Mitigating Shift-Based Covert-Channel Attacks in Racetrack Last Level Caches

Lei Zhao
University of Pittsburgh
lez21@pitt.edu

Youtao Zhang
University of Pittsburgh
zhangyt@cs.pitt.edu

Jun Yang
University of Pittsburgh
juy9@pitt.edu

ABSTRACT

Racetrack memory (RM, also known as DWM (domain wall memory)) is an emerging memory technology that has many advantages such as low power, high density, and low access latency. Recent studies have shown that it is promising to architect RM as last level cache (LLC). Given that a RM track consists of m domains (for storing m bits data) and n access heads ($1 \leq n < m$), one RM access often requires multiple hops of shift to move the access head above the domain to be accessed. This leads to variant access latency, which may be exploited to initiate covert channel attacks to leak sensitive information in secure computing environment. In this paper, we elaborate the feasibility of such attacks and propose secure head management policies to effectively mitigate the attacks in RM LLCs. Our experimental results show that the proposed schemes can reduce the new discovered shift-based covert channel's capacity by up to 260 times with modest performance overhead.

Keywords

Racetrack Memory, Covert Channel, Last Level Cache

1. INTRODUCTION

Chip multiprocessors (CMPs) are widely adopted in modern computer systems, making it crucial to integrate large capacity last level cache (LLC) to achieve scalable performance improvement. However, SRAM, the traditional memory technology for constructing onchip caches, faces severe density, leakage and reliability problems [8]. Emerging memory technologies such as STT-MRAM [3], PCM [21], and racetrack memory [11, 15], have been proposed to construct LLCs. Racetrack memory (RM), due to its low power, high density, and low access latency, is considered as one of the most promising SRAM supplement in future LLCs [15, 16].

To maximize system performance, the multiple co-running threads on a CMP need to share onchip hardware resources, such as LLCs and communication buses. Resource sharing introduces conflicts and leads to service time fluctuation, e.g., an access to a shared cache may be a hit if there is no other thread accessing the target cache set, or a miss if the cache set was flushed by another thread before the access. *Timing channel attack* [19] exploits this difference to reveal the status of the shared resource so that sensitive information can be leaked. In the cases if one thread cannot communicate with the remote attacker directly, e.g., being restricted from using network, a hidden data path, referred to as *covert channel*, can be constructed between two co-running threads. The first thread leaks the information through covert channel to the second thread, and then the second thread sends the information to the remote attacker. [12, 13] presented covert channel attacks that exploit L2 cache misses. Due to low channel efficiency, the bit rate of the covert channel is low so that practically only small secret data such as private keys can be leaked.

In this paper, we focus on mitigating covert channel attacks in RM LLCs. Given that a RM track consists of m domains (for storing m bits data) and n access heads ($1 \leq n < m$), an access to a track domain often requires shift operations to move the access head above the domain to be accessed. By measuring the latency difference of consecutive cache accesses, a covert channel can be constructed between two co-running threads. Our study shows that the bit rate of such a covert channel is 260 × that of the conventional channel exploiting LLC cache misses. The existence of this high capacity covert channel may become a major concern when adopting racetrack LLCs in server environments.

To mitigate covert channel attacks, we propose secure head update approaches with tradeoffs between channel capacity and performance overhead. In covert channel attacks, the sender is usually a Trojan program attached to the victim thread that possesses secure data; the receiver does not have the access directly. Given that the thread having secure data is often at high security level, covert channel leaks information from high security level thread (sender) to low security level thread (receiver). Based on this observation, we propose to partition co-running threads to multiple security levels and defend covert channel attacks by prohibiting information flowing from high to low levels. Given RM LLCs effectively reduce cache misses, we then propose an epoch based approach to exploit cache access locality to improve cache performance. In this design, the capacity of potential covert channels is linear to the length of the epoch, making it flexible to adjust at runtime to match the security demand of the system.

We summarize our contributions as follows.

