Avenger: Punishing the Cross-Core Last-Level Cache Attacker and Not the Victim by Isolating the Attacker

Yashika Verma
Dept. of Computer Science and Engineering
Indian Institute of Technology Kanpur
yashikav@cse.iitk.ac.in

Biswabandan Panda
Dept. of Computer Science and Engineering
Indian Institute of Technology Bombay
biswa@cse.iitb.ac.in

***Abstract*—On a multi-core system, the shared last-level cache(LLC) is vulnerable to various kinds of cross-core contention-based attacks. LLC randomization and LLC partitioning are two promising mitigation strategies that mitigate these attacks. LLC-randomization techniques make an attacker's life difficult in mounting contention-based attacks but do not entirely mitigate them. Randomized caches are also ineffective in preventing occupancy-based attacks. In contrast, state-of-the-art LLC partitioning techniques mitigate all possible LLC contention-based attacks by allocating isolated LLC regions to different processes or security domains. However, restricting processes to isolated LLC region(s) affects overall LLC utilization and incurs performance overhead (as high as 72%) and memory subsystem energy overhead (as high as 89%); effectively providing security guarantee at the cost of performance and energy.**

One of the primary reasons for this trend is the fundamental design choice that drives all the state-of-the-art secure LLC partitioning techniques, which isolate all the applications and all the time at the LLC. We revisit this choice and argue that we need to isolate only the attacker process and not all the processes. To isolate the attacker, we propose Avenger, a mitigation technique that uses a state-of-the-art LLC contention attack detector and isolates only the attacker. The detector is flexible and can be trained as per the security requirements of any organization or cloud provider. Experimental results on a 16-core simulated system with one attacker and 15 victims show that Avenger outperforms three state-of-the-art secure LLC partitioning techniques in performance and energy overhead without affecting security. Overall, Avenger provides a robust security guarantee against all contention-based cross-core LLC attacks with 1% average performance overhead in contrast to an average performance overhead of more than 17% with the state-of-the-art secure LLC partitioning techniques.

I. Introduction

Cross-core contention-based attacks at the shared last-level cache (LLC) are successful because an attacker (spy application) can cause controlled contention at the LLC sets and later can observe the effect of contention by measuring the latency differences between an LLC hit and an LLC miss. As multiple cores share the LLC, a successful attacker can cause information leakage by disclosing the victim's sensitive data. Past works show that these contention-based LLC side-channel attacks have been successful in recovering cryptographic keys [31], user keystrokes [33], and browsing history [36]. These attacks are practical even on a cloud setting [33].

LLC attacks can cause information leakage either through (i) eviction-based attacks like Prime+Probe [31], (ii) shared memory based flush attacks like Flush+Reload [51] that uses a `clflush` like instruction to flush out victim's cache lines, and (iii) cache occupancy-based attack that can perform remote website fingerprinting across browser tabs using just HTML+CSS, and observing the LLC space (working set) occupied by a victim [36].

There are two broad categories of cross-core LLC contention-based attack mitigation techniques: LLC randomization and LLC partitioning. Randomized LLCs [39], [47]–[49] randomize the address to cache set mapping, iteratively with the help of a parameter called *remap interval*, which makes the life of an eviction-based attacker difficult. Although randomized LLCs are simple to implement, these techniques are not effective across all kinds of attacks. For example, randomized LLCs do not mitigate cache occupancy-based attacks [36] and performance (denial of service) attacks [29]. A recent work named DAMARU [29] shows that randomized caches can cause significant performance degradation because of encryption and remapping of LLC blocks. So, in summary, randomized caches do not mitigate all the possible LLC contention attacks.

Recent LLC partitioning techniques [15], [26], [34] for security, on the other hand, partition the LLC either at the cache way level or the cache set level. These techniques provide complete isolation from an attacker (in principle, provide no information leakage) and hence mitigate all kinds of cross-core LLC contention-based attacks. However, partitioning techniques incur performance, memory subsystem energy, and fairness overheads. A recent LLC partitioning technique [34] provides finer granularity of cache partitioning that is flexible and can provide LLC space as per the application's cache footprint requirement. However, it is still a rigid approach, and it isolates all the applications and incurs performance overhead.

Figure 1 shows performance degradation with state-of-the-art secure LLC partitioning techniques for a 16-core system that runs 18 representative multi-programmed workloads created from SPEC 2017 [12] and GAP [11] benchmark suites, normalized to a non-secure baseline with no partitioning. On average, there are performance slowdowns of 22%, 18%, and 17% with maximum slowdowns of 72%, 65%, and 67% with page coloring [15], DAWG [26], and BCE [34], respectively. Note that it takes a decade of research in microarchitecture optimizations at the LLC to offer 4% performance improvement [35] and average performance overhead of 17% is significant. In terms of dynamic energy, we see an average increase in the DRAM dynamic energy consumption of as much as 89% with page coloring (refer to Figure 9). Table II lists all simulated parameters (similar to Intel Sunny Cove) [3]. Table III shows the LLC misses per kilo instructions (MPKI) for the benchmarks that we use. Table IV provides details about representative mixes.

The problem. Randomized LLCs do not mitigate all possible LLC attacks, and the agility of the attacker limits the effectiveness of these techniques. Partitioned LLCs provide a robust security guarantee at the cost of performance and memory subsystem energy. One of the primary reasons for performance and energy overheads is the rigid approach used by state-of-the-art LLC partitioning techniques that isolate all the applications running on a system and not the attacker. The fundamental principle behind mitigating LLC contention-based attacks should be to prevent the *cyclic-interference* that causes cross-core LLC evictions between the attacker and the victim applications at the LLC, and not among all the benign applications. A cyclic interference between an attacker (A) and victim (V) for a given LLC location follows this access pattern: A $\prec$ V $\prec$ A; an interference in *both* the directions A $\prec$ V and V $\prec$ A, with information leakage from V to A. Note that cyclic interference can occur even among benign applications. However, the frequency of cyclic interference is extremely low (e.g., on average, two times in 100,000 cycles for SPEC CPU 2017 and GAP benchmarks in contrast to more than four times in 1500 cycles for an agile attacker [48]). **The pertinent question.** *Is it possible to* isolate *the attacker and not the victims while providing a security guarantee? This is a pertinent question because we do not need to isolate all the applications at the LLC. So, instead of a rigid approach, is it possible to propose a flexible approach that isolates the attacker to provide a security guarantee with negligible performance and energy overheads?*

Our goal and approach. We envision a secure LLC that does not incur significant performance and energy overheads. We provide a security guarantee with a philosophy of *punishing (isolating) the attacker and not the victim*, hence named Avenger. To achieve the same, we use an attack-detector based *LLC way* based isolation approach, and we isolate the attacker at the LLC only when the detector classifies an application as the attacker. Note that, at a given time, multiple attackers may be active on a system, and providing an LLC way-based isolation for each attacker will not be scalable (for example, 15 attackers on a 16-core system can occupy 15 out of 16 ways). Avenger, instead, makes sure that all the attackers are restricted to limited LLC ways. For example, on a 16-core system, if there are 15 attackers, then all the attackers (once they get detected) will be allocated to the same LLC way.

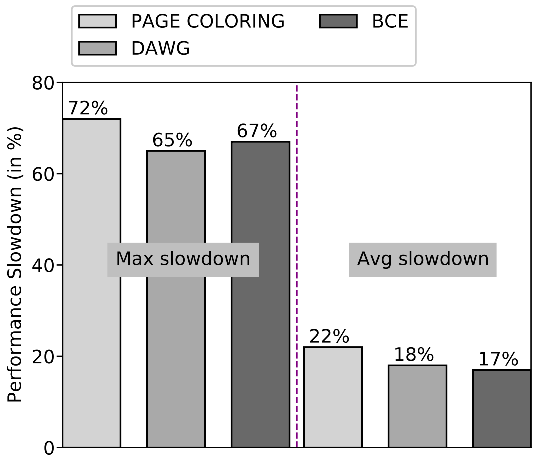

Fig. 1. Maximum and average performance degradation among 18 16-core representative mixes of SPEC CPU 2017 [12] and GAP [11] benchmarks with three state-of-the-art LLC partitioning techniques [15], [26], [34] normalized to a non-secure baseline with no LLC partitioning.

