Performance of BITP for 8/16-core (16/32MB L3, 2/4 DRAM controllers) systems.

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-thrashing 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

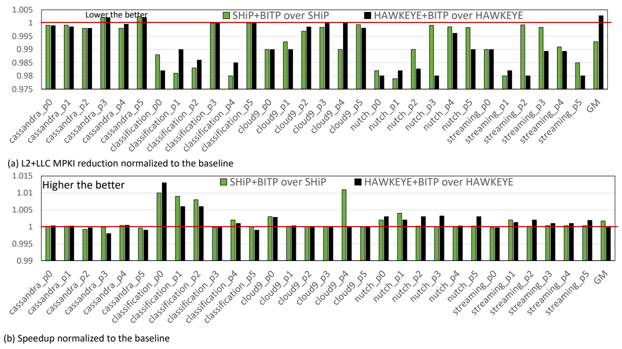

Fig. 10. Normalized L2+LLC MPKI and speedup for 4-core CloudSuite benchmarks. _px: phase x. SHiP:SHiP++

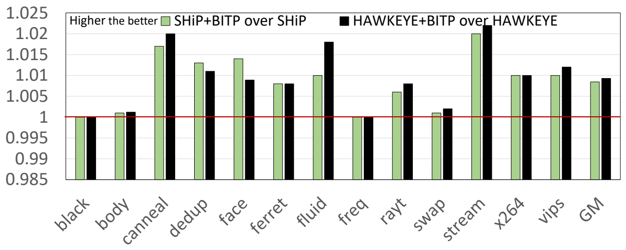

Fig. 11. Normalized speedup averaged across 8-threaded/16-threaded PARSEC applications.

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-invalidations 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.

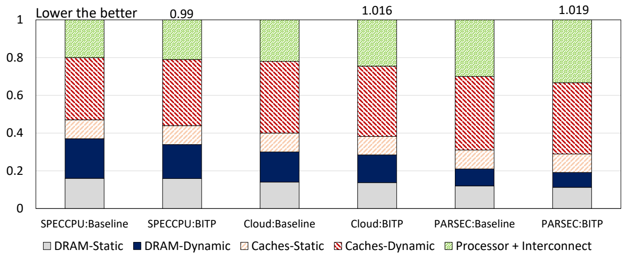

Fig. 12. Normalized system energy consumption of BITP compared to the baseline.

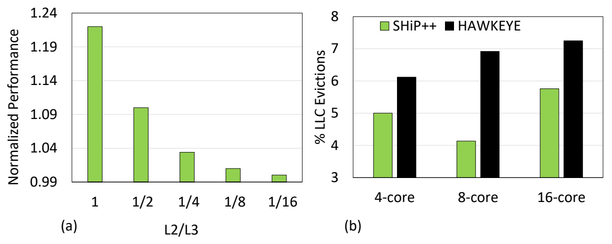

Fig. 13. (a) Normalized performance of BITP for different L2/L3s. (b) LLC evictions that cause RRPV increments with SHARP in %.

C. Comparison with the prior works

SHARP prevents cross-core evictions that cause back-invalidation-hits on LRU based replacement policies by sending four block addresses to L2, as mentioned in Section III-H. However, when we try to do the same with SHiP++ and HAWKEYE, some mixes are significantly affected. For example, one of the 4-core mixes show a performance degradation of 4.8% with SHARP that uses SHiP++ replacement policy.

The primary reason for this behavior is that while preventing cross-core evictions, SHARP increases the RRPVs of multiple blocks (in contrast to LRU) and because of which these blocks get evicted quickly.

For example, on a two-core system, if a core is trying to evict a block with RRPV=3 and the block is present in another core's L2 cache, SHARP does not evict the block and explores other possible cache blocks that will not create inclusion victims. There are cases, where the rest of the blocks belong to RRPV values of less than 3 and the replacement policy increments their RPPV values. In the worst case, rest of the blocks may have RRPV values of 0 (cache friendly blocks) and to prevent cross-core evictions, SHARP increments their RRPVs to 3 (causing early evictions). This situation becomes equally worse for SHARP with HAWKEYE as HAWKEYE uses 3-bit RRPV values (0 to 7).

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 RRIP 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 victims at other cores, a core evicts its useful blocks at the LLC and L2. Applications with the large working set (LLC thrashing type) suffer from this behavior. On average SHARP degrades performance by 1.7% (maximum degradation of 5.1%).

Compared to SHARP, RIC does not suffer from this behavior. There are few mixes in which BITP performs slightly better (in the order of 0.5%) than RIC in which an L2 block reference is not present in L2 and LLC (with RIC) but present in L2 (with BITP). Note that, in RIC [15], the authors report significant performance improvement with RIC for an 8-core system with 2MB LLC (256KB per core with per core L2:L3 ratio of 1.0) and 4MB LLC (L2:L3 ratio of 1:2). 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.

TABLE VII
SHARP [16], RIC [15], AND BITP: A PERFORMANCE COMPARISON.

	BITP	RIC	SHARP
Avg. perf.	**+1.2%**	+1.1%	-1.7%
Max. perf.	**+4.5%**	+4.2%	+1.8%
Min. perf.	**-0.02%**	-0.08%	-5.1%
DRAM traffic (demand+prefetch)	+1.05% (improves)	+1.73%	-3.12% (degrades)

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.

References

[1] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, 2015, pp. 897–912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss

[2] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 605–622. [Online]. Available: http://dx.doi.org/10.1109/SP.2015.43

[3] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 591–604. [Online]. Available: http://dx.doi.org/10.1109/SP.2015.42

[4] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653687

[5] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 719–732. [Online]. Available: http://dl.acm.org/citation.cfm?id=2671225.2671271

[6] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 990–1003. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660356

[7] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 549–564. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp

[8] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *2019 2019 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 56–72. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2019.00004

[9] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 871–882. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978324

[10] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 406–418.