- We elaborate the details of covert channel attacks in RM LLCs. Our quantitative analysis shows that the capacity of the channel can be 260 times higher than that of the conventional cache miss based channel. To the best of our knowledge, this is the first work that studies timing channel attacks introduced by shift operations in RM memory.
- We propose two mitigation schemes to defend this new covert channel attack. We evaluate the proposed schemes and study their effectiveness under different settings. The low cost head update policies help to achieve good tradeoffs among performance, security levels, and information leakage rate.

The rest of paper is organized as follows. We discuss the background in Section 2. We elaborate the attack details and present our schemes in Section 3 and Section 4, respectively. The experimental methodology and results are discussed in Section 5. Section 6 discuss additional related work. We conclude the paper in Section 7.

2. BACKGROUND

In this section, we briefly discuss RM basics and then introduce traditional miss based timing channel attacks.

2.1 Racetrack Memory

Basic structure. Racetrack memory (RM) is an emerging nonvolatile memory that exploits spintronic magnetoresistive effect to store data on tracks. Figure 1 shows the structure of a RM track that contains multiple domains (blue parts) separated by domain walls (red parts). One bit is saved in one track domain.

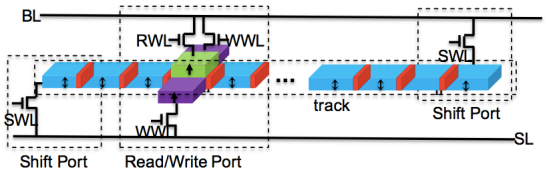

Figure 1: The structure of a RM track. BL: bit line. SL: source line. RWL: read wordline. WWL: write wordline. SWL: shift wordline.

There can be one or multiple read and write heads associated with each track. The data stored in the domain is read out through a fixed layer (the green part) on top of it. To write data into a domain, current is injected to the two fixed layers (the purple parts) beside the track, so that the bit stored in one of the two fixed layers is shifted into the domain on the track [17].

Because of area restictions, the number of read/write heads are much smaller than that of domains. Multiple adjacent domains has to share one head. Moving the target domain under the head is performed by shift operations, i.e., by injecting current through two shift ports at the ends of the track, all domains can be shifted in either direction along the track in lock-step fashion.

Racetrack based LLC. In recently proposed RM LLCs [15, 16], RM is exploited to replace SRAM data array while the tag array still uses SRAM in order to perform fast parallel tag lookup. For a 16-way set associative cache, the data of one cache set includes 16 cachelines and each cacheline consists of 512 bits (or 64B). Assuming each track saves 64 bits and has 4 access heads (for both read and write). These bits are saved in multiple tracks as shown in Figure 2.

We adopt the common sourceline cell array organization as in [25] such that one cache set is saved to 128 tracks with each track saving 4 bits from each cacheline in the set. The 4 bits of each line are assigned to different heads in lock-step such that they are accessed simultaneously. Conceptually, all the 128 4-head 64-domain tracks can be abstracted as a single-head 16-domain 512-bit/domain super track, as shown Figure 2(b).

In the rest of the paper, we use the super track abstraction to simplify the discussion.

Head Update Policy. The head update policy determines the head position after each access. Two widely adopted policies are eager policy and lazy policy [15]. The eager policy moves the head back to a fixed position, e.g., the middle of the 16 domains, while the lazy policy leaves the head at the position after each access. The lazy policy usually achieves better performance as it can adapt to the data locality on the track, but requires slightly more complex logic to record the track positions.

2.2 Cache Miss based Covert Channel Attacks

The system that we target at is a CMP system that has multiple co-running threads sharing the onchip hardware resources such as LLCs. By launching timing channel attacks, a malicious thread closely monitors its own execution from which it tries to infer sensitive information from the co-running victim thread. There exist two types: 1) for side channel attacks, the malicious thread and victim thread are in different security domains. The victim thread leaks information unintentionally through contended shared resources. 2) for covert channel attacks, the malicious thread (sender) has already extracted the secret data, but was restricted from sending the data out, e.g., OS does not allow this thread to communicate with remote servers. In this case, the malicious thread (sender) fabricates its access pattern to the shared resource so that its observable behavior can be sensed by co-running thread (receiver). This paper targets at covert channel attacks.