We use an agile state-of-the-art attack detector (Cyclone) [21] and isolate *only* the attacker(s) the moment a detector detects an attack. Once we isolate an attacker, the attacker cannot perform contention-based attacks, providing complete security. This approach makes sure that co-running benign applications are not isolated and their performance is unaffected. *Note that Cyclone detects an attack but not the attacker.* We extend Cyclone with the additional functionality of detecting an attacker. We identify an attacker based on the direction of interference. For example, the interference count from A $\prec$ V $\prec$ A is significantly more than the interference count from V $\prec$ A $\prec$ V.

Challenges. Though an attack-detector approach is a promising approach in providing isolation, there are many challenges that are as follows: (i) the attack detector should be accurate, (ii) the detector should be flexible so that future attacks can be trained and detected, and (iii) the detector should be agile and should have extremely low false-positive rate. Cyclone tackles these challenges as it is implemented as programmable hardware. It is flexible (e.g., a cloud provider can tune it and train it as per its security requirements), accurate (provides 100% accuracy), agile (can raise an alert in a few thousands of cycles), and provides a false positive rate of less than 0.0000001%.

Overall, we make the following key contributions:

- We evaluate the state-of-the-art secure but rigid LLC partitioning techniques keeping performance and memory sub-system energy in mind. We analyze the primary reasons behind significant performance and energy overhead in the case of workloads that contain memory-intensive applications with high LLC MPKIs (Section II-E).
- We make a case for an LLC-contention attack detector-

based mitigation approach that is non-rigid and flexible. We propose Avenger that isolates the attacker and not the benign applications (Section III).
- Empirical evaluations on a 16-core simulated system show that Avenger outperforms three state-of-the-art secure LLC partitioning techniques [15], [26], [34]. Avenger delivers performance and energy closer to the non-secure baseline without affecting the security guarantee (Section IV).

II. Background

A. Threat Model

We assume the following capabilities in our attacker:
(i) She is aware of LLC indexing (through reverse-engineering).
(ii) She can access the LLC by sending controlled memory accesses to her data, but she cannot access any cache line that is not part of her own address space. However, she can accurately differentiate between an LLC hit and an LLC miss by measuring memory access latency difference.
(iii) She is capable of mounting all the possible LLC contention attacks that exploit a timing channel, such as an eviction-based attack, flush-based attacks, and cache occupancy-based attacks. For eviction and occupancy-based attacks, she is not restricted by time to form an eviction set and then attacks the victim. She can use various eviction strategies to create an eviction set.
(iv) Her goal is to obtain the complete address or the cache index bits of the address for eviction-based attacks.
(v) The attacker and the victim are running on two different cores on a multi-core system, sharing LLC. Private caches that are shared by multiple processes of a hyper-threading processor core are spatially and temporally partitioned among processes and flushed on context switches, providing isolation. Without loss of generality, our threat model excludes attacks on other micro-architecture units like on-chip interconnect, cache coherence directories, and other port contention-based attacks as we focus primarily on LLC attacks.
(vi) Similar to other LLC partitioning techniques [15], [26], [34], we assume secure system software support (OS or security monitor [15]) is available.

B. LLC Contention Attacks

LLC contention attacks can be broadly grouped into four categories that are as follows:
Eviction-based cache attacks. In eviction-based attacks, an attacker fills its data into an LLC set that conflicts with the victim's data. Later in the Probe step, the attacker re-accesses its data, and if it observes longer access latency, then it means that the victim has evicted some of the attacker's lines (e.g., Prime+Probe [31]).
Shared memory-based attacks. In shared memory-based attack (like Flush+Reload [51]), an attacker shares its address space with the victim (e.g., shared libraries). The attacker flushes cache lines that are shared by both the attacker and the victim and observe the victim's access to the same cache lines by observing memory access latency.
Occupancy-based attacks. An LLC occupancy-based attacker observes the LLC space occupied by the victim application. Recent attacks on website fingerprinting [36] exploit the dynamic LLC usage between the attacker and the victim.
Flush-based eviction attack. A recent work [38] shows that an attacker can mount an eviction-based attack by flushing her private data while creating an eviction set. This method is faster than conventional eviction attacks like Prime+Probe. Note that this is different from shared memory-based flush attacks where the attacker flushes shared and read-only LLC lines.

C. Recent Advancements

CEASER [47]. CEASER provides randomization by encryption. It encrypts a physical address, based on a *key* to get the encrypted address on an LLC access. To mitigate eviction-based attacks, CEASER remaps cache lines with a different *key* after a fixed interval known as the remapping period. During a remap period, CEASER remaps with a new *key* that remaps cache lines into a new LLC set. As per a recent S&P 2021 paper [38], remapping based on LLC evictions instead of LLC accesses is recommended. Even in the encrypted address space, LLC contention-based attacks are still possible.
CEASER-S [48] and SCATTERCache [39]. CEASER-S and SCATTERCache go one step ahead of CEASER and propose randomization with a skewed associative LLC to mitigate an agile eviction-based LLC attacker that can attack CEASER with slow remapping rates.
MIRAGE [49]. MIRAGE is a fully-associative LLC that uses multi-index randomization with a global eviction policy. It provides a proxy for a fully associative LLC with the help of random replacement. To enable global random replacement, it decouples the tag array from the data array. MIRAGE incurs 20% storage overhead at the LLC. LLC partitioning techniques, on the other hand, provide isolation boundaries (through way or set partitioning) and can mitigate all the contention-based LLC attacks.
DAWG [26]. Dynamically Allocated Way Guard (DAWG) at the LLC uses a software configurable mask to decide *way* allocations among multiple security domains running on a multi-core system. Through this way partitioning, DAWG prevents cross-domain (cross-core) interference because of LLC hits, LLC misses, and replacement policy updates because of LLC hits and misses. DAWG allows sharing among multiple domains by duplicating a shared cache line across security domains. One of the limitations of DAWG is the upper limit on the isolated domains that are bounded by the number of LLC ways.
Page Coloring [15]. Page Coloring at the LLC creates isolated regions at the LLC set level. LLC partitioning via page-coloring creates different DRAM regions and uses DRAM region bits with LLC index bits to access LLC. The usage of DRAM region bits in the index bits guarantee that each DRAM region gets non-overlapped sets at the LLC. MI6 [15] is one

of the recent techniques that use page coloring at the LLC for isolation. MI6 statically partitions the DRAM and LLC space. One of the limitations of page coloring technique is that it cannot manage LLC space and DRAM space independently. For example, if a security domain is allocated 3/4th of an LLC, then page coloring also allocates 3/4th of DRAM space to the same security domain, and vice versa.

BCE [34]. A recent LLC partitioning technique Bespoke Cache Enclave (BCE), makes a case for flexible cache partitioning that provides isolation by creating partitions as small as 64KBs. One of the key benefits of BCE is that the number of partitions are not restricted by the number of LLC ways, and LLC space allocation is independent of DRAM space allocation, making it a scalable technique compared with DAWG and page coloring.

D. LLC Contention Attack Detectors

In recent years, LLC attack detectors [18], [21], [38], [46] have been proposed to detect malicious activity at the LLC. A recent hardware detector [38] detects an eviction-based attacker by observing LLC evictions and their distributions at LLC set level. Most of these detectors use machine learning (ML) to improve their precision. Out of all the hardware detectors, we find Cyclone [21] to be the most robust and agile.