[11] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 118–129. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337159.2337173

[12] F. Liu and R. B. Lee, "Random fill cache architecture," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 203–215. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.28

[13] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "Secdcp: Secure dynamic cache partitioning for efficient timing channel protection," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 74:1–74:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2898086

[14] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 494–505. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250723

[15] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: ACM, 2017, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/3061639.3062313

[16] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (sharp): Defending against cache-based side channel atacks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 347–360. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080222

[17] A. Jaleel, E. Borch, M. Bhandaru, S. C. S. Jr., and J. S. Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies," in *43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2010, 4-8 December 2010, Atlanta, Georgia, USA*, 2010, pp. 151–162. [Online]. Available: https://doi.org/10.1109/MICRO.2010.52

[18] "Gnupg, https://www.gnupg.org/software/index.html."

[19] "Poppler, https://poppler.freedesktop.org/."

[20] C. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. S. Jr., and J. S. Emer, "Ship: signature-based hit predictor for high performance caching," in *44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, Porto Alegre, Brazil, December 3-7, 2011*, 2011, pp. 430–441. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155671

[21] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 78–89. [Online]. Available: https://doi.org/10.1109/ISCA.2016.17

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

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[24] "Cloudsuite, http://parsa.epfl.ch/cloudsuite/cloudsuite.html."

[25] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. S. B. Altaf, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[26] "CRC2 objective: http://crc2.ece.tamu.edu/#objective,."

[27] "CRC1, https://www.jilp.org/jwac-1/."

[28] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, ser. DIMVA 2016. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 279–299. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-40667-1_14

[29] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross processor cache attacks," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: ACM, 2016, pp. 353–364. [Online]. Available: http://doi.acm.org/10.1145/2897845.2897867

[30] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel aslr," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 368–379. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978356

[31] "Cacheflush, http://man7.org/linux/man-pages/man2/cacheflush.2.html."

[32] "NACL, https://developer.chrome.com/native-client."

[33] T. Allan, B. B. Brumley, K. Falkner, J. van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 422–435. [Online]. Available: http://doi.acm.org/10.1145/2991079.2991084

[34] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. S. Emer, “High performance cache replacement using re-reference interval prediction (RRIP),” in *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, 2010, pp. 60–71. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815971

[35] R. B. Lee and W. Shi, “HASP 2013, the second workshop on hardware and architectural support for security and privacy, tel-aviv, israel, june 23-24, 2013.” ACM, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487726

[36] Z. He and R. B. Lee, “How secure is your cache against side-channel attacks?” in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 ’17. New York, NY, USA: ACM, 2017, pp. 341–353. [Online]. Available: http://doi.acm.org/10.1145/3123939.3124546

[37] C. Maurice, N. Scouarnec, C. Neumann, O. Heen, and A. Francillon, “Reverse engineering intel last-level cache complex addressing using performance counters,” in *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404*, ser. RAID 2015. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 48–65. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26362-5_3

[38] D. M. Gordon, “A survey of fast exponentiation methods,” *J. Algorithms*, vol. 27, no. 1, pp. 129–146, Apr. 1998. [Online]. Available: http://dx.doi.org/10.1006/jagm.1997.0913

[39] O. Aciiçmez and W. Schindler, “A vulnerability in rsa implementations due to instruction cache analysis and its demonstration on openssl,” in *Proceedings of the 2008 The Cryptopgraphers’ Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 256–273. [Online]. Available: http://dl.acm.org/citation.cfm?id=1791688.1791711

[40] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-vm side channels and their use to extract private keys,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS ’12. New York, NY, USA: ACM, 2012, pp. 305–316. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382230

[41] T. Hornby, “Side-channel attacks on everyday applications: Distinguishing inputs with flush+reload,” 2016.

[42] G. Irazoqui, T. Eisenbarth, and B. Sunar, “S$a: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes,” in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 591–604.

[43] P. Michaud, “Best-offset hardware prefetching,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 469–480.

[44] “Champsim, https://github.com/ChampSim/ChampSim.”

[45] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “Dramsim2: A cycle accurate memory system simulator,” *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan 2011.

[46] K. Luo, J. Gummaraju, and M. Franklin, “Balancing thoughput and fairness in smt processors,” in *2001 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS.*, 2001, pp. 164–171.

[47] “CACTI, http://www.hpl.hp.com/research/cacti.”

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

[49] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, “Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE ’09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 423–428. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874721

[50] D. Kumar, C. S. Yashavant, B. Panda, and V. Gupta, “How sharp is SHARP ?” in *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: https://www.usenix.org/conference/woot19/presentation/kumar

[51] Z. Zhou, M. K. Reiter, and Y. Zhang, “A software approach to defeating side channels in last-level caches,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: ACM, 2016, pp. 871–882. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978324

[52] T. Kim, M. Peinado, and G. Mainar-Ruiz, “Stealthmem: System-level protection against cache-based side channel attacks in the cloud,” in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362804

[53] W.-M. Hu, “Reducing timing channels with fuzzy time,” *J. Comput. Secur.*, vol. 1, no. 3-4, pp. 233–254, May 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=2699806.2699810

[54] J. Chen and G. Venkataramani, “Cc-hunter: Uncovering covert timing channels on shared processor hardware,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 216–228. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.42

[55] M. Yan, Y. Shalabi, and J. Torrellas, “Replayconfusion: Detecting cache-based covert channel attacks using record and replay,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–14.

[56] M. Payer, “Hexpads: A platform to detect ”stealth” attacks,” in *Proceedings of the 8th International Symposium on Engineering Secure Software and Systems - Volume 9639*, ser. ESSoS 2016. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 138–154. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-30806-7_9