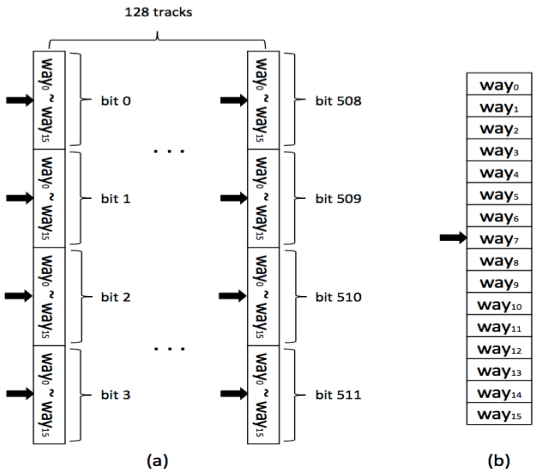

Figure 2: The track organization of one cache set and its super track abstraction.

1). *Prime-probe attack*[10]. In this attack, one thread fills in the cache sets with its own data and waits for the other thread to execute and evict entries from the cache. Then, the first thread accesses its data again and, based on if an access is a cache miss or not, reveals the information about the second thread.

2). *Flush-reload attack*[23]. When two threads share part of the memory address space, one thread may flush the cache with its private data and wait for the other thread to execute for a period of time. The first thread then tries to access the shared data and, based on if the data is in the cache or not, reveals the information about the second thread.

3). *Evict-time attack*[10]. In this attack, one thread may evict the data of a specific cache set, and then triggers and measures the execution of the other thread. The first thread keeps doing this by evicting different cache sets and, based on the observed difference of execution time, reveals which cache sets have been visited by the other thread.

3. THE SHIFT BASED COVERT CHANNELS

In this section, we elaborate the existence of a new timing channel based on the access latency difference of cache hit operations and compare its channel bit rate with the conventional cache miss based covert channels. We focus on covert channels since they can represent the upper bound of side-channel capacities [4].

3.1 Attack Details

To service a cache access to RM LLC, the access head first needs to be shifted above the domain to be accessed. The further the domain is from the current head position, the more number of hops the shift operation needs to perform. To exploit this in covert channel attack, a malicious thread can continuously read the same data item on a track and get the data with minimal latency only if there is no other threads

accessing the same track. We next discuss how to exploit this attack in details.

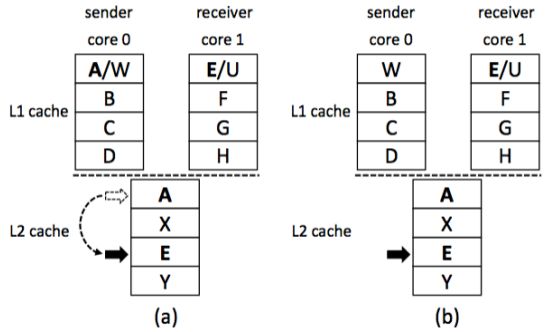

Figure 3: Sending bit'1' and '0'.

Figure 3(a) and (b) illustrate how the sender sends a bit '1' and '0' to the receiver respectively. In this example, we assume both L1 cache and L2 cache are 4-way associative. For simplicity, only one set is drawn for each cache. Data A, B, C, D and W map to core 0's L1 cache, data E, F, G, H and U map to core 1's L1 cache, and A and E are stored in the same set (i.e. on the same track) in L2 cache.

To send a '1', the sender access data in a sequence of WBCDAWBCDA... The sub sequence of WBCD is to ensure that A is evicted from L1 cache, so that every time accessing A, the data will be retrieved from L2 cache. The receiver accesses his own data UFGHEUFGHE... The sub sequence of UFGH is to make sure E will be fetched from L2 cache every time. Since both sender and receiver need to access the same track in L2 cache, the receiver will experience shift operations when accessing E.

To send a '0'. the sender only accesses WBCDWBCD... without accessing A, so that every access will be hit in L1 cache. The receiver still accesses the same sequence as in the previous case. Since no process is competing the track head with the receiver, accessing E will be faster.