Cyclone is a programmable hardware detector for contention-based cache information leakage. It detects inter-application (security domain) cyclic interference at shared resources like LLC and DRAM. For detecting information leakage due to shared LLC, each cache line is governed for cyclic interference. Each cache line is associated with current and previous application domain-ids. Note that the domain ID is provided by the OS or the security monitor, whichever is secure. Cyclone uses local detectors(LD) to track and count cyclic interference counts at the LLC and DRAM. If the cyclic events count exceeds the threshold within a time window, LDs send cyclic interference summary to global detectors(GD). Each LD has a set of counters shared among all the cache lines. The GD can be designed as a classifier trained based on the cyclic interference count of benign applications (e.g., SPEC CPU 2017 benchmarks). Based on our experiments, we find Cyclone provides an accuracy of 100% with a false-positive rate of 0.0000001%.

E. Motivating observations

LLC randomization techniques do not mitigate all possible LLC contention attacks. In contrast, LLC partitioning mitigate contention-based attacks completely by allocating isolated regions in LLC. However, the rigid creation of isolated regions for each security domain leads to LLC utilization restrictions that affect system performance and memory subsystem energy consumption.

Why do state-of-the-art secure partitioning techniques incur performance overhead? Figures 2 shows the performance degradation for the workload mix2. As we can see, there is significant reduction in performance (more than 70% in Figure 2) for memory-intensive applications like `mcf`. One of the primary reasons for performance degradation is the rigid design approach used by DAWG, page coloring, and BCE. In the case of DAWG, each application (process) gets 2MB LLC space, 1-way per set for a 16-way, 32MB LLC (16 2MB LLC slices). With page coloring, each application gets 1MB LLC (maintaining the one-to-one dependency between DRAM page and LLC space allocation). BCE allocates 2MB LLC space per application (2048 LLC sets per application), however, at an LLC set granularity. All the three techniques incur significant LLC misses (conflict and capacity) for mixes that contain applications with high LLC MPKIs. Figure 3 shows the average LLC MPKI for `mcf` and `perlbench` of mix2. Ideally, we need a secure LLC partitioning technique that can provide performance closer to the non-secure baseline without compromising the security guarantee, and our proposal Avenger delivers the same (Figure 2).

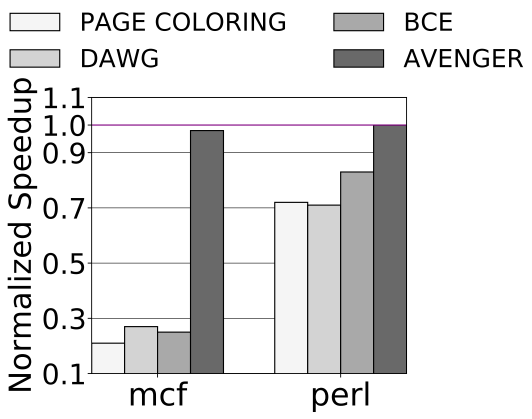

Fig. 2. Normalized IPC and average LLC MPKI for each benchmark in workload mix2 (seven copies of `mcf`, eight copies of `perlbench`, and one attacker) on a 16-core system.

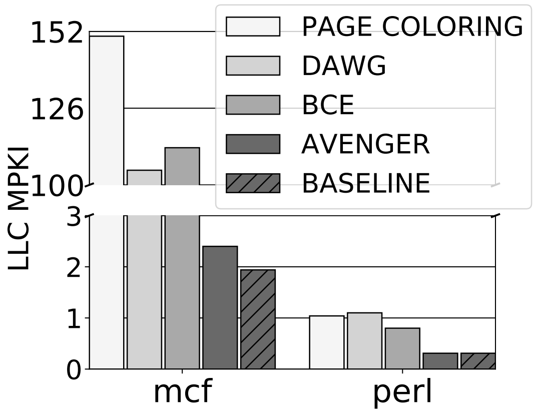

Fig. 3. Average LLC MPKI for workload mix2(seven copies of `mcf`, eight copies of `perlbench (perl)`, and one attacker) on a 16-cores system.

III. AVENGER: ISOLATING THE ATTACKER

System software (OS or security monitor) support. Avenger relies on an allocation of a security domain ID

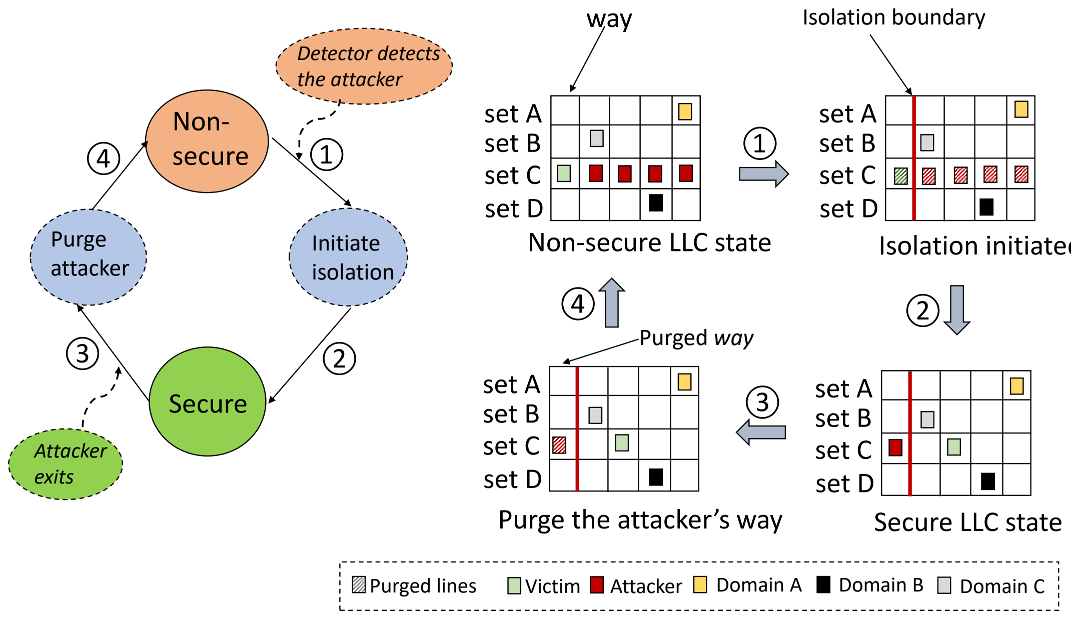

Fig. 4. An LLC with Avenger in action. Avenger starts with a non-secure state and isolates the attacker once the detector raise an alarm, and goes back to the non-secure state once the attacker exits.

that Cyclone uses. The domain ID should be assigned by trustworthy system software. It can be an OS or a security monitor [16] in case the OS is not trustworthy. Domain ID is assigned to each process that is running on a system. For a virtual machine (VM) that runs multiple processes, multiple domain IDs are assigned to ensure isolation within and across VMs. In the case of a system with trusted execution environments like Intel SGX and ARM Trustzone [4] [5], an initiation of trusted execution code leads to assignment of a new domain ID within a single process. An exit from a trusted execution environment or when a process exits, the domain ID is de-allocated by the system software. Page sharing between processes that do not trust each other has been restricted and allocated with different domain IDs; the OS or the security monitor ensures no write sharing. For read-only pages, LLC lines get duplicated for each domain ID, similar to [26], [34], [39], [49]. This eliminates the flush-based LLC contention attacks ensuring flush/hit of one domain on a shared LLC line does not affect another, and as an attacker can only flush its LLC line(s). Cache coherence directory based attacks are mitigated by the zero directory eviction victim [13].

Attack detector extension. We use a modified version of Cyclone that raises an alert after detecting an attack and provides the domain ID of the attacker based on cyclic interference count between two security domains, say domain A and domain B. If domain A is the attacker then the cyclic interference count from domain A to domain B is much higher than from domain B to domain A. Compared to the original Cyclone proposal, we use a relatively more agile version of Cyclone that raises the alarm four times in 1500 cycles. We also use interference counters based on the direction of interference, such as $A \prec V \prec A$, and $V \prec A \prec V$. We store two recent domain IDs per LLC tag. Note that for shared lines (read-only pages), only one domain ID will be active for the lifetime of an LLC line as shared lines are duplicated, similar to [34], [39], [49].