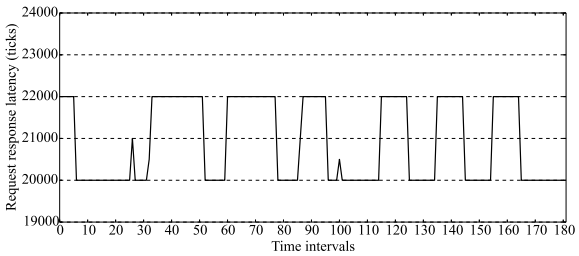

Figure 4: The observed cache hit latencies.

Figure 4 shows the latency of receiver's accesses when sender sends 100011011010010101. The x axis indicates the execution time. The y axis indicates the access latency recorded in counter registers. As shown in the figure, the waveform of the access latency matches the data pattern closely. We next analyze the severity of this attack by evaluating the capacity of the covert channel.

3.2 Shift Based Covert Channel Capacity

In this paper, we adopt *bit rate* as a metric to measure the capacity of the shift based covert channel, similar to that in [22]. The bit rate can be estimated using the sum of the time for the receiver to prepare the target sets with its own data ($T_{receiver}$), and the time for the sender to write the bit into the sets (T_{sender}), that is,

$$BitRate = \frac{1}{T_{receiver} + T_{sender}} \quad (1)$$

We use T_{hit} as the hit latency of the last level cache, and T_{mem} as the latency to fetch data from main memory on a cache miss. In addition, we assume the number of target sets is S with W ways in each set (W-way associative).

For cache miss-based covert channel attacks, the receiver needs to access all the lines in every target set so that $T_{receiver}$ and T_{sender} are estimated as follows.

$$T_{receiver} = T_{send} = T_{mem} * S * W \quad (2)$$

As a comparison, for shift based covert channel attacks, the latency difference can be detected after moving the heads of all target sets to specific positions. Thus, $T_{receiver}$ and T_{sender} can be estimated as:

$$T_{receiver} = T_{send} = T_{hit} * S \quad (3)$$

Note that since both sender and receiver only need to move the accessing head away, there is no need to evict each other's data, if we ignore the noise caused by other processes, both sender and receiver should experience cache hits. Thus we use T_{hit} instead of T_{mem} in the equation.

Since we do not assume either process has access to the virtual-to-physical address mappings of the system, there is no way to make the sender and reciever agree on a specific cache set as the target. Similar to that in [22], we divide the LLC into two parts and use half of the total sets as the target sets. The sender and receiver both allocate a consective memory space with the same size as the LLC. After multiple rounds, the two processes can determine the boundary of the target and untarget cache sets by exploring the maximum resolution.

Given a 32MB 16-way associative LLC with $12ns$ hit latency and $200ns$ miss latency, there are 32768 cache sets in total (16384 target sets). The bit rates of shift based and cache miss based covert channels are computed as follows, respectively.

$$BitRate_{miss-based} = \frac{1}{2 * 200ns * 16 * 16384} = 9.5bps \quad (4)$$

$$BitRate_{shift-based} = \frac{1}{2 * 12ns * 16384} = 2543bps \quad (5)$$

It is unfortunate that the capacity of shift based covert channel is 260 times that of cache miss based channel. With this difference, leaking one single medical record of 256KB (or 2M bits) needs 61.3 hours if using the cache miss based covert channel, but only 13.7 minutes if using the shift based covert channel. As another example, to leak a diagnosis video of 500MB, an attacker needs to maintain this channel stable for 14 years and 19 days, respectively.

There are two reasons that shift based covert channels have high bit rates: (i) the attacker needs to configure the target cache sets, i.e., set up the covert channel, before each attack. For the miss based attacks, the receiver fills in all lines of all target cache sets. For the shift based attacks, the receiver only needs to access one line in each set, which is much faster than the former approach. (ii) The receiver needs to measure the timing difference of accessing the contended resource. It takes much longer time, i.e., several hundreds of cycles, to differentiate cache misses from hits. For shift based attacks, consecutive cache hit accesses can be differentiated such that it is much faster.

4. THE PROPOSED MITIGATION SCHEMES

In this section, we propose schemes to mitigate the shift based covert channel attacks in RM LLCs. We first present the attack model and then our designs.

4.1 The Attack Model and Simple Mitigation

Studies showed that it often results in large performance degradation if we try to eliminate all possible covert channels. For example, the RPcache [19] in our enviroment introduce more than 30% slowdowns in order to eliminate the cache miss based covert channels completely. Given that the attempt to send out large file through low capacity covert channel exhibits abnormal access patterns for excessive long duration, e.g., one year, it is relatively easy to detect such behavior with frameworks like [2].

In this paper, we are to enforce modest security demand, which is relaxed in two scenarios from the ideal design that has no covert channel at all.

- Section 2.2 has shown that a covert channel is often set up between a highly secure thread and a less secure thread, e.g., a hacker may need to hack a highly secure thread that can access sensitive data with a less secure thread that can freely access network. Based on this observation, we assume that co-running threads can be categorized at different security levels. Information flowing from low security level threads to high level ones is tolerable, but not vice verse.
- Threads may be accessing large files that are sensitive. Leaking such data takes a long time under low capacity covert channels. In this case, the security demand can be relaxed to restricting the capacity of potential covert channels, with a tradeoff for achieving better system performance.

Since the shift-based covert channels are introduced by non-fixed number of shift hops before each cache access, a naive mitigation solution is to enforce eager head update policy, i.e., to reset the head to a fixed position after each access. The latency to access a cacheline in one cache set only depends on its target location but not the preceding access, which eliminates the hidden channel between two threads. This policy, due to its large performance overhead, is adopted as a simple mitigation for comparison in this paper.

4.2 Security Level-Aware Approach

Our security level aware mitigation approach is proposed to restrict sensitive information from flowing from high security level threads to low level ones.

For each cacheline in the LLC, a two-bit flag L is attached (to the tag array) to indicates the security level of the thread that brings the line from memory. The two-bit security flag can differentiate four levels, which is sufficient as most systems need to protect one or two threads [19]. In addition, a 3-bit flag R is employed to implement 3-bit DRRIP [7] to track the recency information of each line. Two threads are placed at the same security level if they have shared data.

We revise the head update policy such that, after each access, the head is moved to a hot position that belongs to the thread with the lowest security level. We first look for a cacheline with L=0 and $R \neq$7 and, if not found, a line with L=0 and R=7. Here, large R values mark the least recent used items in the cache set. We continue to look for a line with L=1 if there is no line at L=0. The search continues until we find a target line. The search is performed during tag lookup. The head is moved based on the result of the search to the target domain on the track.

By always resetting the head to a cacheline at the lowest security level on track, consecutive accesses from this level exhibit minimal access latency difference, which prevents information leakage from the high levels to the lower levels. The only possible information leakage path is from the lowest level to high levels, which is tolerable as we discussed in Section 4.1.

4.3 Epoch-based Approach

Racetrack memory, due to its high density, enables the construction of large capacity LLCs that significantly reduce cache miss rates. Exploiting cache access locality, i.e., moving the head closer to hot cachelines, improves performance but creates covert channels. We next describe an epoch design that makes tradeoffs between performance degradation and covert channel capacity.

The scheme works as follows. We first divide the program execution to epochs, e.g., we keep a 64-bit counter and treat a duration of 50M processor cycles as an epoch. Within each epoch, we adopt eager head update policy such that the head is reset to a fixed position after each access. We choose the position to be the hot position of the track in the past epoch, which may be different from epoch to epoch. To identify this position, we use the 3-bit DRRIP tracking flag and choose the position with the smallest value. In this approach, we do not consider the security level of the thread such that the position may belong to different threads from epoch to epoch. In the case if there are multiple positions having the same value, we choose any one of them.

Figure 5 shows an example adopting the epoch based mitigation. The head is rest to cacheline `b` and `a` after each access in two consecutive epochs, respectively (Figure 5(a) and (b)). Within one epoch, no covert channel exists as two consecutive accesses cannot pass information due to head reset. However, a covert channel can be set up by monitoring the timing difference accessing `d` in two epochs, i.e., the sender fabricates its access pattern such that `b` and `a` become hot positions in two epochs. The fact whether accessing `d` has the same latency in two consecutive epochs passes one bit information. That is, there exists a covert channel with bit rate being one bit per epoch switch.