We use buckets for monitoring cross-core cyclic interference counters as part of the local detector of Cyclone. The bucketing technique maps each LLC line to an interference counter instead of keeping an interference counter for each cache set. When a cyclic interference event occurs, a hash is used to map the address to a bucket and increment the counter. We do not use any counters at the DRAM as the shared memory-based flush attacks are mitigated by design thanks to the system software support. As mentioned in Section I, the detector is implemented as a programmable hardware implemented using systolic arrays [42]. Note that the detector has to be trained for benign and non-benign applications as per the security requirement of an organization or a cloud provider.

Avenger in a nutshell. At the LLC, Avenger provides LLC isolation based on an LLC attack detector, thanks to 100% accuracy provided by Cyclone [21]. Avenger provides high performance and no security guarantee in the *non-secure* LLC state and transitions into the *secure* LLC state that offers a strong security guarantee with minimal performance loss. The transition happens the moment Cyclone triggers an alarm about a contention-based LLC attacker. The key idea of Avenger is that it is a non-rigid and scalable approach that does not isolate benign applications.

A. Avenger in Action

Process starts. When an application gets scheduled by the OS or the security monitor, the domain ID is also communicated to the memory hierarchy. This is similar to prior proposals [15], [26], [34]. Figure 4 illustrates the state of an LLC with Avenger. An LLC with Avenger begins with the *non-secure* LLC state, with Cyclone monitoring the LLC state for a possibility of an attack. The OS or the security monitor provides domain IDs for running processes to Cyclone and the LLC controller. The moment Cyclone triggers an alarm to the LLC controller about the possibility of an LLC attack (step 1 of Figure 4), Avenger kicks in and starts the isolation phase (which is a transition phase in between the non-secure and the secure LLC states).

Isolation for Secure LLC. Avenger makes sure the transition phase does not leak information. To achieve the same, it *isolates* the domain ID provided by Cyclone. Avenger uses an LLC way-based isolation, and depending on the number of concurrent domains; it allocates LLC ways. For example, for a 16-way LLC with 16 different security domains, Avenger isolates the attacker domain ID by providing only one LLC way. Similarly, if a 16-way LLC is shared by two, four, or eight co-running domains, Avenger allocates eight, four, and two LLC ways to the attacker domain ID. For the sake of simplicity, we explain Avenger for a 16-way LLC with 16 applications running concurrently. Before isolating an LLC way (say way 0 for the attacker), the LLC controller `purges` all the cache lines that are present in way 0. Similarly, the controller `purges` attacker's cache lines that are present at way 1 to way 15. Purging is a process where the selected cache lines are invalidated. If the purged lines are dirty, it is written back to the DRAM. During the *purging* process, applications are not allowed to access LLC. Once the *purging* process completes, the LLC enters into the secure LLC state (step 2 of Figure 4). Note that in the secure LLC state, the LLC hits, misses, evictions, replacement policy metadata updates, are restricted to isolated region of the attacker only and are not shared between the attacker and the victim(s). Hence, in the secure LLC state, the attacker can not infer the victim's information.

Attacker process exits, and transition from Secure to non-secure LLC. Once the attacker exits, the OS or the security monitor informs the LLC controller, and Avenger initiates a transition from secure LLC to the non-secure LLC (step 3 of Figure 4). Avenger `purges` the one LLC way allocated to the attacker. Note that now there is no need to `purge` benign application's LLC ways as these are not shared by the attacker. Once the *purging* process gets over, the LLC enters into the non-secure LLC state (step 4 of Figure 4). Note that Cyclone runs and monitors LLC irrespective of the LLC state. Note that in the case of multiple attackers, Avenger isolates all the attackers, for example by providing one LLC way for all 15 attackers on a 16-core system. This ensures, with Avenger, there is no limit on the number of concurrent domains that can be active at a given point in time.

DRAM Controller support. As Avenger assigns only one LLC way to the attacker(s), the LLC contention attacker can now mount a denial of service (DOS) attack at the DRAM controller introducing significant LLC-DRAM read and write traffic. We handle this attack by de-prioritizing attacker's requests at the DRAM controller so that it cannot swamp the DRAM scheduler and cause a DOS attack as mentioned in [43]. We also partition the LLC miss status holding registers (MSHRs) so that attacker cannot observe timing difference because of MSHR occupancy. Note that, recent secure LLC partitioning techniques do not take care of performance attacks at the DRAM that can happen as a side-effect of LLC partitioning.

B. Design Choices

Why not an OS-based Avenger? It can be argued that an OS based approach that can kill the attacker process is a better approach than partitioning the LLC. We do not make a case for an OS based approach for the following reason: In case of a *multi-process* attack, once the OS kills one of the attack processes, say P1 (after getting an alert from the attack detector), a new attacker process can be spawned by the attacker, say P2. P2 cannot access the addresses mapped to P1. However, it can still probe the LLC and can deduce LLC contention because of cyclic interference that has happened just before the OS killed P1. Note that this is an extremely difficult attack to mount but certainly not impossible. Avenger, in its current form, instead makes sure no such information can be deduced at the LLC. It can also be argued that an OS can de-schedule the attacker core and can migrate it to another core or another socket, and can `clflush` atacker's cache lines from the LLC. However, frequent descheduling, migrating, and flushing cache lines will lead to a denial of service attack at the LLC. We argue that we should not mitigate one form of LLC attack by creating another.

Isolating LLC ways and not LLC sets. Avenger isolates an attacker at the LLC ways and not at the LLC sets, although both the designs are possible. Avenger uses way-based isolation for two reasons: (i) Modern LLCs use data direct IO technology (DDIO) [41] that reserves few LLC ways for I/O accesses. With LLC set-based isolation, it will not be possible to provide DDIO as the attacker process can cause contention through I/O requests at the reserved LLC ways for DDIO. (ii) Way-based isolation is simpler to implement than LLC set-based as modern LLCs use slices for providing high LLC bandwidth among concurrent applications, and providing isolation at the LLC sets across LLC slices demands a change in the LLC indexing.

Agility of the detector. Cyclone is an agile detector that detects all kinds of LLC contention-based attacks as soon as it detects a cyclic interference between the attacker and victim(s). For our study, we use the fastest eviction-attack algorithm like group elimination [48] and observe that Cyclone detects all kinds of contention-based attacks within 1300 to 1500 cycles with at-least four alerts.

Uncertain future in terms of the LLC attacks. Cyclone can detect all the possible LLC contention attacks proposed so far as all the attacks cause cyclic interference, which is the key. Based on the experiments, we find Cyclone provides 100% accuracy in detecting a cross-core LLC attacker. However,

TABLE I
STORAGE OVERHEAD WITH AVENGER.

	#entries	Size
domain IDs	two eight-bits per tag entry	2 bytes
cyclic-interference counters	16 10-bit counters × 16 LLC slices	2560 bits
direction-interference counters	32 10-bit counters × 16 LLC slices	5120 bits
Buckets	four 6-bit register × 16 buckets	384 bits
Interval counter	9 bit × 16 LLC slices	144 bits
Global detector	pre-trained SVM model	300 bytes
Total	-	**3.2% of the LLC size**

in the future if a new cross-core LLC attack emerges, then Cyclone has to be retrained for cyclic interference thresholds. As Cyclone is a programmable hardware, ideas similar to post-silicon and fabrication [28] [30] microarchitecture can be easily implemented with Cylcone to make it robust as and when new cross-core LLC attacks emerge.

Benign detected as an attacker. Cyclone does not provide false alarms with the benign SPEC CPU 2017 applications thanks to a false positive rate of 0.0000001%. However, if there is a future benign application that gets classified as an attacker then the performance guarantee of Avenger will be similar to prior proposals like DAWG and BCE. Also, the Cyclone detector has to be re-trained keeping the new benign application in mind.