The epoch based mitigation approach is an approach with relaxed security. With the bit rate of the potential covert channel being proportional to epoch length, this approach provides a way to achieve better tradeoff between security demand and performance overhead.

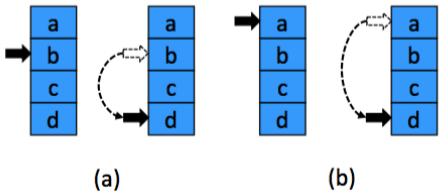

Figure 5: The covert channel in epoch based mitigation has low bit rate (i.e., one bit per epoch switch).

5. EXPERIMENTS

5.1 Methodology

To evaluate the effectiveness of our proposed mitigation schemes, we adopted the cycle-accurate simulator Gem5 [1] to simulate a CMP system with four 8-way OoO (out-of-order) cores and two levels of caches. The setting details are summarized in Table 1. We extended the simulator to simulate RM based L2 cache. We adopted the optimized cell structure in [25] such that each track has one *super* head, as shown in Section 2.1. The latency and energy parameters of racetrack memory are extracted from [24].

We used SPEC2006 benchmark suite for evaluation. We formed six workloads with benchmarks randomly chosen from SPEC2006 and mixed. Table 2 lists the mix details. The memory intensive benchmarks are highlighted using bold fonts.

We evaluated the following head update schemes and compared their effectiveness.

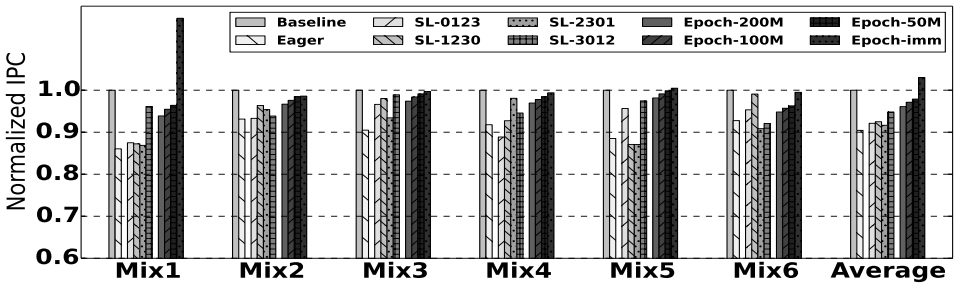

Figure 6: Comparing the IPC of different schemes.

Table 1: Simulation Configurations

Parameter	Value
Processor	Alpha ISA, 4 cores, 8-way OoO core
L1I L1D caches	4-way, 32KB, 2 cycles
L2 cache	16-way, 32MB, R/W/S: 24/24/4 cycles
Memory	DDR3 800MHz, tRAS=35ns, tRCD=13ns, tRP=13ns, tCL=13ns, tWR=15ns

Table 2: Workloads

Mix	Benchmarks			
	core 0	core 1	core 2	core 3
Mix1	perlbench	cactusADM	**povray**	**astar**
Mix2	**bzip2**	**gcc**	sjeng	sphinx3
Mix3	gamess	milc	hmmer	GemsFDTD
Mix4	gromacs	**gobmk**	**soplex**	**lbm**
Mix5	bwaves	zeusmp	**povray**	omnetpp
Mix6	gamess	zeusmp	cactusADM	**gobmk**

— `Baseline`: This scheme adopts the lazy policy in [15], i.e., it leaves the head above the last accessed item. It is vulnerable to covert channel attacks.

— `Eager`: This scheme adopts the eager policy in [15], i.e., it resets the head to the middle position after each access. It eliminates covert channel attacks but has large performance overhead.

— `SL-abcd`: This scheme implements our security level aware mitigation approach. It moves the head to a hot position of the lowest security level on track. Here, a, b, c, and d represents the security level order of the benchmarks running on four cores (Table 2). For example, Low-3012 for workload Mix6 indicates that the security levels are `gobmk` > `gamess` > `zeusmp` > `cactusADM`.

— `Epoch-x`: This scheme implements our epoch based mitigation approach. Here, x indicates the interval length. For example, For `Epoch-100M`, the head updates its hot position every 100 million cycles. Epoch-imm indicates the extreme case that updates the hot position and resets the head to the new position after each cache access.