Storage overhead. In terms of storage overhead, Avenger's primary storage requirement is because of the Cyclone. For each cache line, Avenger stores a domain ID in the tag array. For a 16-core system, we store the two recent domain IDs that have accessed a particular LLC way. On a demand access to a particular LLC set, we compare the domain ID of the request that comes from the processor with the two domain IDs that we store per LLC way. This helps in finding the direction of interference ($A \prec V \prec A$ or $V \prec A \prec V$).

Cyclone uses 16 10-bit cyclic interference counters that store the most frequent inter-core interference counts. Cyclone uses 16 buckets, each with four 6-bit history registers. We also use 32 10-bit counters based on the direction of interference. The global detector that we use is a pre-trained support vector machine (SVM) model that takes around 300 bytes. Avenger also uses hardware registers that store the LLC-way numbers assigned to the attacker(s). For example, a 4-bit register for a 16-way LLC with 16 applications running concurrently. In total, Avenger incurs a storage overhead of 3.2%, which is negligible and in the same order as the competing secure LLC partitioning techniques like BCE that incurs a storage overhead of around 2%. Table I provides the details of storage overhead with Avenger.

C. Security Guarantee

Avenger, by design, provides a strong security guarantee as it isolates the attacker when it observes a cyclic interference between the attacker and victim(s). With Avenger, an attacker cannot infer covert information from other domain IDs. For read only shared memory pages, reads, writes, coherence transactions, and *flushes* are restricted within a domain and there is zero possibility of cross-domain interference. For eviction-based attacks, Avenger isolates the attacker within few thousand cycles mitigating cross-domain evictions. Note that the replacement policy metadata update is also restricted, and no cross-domain replacement policy updates are allowed. LLC occupancy-based attacks are also mitigated as the attacker cannot infer anything about victim's working set. With Avenger, the only information that an attacker can infer is the presence of a co-running application(s) because of Avenger's isolation. However, an attacker cannot infer any side-channel information about co-running application(s).

TABLE II
SIMULATED PARAMETERS.

Core	16 Out-of-order cores, hashed perceptron branch predictor, 4GHz with 6 issue width, 4 retire width, 352 entry ROB
TLBs	64 entry 4-way at L1 DTLB/ITLB (1 cycle), 2048 entries 16-way L2 STLB (8 cycles)
MMU Caches	2 entry (PSCL5), 4 entry (PSCL4), 8 entry (PSCL3), 32 entry (PSCL2), searched parallely, one cycle
L1	32KB 8-way L1I (4 cycles), 48KB 12-way L1D (5 cycles), 16 MSHRs
L2	512KB 8-way associative (10 cycles), RRIP, 32 MSHRs [23]
LLC	32 MB (2MB/slice), 16-way (20 cycles), RRIP [23], 64 MSHRs per slice
DRAM	1 channel/4-cores, 6400 MT/sec per channel

TABLE III
BENCHMARKS WITH THEIR LLC MPKIS.

Benchmark	**LLC MPKI**
mcf-1554B	127
mis-85B	56
lbm-4268B	55
roms-1390B	23
sssp-5B	20
xalan-202B	17
wrf-6673B	14
omnetpp-874B	9
gcc-734B	9
perl-570B	0.06
leela-1083B	0.01

IV. EVALUATION

We use a modified version of ChampSim [10], a trace-driven simulator used for the 2nd and 3rd Data Prefetching Championships (DPC-2 [6] and DPC-3 [9]), and 2nd cache replacement championship (CRC-2) [8]. Table II summarizes our simulated parameters, mimicking an Intel Sunny Cove microarchitecture [2].

Benchmarks and workloads. Table III shows selected benchmarks with their respective LLC MPKIs. We make sure we use benchmarks with different memory intensities (LLC MPKIs varying from 0.01 for `leela` to 127 for `mcf`). For our multicore evaluation, we use 18 16-core representative multiprogrammed mixes created from SPEC CPU2017 [12] and GAP [11] benchmarks with one core running an agile

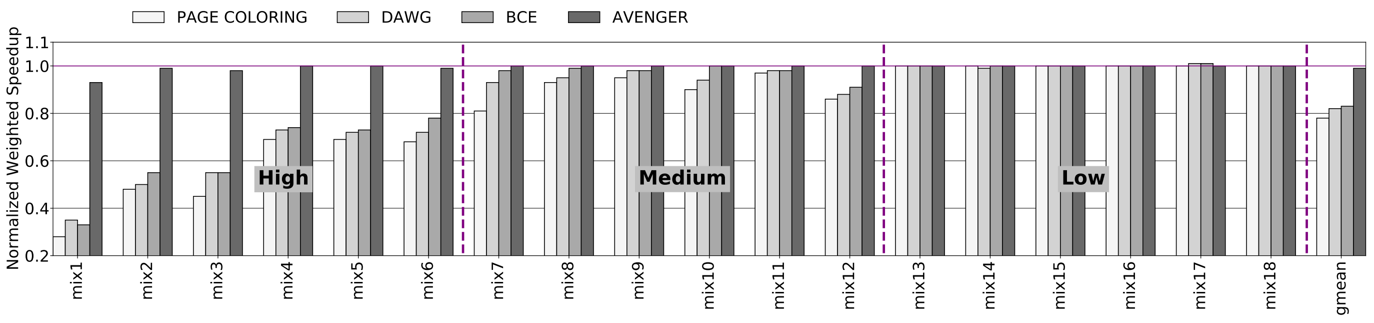

Fig. 5. Normalized performance (in terms of weighted speedup of benign applications) for different LLC partitioning techniques on a 16-core system with attacker running on one core. Bars below 1.0 represent performance slowdowns.

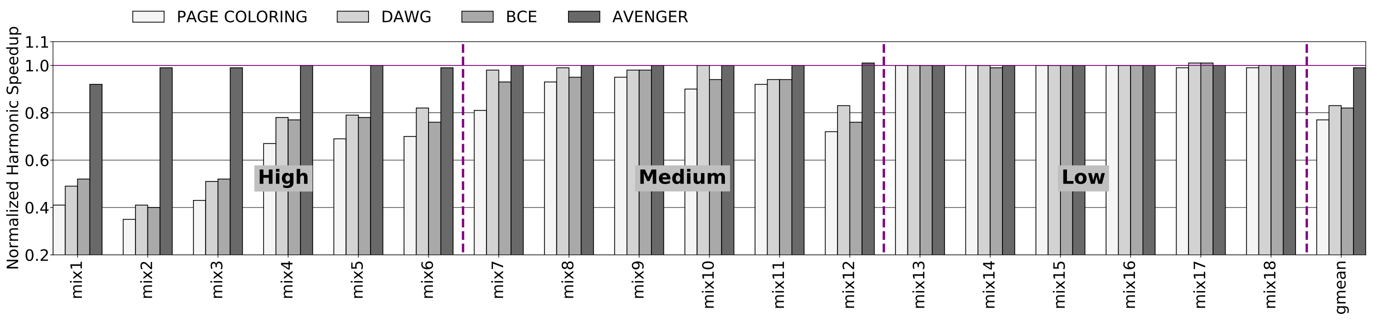

Fig. 6. Normalized performance (in terms of harmonic mean of speedups of benign applications) for different LLC partitioning techniques on a 16-core system with attacker running on one core. Bars below 1.0 represent performance slowdowns.