5.2 Hardware Overhead

The proposed mitigation schemes introduce minimal hardware overheads. For the security level based mitigation, we need 5 bits per line, which is less than 1% of the storage overhead. In particular, the R flag is shared with DRRIP, the hardware LRU implementation. The epoch based mitigation requires one counter to determine the epoch boundary. The storage overhead is negligible. The searching logic for hot positions is integrated with the DRRIP circuits, which has less than 1% overhead based on our simulation. In summary, the hardware overhead of our proposed mitigation schemes is negligible.

5.3 Performance Results

Figure 6 compares the IPC of different schemes with the results normalized to `Baseline`. While `Eager` defends the shift-based covert channel attacks, it introduces 9.6% performance degradation. Figure 7 illustrates the IPC speedups for individual threads when their security levels decrease from level 0 to level 3 (the lowest).

For the security level aware mitigation, we compared four configurations with different benchmarks assigned at the lowest level. Given that cache accesses from this thread tend to be faster, i.e., the thread with the lowest security level gains the largest performance improvement. The more this thread is memory intensive, the larger overall performance improvement we can achieve. From the figure, we found that the improvement largely depends on the workload mix and varies from workload to workload. From the figure, in the case if the memory intensive benchmark is not assigned to the lowest level, e.g., bwaves in Low-1230 for Mix5, the overall performance is even worse than the `Eager`. In general, our security level aware mitigation approach achieves up to 10% performance improvement over the naive mitigation `Eager`.

For the epoch based mitigation, Figure 6 shows that the performance degradation from `Baseline` is linear to the length of the epoch — the larger the epoch is, the more likely we identify a staled hot position, the less access locality we can exploit from head update policy. In the extreme case that the head is always reset to the newly updated hot position, the figure shows that its improvements can be even better than `Baseline`. This is because such an aggressive head update policy exploits locality better and eliminates many pre-access shift potions. When epoch length is 100M, we achieved 7% performance improvement over the naive mitigation `Eager`.

5.4 Security Analysis

We then compared the security of different schemes by evaluating the bit rates of potential shift-based covert channels, which may still exist under performance and security level tradeoffs. Since one bit information may be leaked per epoch switch (Section 4.3), the security strength of an epoch configure is proportional to its length. As such, the bit rate of the covert channel can be computed as follows.

$$BitRate_{shift-based} = \frac{1}{T_{receiver} + T_{sender} + T_{interval}} \quad (6)$$

We found that the channel capacity is about 9.9bps if the epoch length is set at 200M. It takes years to transmit a video file with several hundreds of megabytes. This is comparable to the capacity of the cache miss based covert channel. Setting the epoch to be shorter gains better performance but becomes more vulnerable. The bit rates of the covert channels are 19.9bps and 39.3bps for epoch lengths at 100M and 50M, respectively.

6. RELATED WORK

The CMP systems that share hardware resources are vulnerable to covert channel attacks. Buses and interconnects are common resources of such channels [5][14]. [18] identified a new SMT/FU covert channel and a new speculation-based covert channel with reliability and capacity improved greatly.

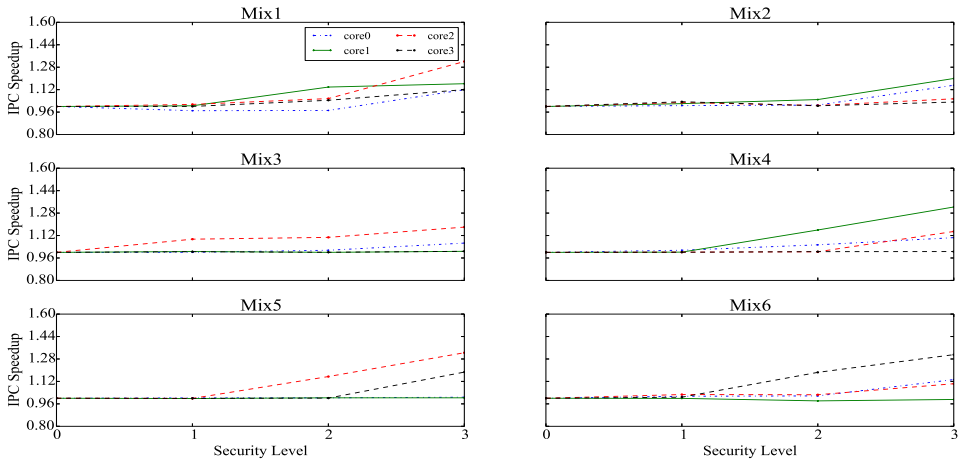

Figure 7: Comparing the sensitivity of assigning individual threads to different security levels.