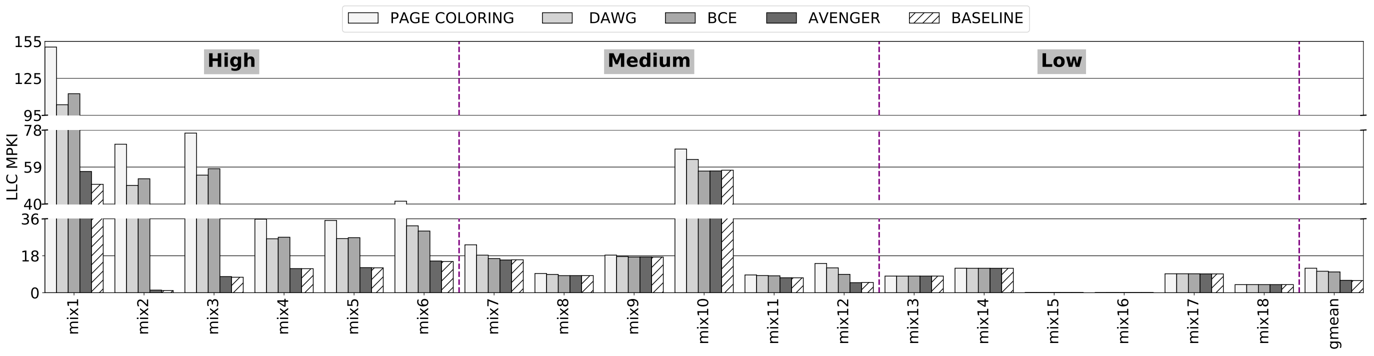

Fig. 7. Average LLC MPKIs per mix for different LLC partitioning techniques.

eviction-based attacker that uses group elimination method (GEM) [48]. Due to space constraints, instead of showing results for more than thousands of mixes, we show detailed results only for the representative mixes. We divide these mixes into three bins (high, medium, and low) based on their sensitivity to LLC partitioning ideas(refer Table IV). We label a mix as high, medium, and low if the performance drop, with partitioning, falls into these three respective categories: more than 20%, in between 1 to 20%, and less than 1%. To ensure a fair distribution of mixes among high, medium, and low, we use six mixes from each category, 18 representative mixes.

Performance metrics. We evaluate Avenger on a 16-core simulated system. For multi-core simulations, we warm-up the caches for 30M instructions per core (480M instructions per mix) and then report performance in terms of the normalized weighted-speedup and fairness with performance (in terms of harmonic mean of speedups) that are defined as follows: weighted speedup $=\sum_{i=0}^{i=N-1} \frac{IPC_{together}(i)}{IPC_{alone}(i)}$ and harmonic mean of speedup = $\frac{N}{\sum_{i=0}^{i=N-1} \frac{IPC_{alone}(i)}{IPC_{together}(i)}}$. These speedups are normalized to a non-secure baseline with no LLC partitioning for the instructions in the respective region of interests. $IPC_{together}(i)$ is the instructions per cycle (IPC) of core i when it runs along with other N–1 applications on an N-core system. $IPC_{alone}(i)$ is the IPC of core i when it runs alone on a multi-core system of N cores. We simulate a mix until each benchmark has executed its region of interest instructions. When a benchmark finishes, it gets replayed until all the

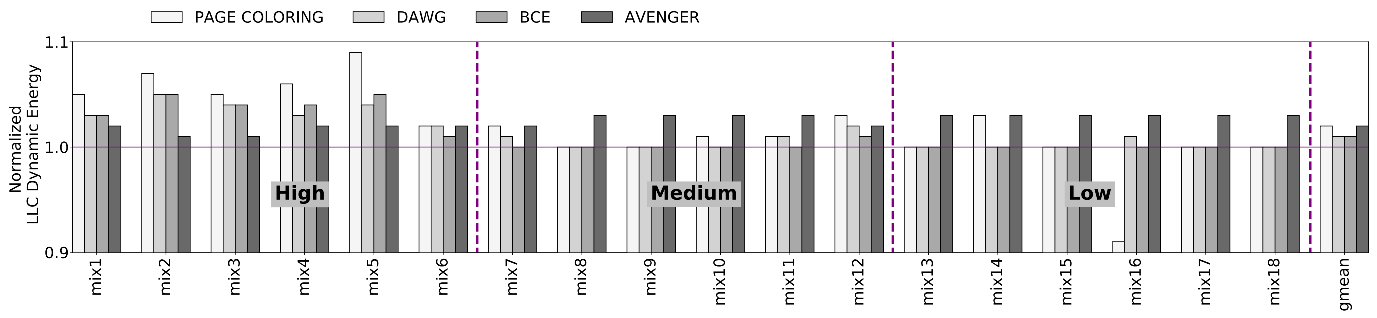

Fig. 8. Normalized dynamic energy at the LLC for different LLC partitioning techniques normalized to non-secure baseline. Bars above 1.0 represent energy overhead.

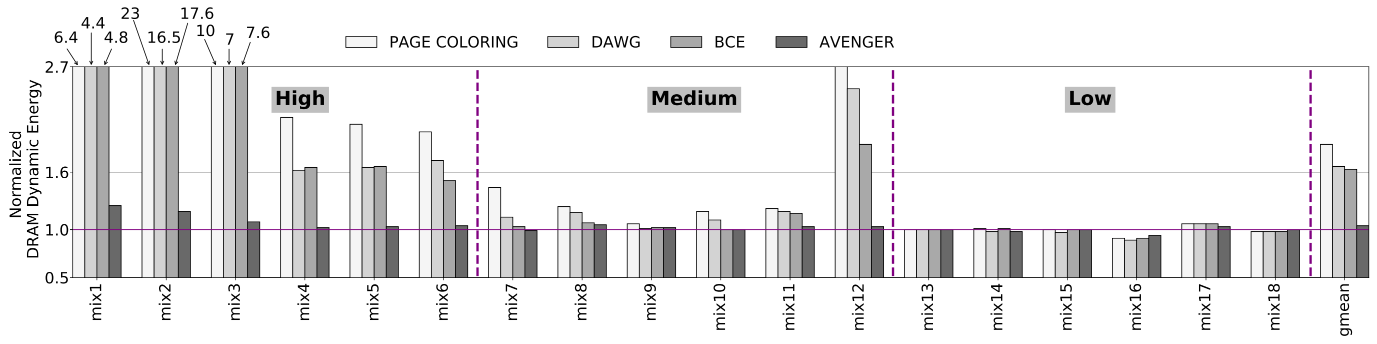

Fig. 9. Normalized dynamic energy at the DRAM for different LLC partitioning techniques normalized to non-secure baseline. Bars above 1.0 represent energy overhead.

TABLE IV
REPRESENTATIVE MIXES GROUPED INTO THREE BINS DENOTING HIGH, MEDIUM, AND LOW SENSITIVITY TO PARTITIONING TECHNIQUES, WITH ONE CORE RUNNING THE ATTACKER.

Mix	Composition	Bin
mix1	mcf(15)	High
mix2	mcf(7)-perl(8)	High
mix3	mcf(7)-xalan(2)-omnetpp(2)-wrf(2)-perl(2)	High
mix4	xalan(7)-mcf(2)-omnetpp(2)-wrf(2)-perl(2)	High
mix5	wrf(7)-mcf(2)-xalan(2)-omnetpp(2)-perl(2)	High
mix6	sssp(7)-mcf(2)-xalan(2)-omnetpp(2)-wrf(2)	High
mix7	xalan(15)	Medium
mix8	omnetpp(15)	Medium
mix9	wrf(15)	Medium
mix10	mis(15)	Medium
mix11	wrf(7)-perl(8)	Medium
mix12	sssp(7)-perl(8)	Medium
mix13	gcc(15)	Low
mix14	lbm(15)	Low
mix15	perl(15)	Low
mix16	leela(15)	Low
mix17	roms(7)-perl(8)	Low
mix18	gcc(7)-perl(8)	Low

benchmarks finish their respective regions of interest.

Energy model. We report the dynamic energy consumption of the LLC and DRAM. We obtain the energy consumption of tag accesses, reads, writes, and fills to LLC and DRAM with PCACTI [1] and Micron DRAM power calculator [7]. Then, we compute the total energy expenditure by accounting for the number of accesses of each type at the LLC and DRAM. We use 7nm process technology for our energy calculations.

Evaluated LLC partitioning Techniques. We compare the effectiveness of Avenger with three secure LLC partitioning techniques: page coloring, DAWG, and BCE, normalized to a non-secure non-partitioned LLC. We compare these techniques based on performance (weighted speedup), performance with fairness (harmonic mean of speedups), and dynamic energy consumption at the LLC and the DRAM. As static energy can be correlated with the performance slowdown;we do not show static energy overhead, explicitly.

A. *Performance and energy overhead*

Weighted and Harmonic mean of speedups. Figure 5 shows the performance overhead with the Avenger along with the competitive LLC partitioning techniques. On average, Avenger provides performance closer to the non-secure baseline (less than 1% overhead) than the competitive secure LLC partitioning techniques that lead to an average performance overhead of more than 17%. The maximum performance overhead with Avenger is 6% (for mix1), whereas competing techniques provide performance overheads as high as 72%. Figure 6 shows the performance overhead, keeping performance and fairness in mind. Even with harmonic mean of speedups, Avenger significantly outperforms all the competing techniques. One of the primary reasons for showing both weighted and harmonic mean of speedups is that Avenger is effective with performance and fairness consistently across all

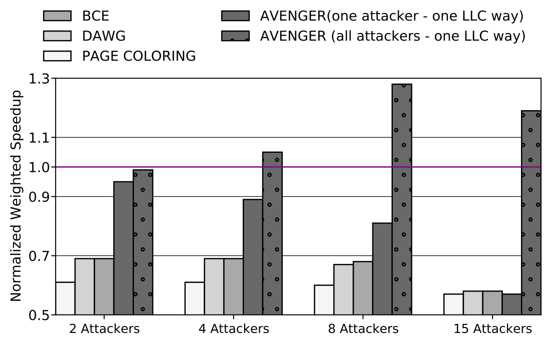

Fig. 10. Normalized speedup with multiple attackers.

the mixes. However, that is not the case for other techniques. For example, for mix12, BCE is better than DAWG, and page coloring in terms of weighted speedup (Figure 5) and DAWG is better than BCE if we consider performance along with fairness (Figure 6). Figure 7 shows the average LLC MPKI per mix, which correlates strongly with the performance improvement as shown in Figure 5.

In summary, for mixes, not sensitive to LLC space and partitioning decisions (the mixes labeled as low), there is no difference or marginal difference in performance among all the partitioning techniques. However, for mixes labeled as medium and high, there is a significant difference in performance primarily because of an increase in LLC conflict misses as shown in Figures 1 for mix2. Page coloring fails to provide enough LLC space because LLC allocation is driven by the page allocation at the DRAM. DAWG and BCE provide an equal amounts of space (2MB per application) but due to the rigid and fix allocation of LLC space for each application, these techniques increase LLC misses compared to a non-secure non-partitioned LLC. Avenger, on the other hand, provides LLC misses closer to the non-secure baseline as only one LLC way is reserved by the attacker.

Memory sub-system energy. Figures 8 and 9 show the effect of LLC partitioning techniques on the dynamic energy at the LLC and DRAM. Note that the static energy consumption correlates strongly with the performance overhead. So, we show the dynamic energy per LLC and DRAM accesses normalized to a non-secure baseline. Mixes labeled as highly sensitive to LLC partitioning with high LLC MPKI incur significant DRAM energy overhead as high as 23 times for mix2. Note that DRAM energy is a function of both DRAM read and DRAM write energy. For write-intensive benchmarks (`lbm`) with more than 40% of the LLC blocks that are dirty, frequent LLC misses causing frequent write-backs increasing the write energy.

Compared to DRAM energy, LLC energy overhead is within 9% and mixes with high MPKIs incur high LLC energy overhead primarily because of increase in LLC fills. Note that we include the energy consumed by the cyclic interference counters of Cylone at the LLC. On average, BCE, DAWG, and page coloring incur an energy overhead of more than 63% and 1% at the DRAM and LLC, while Avenger incurs an overhead of 4% and 2% for DRAM and LLC, respectively. There is an outlier in the form of mix 16 in Figure 8 where the LLC energy improved a lot with page coloring. One of the primary reasons for this trend is that mix 16 has extremely low LLC MPKI, and with page coloring, the page allocation changes at the DRAM providing more L2 hits, causing relatively lesser LLC accesses. The energy overhead of Avenger at the LLC is primarily because of the extended tag entry that stores last two domain IDs. Overall, Avenger is the most energy-efficient and secure LLC partitioning technique among competing techniques.

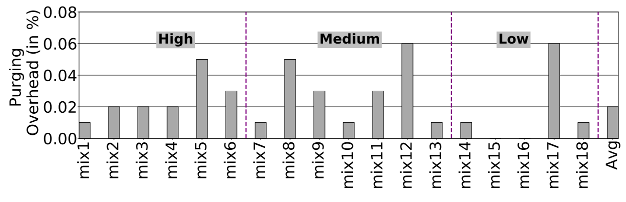

Fig. 11. Purging overhead on performance.

Table V summarizes performance, performance with fairness, memory subsystem energy, and storage overhead with Avenger, page coloring, DAWG, and BCE. Avenger provides security guarantee with minimal performance, energy, and storage overheads.

Effect of Multiple attackers. So far, we have shown Avenger with a single-threaded attacker. However, for mounting a cross-core LLC attack, an attacker process can use multithreading where multiple cores execute attacker code or multiple independent attackers to perform DOS or occupancy-based attacks. In the case of a multi-threaded attacker, all the threads get one LLC way as all the threads will be part of one domain ID. However, for multi-attacker scenario, Avenger has two choices: (i) allocate one LLC way per attacker and (ii) allocate one LLC way for all the attackers. Figure 10 shows the effect of multiple attackers on performance (weighted speedup). For eight and 15 attackers, BCE, DAWG, and page coloring provide eight different isolated chunks of LLC, effectively hampering performance of co-running benign applications. Avenger, on the other hand, allocates one LLC way for all the attackers to improve the performance of benign applications. With Avenger, when we move from eight attackers to 15 attackers, the performance improvement goes down as the baseline performance of the benign application improves thanks to inter-attacker interference at the shared resources like LLC and DRAM.

Purging Overhead on performance. While transitioning from a non-secure LLC state to the secure LLC state, Avenger purges LLC lines of interest, ensuring an isolation boundary with strong security guarantee. Figure 11 shows purging overhead in terms of LLC stalls, which is less than 0.1%. Note that purging happens only twice (once in the beginning while transiting from non-secure LLC to secure LLC and again while transitioning from secure LLC to non-secure LLC). We also quantify the purging overhead by varying DRAM write

TABLE V
SUMMARY OF PERFORMANCE, ENERGY, AND STORAGE OVERHEADS (IN %) AVERAGED ACROSS 18 MIXES.

Technique	Performance	Fair performance	DRAM energy	LLC energy	Storage
Page coloring	22%	23%	89%	2%	$< 0.5\%$
DAWG	18%	17%	66%	1%	$< 0.5\%$
BCE	17%	18%	63%	1%	2%
Avenger	1%	1%	4%	2%	3.2%

queue sizes and find that the overhead is marginal.

Effect of hardware prefetchers. Modern processors use hardware prefetchers to hide costly DRAM access latency. We evaluate competitive secure LLC partitioning techniques in the presence of state-of-the-art hardware prefetchers like IPCP [44], SPP [25], and Bingo [14] at L1 and L2 caches, respectively. All the techniques show similar performance overheads (with minor performance change of less than 2%) for the baseline with prefetching as compared to the baseline with no prefetching. BCE and Avenger are equally effective in the presence of hardware prefetchers. However, DAWG and page coloring lose their effectiveness primarily because of prefetcher caused LLC pollution.

V. Related Work

In this Section, we make a qualitative comparison with relevant related works.