[12] pioneered the studies of cache miss based covert channels. [13] investigated the LLC covert channels in cloud computing environment. [9] implementd LLC covert channel on native and virtualized environments, and made tradeoffs between bit rate and error rate. [22] showed that LLC covert channel capacity may be increased in cloud computing.

For mitigating cache based timing channels, [19] proposed a partition-locked cache to prevent critical data being evicted by other process, and a random permutation cache to shuffle set mapping whenever a process's data need to evict another process's data. Based on the access pattern of encoding algorithm, [6] proposed to bring in a random data into the cache instead of the required data. [20] proposed a larger logical direct mapped cache for better security and performance tradeoff. These schemes are orthogonal to our designs as shift-based covert channels can be initiated without introducing any cache misses.

7. CONCLUSIONS

In this paper, we elaborate the existence and severity of covert channels attacks in RM LLCs. Simple mitigation approach, while eliminating covert channels, tends to introduce large performance degradation. We then designed two mitigation approaches with tradeoffs among performance, security level, and channel capacity. Our experimental results show that the proposed schemes can reduce the newly discovered covert channel's information leakage rate by up to 260 times with modest performance overhead.

8. REFERENCES

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. In *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[2] J. Chen and G. Venkataramani Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *MICRO*. 2014.

[3] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *DAC*. 2008.

[4] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. Understanding contention-based channels and using them for defense. In *HPCA*. 2015.

[5] J. W. Gray III. On introducing noise into the bus-contention channel. In *IEEE Security and Privacy*. 1993.

[6] F. Liu and R. B. Lee. Random fill cache architecture. In *MICRO*. 2014.

[7] A. Jaleel, K.B. Theobald, S.C. Steely Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *ISCA*. 2010.

[8] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Impact of NBTI on SRAM Read Stability and Design for Reliability. In *ISQED*. 2006.

[9] C. Maurice, C. Neumann, O. Heen, and A. Francillon. C5: cross-cores cache covert channel. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2015.

[10] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of aes. In *Springer, Topics in Cryptology–CT-RSA*. 2006.

[11] S. Parkin, M. Hayashi, et al. Magnetic domain-wall racetrack memory. In *Science*, 2008.

[12] C. Percival. Cache missing for fun and profit, In *BSDCan*. 2005.

[13] 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 *CCS*. 2009.

[14] B. Saltaformaggio, D. Xu, and X. Zhang. Busmonitor: A hypervisor-based solution for memory bus covert channels. In *EuroSec*, 2013.

[15] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan. Tapecache: a high density, energy efficient cache based on domain wall memory. In *ISLPED*. 2012.

[16] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan. Stag: Spintronic-tape architecture for gpgpu cache hierarchies. In *ISCA*. 2014.

[17] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan. Dwm-tapestri-an energy efficient all-spin cache using domain wall shift based writes. In *DATE*. 2013.

[18] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *ACSAC*. 2006.

[19] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*. 2007.

[20] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *MICRO*. 2008.

[21] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *ISCA*. 2009.

[22] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *The 3rd ACM workshop on Cloud computing security*. 2011.

[23] Y. Yarom and K. Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *USENIX Security*. 2014.

[24] C. Zhang, G. Sun, W. Zhang, F. Mi, H. Li, and W. Zhao. Quantitative modeling of racetrack memory, a tradeoff among area, performance, and power. In *ASP-DAC*. 2015.

[25] X. Zhang, L. Zhao, Y. Zhang, and J. Yang. Exploit Common Source-Line to Construct Energy Efficient Domain Wall Memory based Caches. IN *ICCD*. 2015.