LLC partitioning. CATalyst [32] partitions the LLC into insecure and secure partitions. Also, within the secure partition, it prevents the replacement of cache blocks that store the secure data. CATalyst demands changes to the programming language and run-time. With CATalyst, The number of partitions is limited by the number of DRAM pages in a partition, and limitations are similar to a page coloring approach. Also, CATalyst does not prevent LLC hit-based replacement policy attack as the underlying partitioning is motivated by Intel CAT [22] (DAWG prevents hit-based replacement policy attack). Secure DCP [37] NoMo [17] isolate an attacker, which is similar to Avenger. However, these techniques do not mitigate cache occupancy attacks because of dynamic LLC partitioning. Also, the scalability of SecDCP is limited by the number of LLC ways. In contrast, Avenger is not limited by the number of LLC ways as all the attackers get only one LLC way instead of multiple ways. Jumanji [50] isolates VMs at the LLC, another rigid approach. Jumanji does not mitigate attacks like occupancy-based attacks and cross-domain flush-based attacks. Also, it does not differentiate between processes that are part of the same VM.

Other approaches. SHARP [40], BITP [45], RIC [24], and Seclusive cache [20] provide probabilistic security guarantees and do not mitigate all kinds of contention-based cross-core LLC attacks. The SHARP policy has a lot of loopholes and is not secure [27]. HybCache [19] proposes a fully associative LLC for mitigating conflict-based attacks. However, the approach incurs implementation complexity, especially for large LLCs. For example, implementing a fully-associative mapping for one-way LLC would require concurrent access to thousand of cache lines per LLC access that would considerably increase the cache energy consumption. Also, HybCache cannot mitigate the occupancy-based attack.

VI. Conclusion

We proposed Avenger, an LLC partitioning technique that isolates the attacker with the help of an attack detector. Avenger is driven by the design principle that the attacker should be isolated and not the victims, which is in contrast to the recently proposed secure LLC partitioning techniques that isolate all the applications, incurring performance and energy overhead. Avenger is a flexible and non-rigid LLC partitioning technique that uses a hardware programmable attack detector. The detector can be trained by organizations and cloud providers as per the security requirements. Avenger outperforms the state-of-the-art secure LLC partitioning techniques and provides performance closer to a non-secure baseline. Overall, Avenger is a lightweight, secure, high performing, and energy-efficient LLC partitioning technique that provides a sweet spot in terms of security, and overheads in terms of performance and energy.

VII. Acknowledgement

We would like to thank Mainak Chaudhuri, Gururaj Saileshwar, and CARS members for their feedback on the draft.

References

[1] Pcacti tool, Online. Available: https://sportlab.usc.edu/downloads/.
[2] "Examining intel's ice lake processors: Taking a bite of the sunny cove microarchitecture," Available at https://www.anandtech.com/show/14514/examining-intels-ice-lake-microarchitecture-and-sunny-cove.
[3] "Examining intel's ice lake processors: Taking a bite of the sunny cove microarchitecture," *Available at https://www.anandtech.com/show/14514/examining-intels-ice-lake-microarchitecture-and-sunny-cove*.
[4] "Intel sgx," Available at https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html.
[5] "Intel sgx," Available at https://developer.arm.com/ip-products/security-ip/trustzone.
[6] "The 2nd data prefetching championship (dpc-2)," Jun. 2015. [Online]. Available: https://comparch-conf.gatech.edu/dpc2/
[7] "Micron dram power calculator," https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf, Dec. 2015.
[8] "The 2nd cache replacement championship (crc-2)," Jun. 2017. [Online]. Available: https://crc2.ece.tamu.edu/
[9] "The 3rd data prefetching championship (dpc-3)," Jun. 2019. [Online]. Available: https://dpc3.compas.cs.stonybrook.edu/
[10] Online. Available: https://github.com/ChampSim/ChampSim, Champsim Simulator.
[11] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.
[12] p. y. Bucek et al., booktitle=Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, "Spec cpu2017: Next-generation compute benchmark."

[13] M. Chaudhuri, "Zero directory eviction victim: Unbounded coherence directory and core cache isolation," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 277–290.

[14] B. et al., "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 399–411.

[15] B. et al., "Mi6: Secure enclaves in a speculative out-of-order processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 42–56.

[16] C. et al., "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 857–874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[17] D. et al., "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," vol. 8, no. 4, jan 2012. [Online]. Available: https://doi.org/10.1145/2086696.2086714

[18] F. et al., "A noise-resilient detection method against advanced cache timing channel attack," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 237–241.

[19] G. et al., "HybCache: Hybrid Side-Channel-Resilient caches for trusted execution environments," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 451–468. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/dessouky

[20] G. et al., "Seclusive cache hierarchy for mitigating cross-core cache and coherence directory attacks," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 637–640.

[21] H. et al., "Cyclone: Detecting contention-based cache information leaks through cyclic interference," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 57–72.

[22] H. et al., "Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 657–668.

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

[24] K. et al., "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[25] K. et al., "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[26] K. et al., "Dawg: A defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 974–987.

[27] K. et al., "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

[28] K. et al., "Post-silicon microarchitecture," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 26–29, 2020.

[29] K. et al., "Damaru: A denial-of-service attack on randomized last-level caches," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 138–141, 2021.

[30] K. et al., "Post-fabrication microarchitecture," in *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 2021, pp. 1270–1281. [Online]. Available: https://doi.org/10.1145/3466752.3480119

[31] L. et al., "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 605–622.

[32] L. et al., "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 406–418.

[33] R. et al., "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: Association for Computing Machinery, 2009, p. 199–212. [Online]. Available: https://doi.org/10.1145/1653662.1653687

[34] S. et al., "Bespoke cache enclaves: Fine-grained and scalable isolation from cache side-channels via flexible set-partitioning," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 37–49.

[35] S. et al., "Effective mimicry of belady's min policy," in *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture*, ser. HPCA '22. South Korea: Association for Computing Machinery, 2022, p. 1–15.

[36] S. et al., "Robust website fingerprinting through the cache occupancy channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 639–656.

[37] W. et al., "Secdcp: Secure dynamic cache partitioning for efficient timing channel protection," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[38] W. et al., "Randomized last-level caches are still vulnerable to cache side-channel attacks! but we can fix it," in *Proceedings - 2021 IEEE Symposium on Security and Privacy, SP 2021*, ser. Proceedings - IEEE Symposium on Security and Privacy. United States: Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 955–969.

[39] W. et al., "Scattercache: Thwarting cache attacks via cache set randomization," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 675–692.

[40] Y. et al., "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: Association for Computing Machinery, 2017, p. 347–360. [Online]. Available: https://doi.org/10.1145/3079856.3080222

[41] Y. et al., "Don't forget the i/o when allocating your llc," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 112–125.

[42] C. Kyrkou and T. Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, 2012.

[43] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in Multi-Core systems," in *16th USENIX Security Symposium (USENIX Security 07)*. Boston, MA: USENIX Association, Aug. 2007. [Online]. Available: https://www.usenix.org/conference/16th-usenix-security-symposium/memory-performance-attacks-denial-memory-service-multi

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

[45] B. Panda, "Fooling the sense of cross-core last-level cache eviction based attacker by prefetching common sense," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 138–150.

[46] M. Payer, "Hexpads: A platform to detect "stealth" attacks," in *Engineering Secure Software and Systems - 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*, ser. Lecture Notes in Computer Science, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds., vol. 9639. Springer, 2016, pp. 138–154. [Online]. Available: https://doi.org/10.1007/978-3-319-30806-7_9

[47] M. K. Qureshi, "Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 775–787.

[48] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 360–371. [Online]. Available: https://doi.org/10.1145/3307650.3322246

[49] G. Saileshwar and M. Qureshi, "{MIRAGE}: Mitigating conflict-based cache attacks with a practical fully-associative design," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[50] B. C. Schwedock and N. Beckmann, "Jumanji: The case for dynamic NUCA in the datacenter," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 2020, pp. 665–680. [Online]. Available: https://doi.org/10.1109/MICRO50266.2020.00061

[51] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.