2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED) | 978-1-6654-8526-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEED55351.2022.00024

Protecting On-Chip Data Access Against Timing-Based Side-Channel Attacks on Multicores

Usman Ali, Abdul Rasheed Sahni, Omer Khan
{usman.ali, abdul.rasheed, khan}@uconn.edu
University of Connecticut, Storrs, CT, USA

***Abstract*—Shared hardware resources in multicore processors enable massive performance gains but bring security challenges. Adversaries have successfully leaked on-chip data using timing-based side-channel attacks (SCA) targeting shared caches, network-on-chip, and memory controller hardware. Existing mitigation schemes protect on-chip data against attacks targeting a single hardware resource, while leaving the additional channels unprotected. This paper implements multi-level mitigation schemes that protect on-chip data access against timing-based SCA targeting multiple shared hardware resources. It evaluates the performance implications of individual and multi-level mitigation schemes using security-critical graph and machine learning workloads. The multi-level mitigation schemes are shown to have ∼2% overall performance implication that primarily depends on the cache behavior of the workloads.**

I. Introduction

In today's multicore systems, on-chip data traverses various shared hardware resources to complete memory operations. This sharing of underlying hardware resources brings massive performance gains but introduces security challenges. The hardware sharing allows interference between otherwise virtually isolated processes, which adversary applications use to leak information. For example, adversaries have shown exploitation of Caches [1], [2], Network-on-Chip (NoC) [3], and Memory Controller hardware [4] using timing-based side-channel attacks. The literature proposes various mitigations schemes at software and hardware levels to ensure protection for on-chip data. For example, process-level isolation schemes at the software level (i.e., KASLR [5]) are adopted across co-executing processes to guarantee memory isolation, but it fails to protect against timing attacks targeting shared hardware. Contrary, hardware-level schemes protect individual shared hardware resources while leaving other channels vulnerable.

Existing mitigation schemes consider attacks on a discrete shared hardware resource as an independent problem. For example, randomization [6]–[8] and isolation schemes [9]–[11] protect shared caches against timing-attacks. Similarly, temporal isolation-based mitigation schemes, such as TDM [12] and SurfNOC [13] protect against timing attacks on network-on-chip shared routers and wires. Whereas, memory controller shared hardware queues are protected using temporal [4] or spatial [14] isolation schemes. However, a multi-level mitigation scheme that protects against multi-channel attacks is missing in the literature. What are the performance implications when two or more mitigation schemes are implemented simultaneously? Which combination of mitigation schemes is performance efficient? This paper aims to answer these questions by considering the data access path and its shared hardware resources holistically.

We consider multicore processors with on-chip data access that traverses the shared caches, NoC routers and wires, and memory controller hardware queues. The performance implications of protecting individual and multi-level hardware channels against timing-based side-channel attacks are evaluated using security-critical graph and machine learning workloads. We implement state-of-the-art randomization scheme [8] and partitioning scheme [10] [11] for cache protection. The literature shows that both schemes can be used for practical purposes, but it misses a direct performance comparison between them. Although both schemes require changes at the hardware level, the partitioning scheme is easier to implement with less hardware overhead but requires system support to tune cache partition sizes. Whereas, a randomization scheme requires more intrusive hardware changes, but it does not require system software support. The network-on-chip hardware is protected using a state-of-the-art, performance-efficient SurfNOC [13] scheme that temporally isolates shared NoC hardware to protect against timing attacks. Similarly, spatial and temporal [4] isolation schemes are implemented to protect the memory-controller hardware against timing-based side-channel attacks. The spatial partitioning scheme is shown to be performance efficient in literature [14]. Two combinations of multi-level mitigation schemes RSP (Randomization for cache, SurfNOC for NoC, and Spatial Partitioning for memory controller), and PSP (Partitioning for cache, SurfNOC for NoC, and Spatial Partitioning for memory controller) are derived based on the performance characterization of the individual channel mitigation schemes, and evaluated against each other.

The hardware intrusive schemes, such as SurfNOC, require hardware support that is not available in commercial off-the-shelf machines. Therefore, to study performance implications of mitigation schemes using realistic workloads, a detailed application-level simulator is needed. The

978-1-6654-8526-5/22/$31.00 ©2022 IEEE
DOI 10.1109/SEED55351.2022.00024

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

evaluated mitigation schemes are implemented on a RISC-V multicore simulator [15]–[17] that has been derived from the MIT Graphite [15] simulator with support for the RISC-V instruction set architecture. The hardware performance models are also updated in the simulator to study the performance implications of the timing-based side-channel mitigation schemes for a set of memory bound graph and machine learning workloads.

The evaluations show that the overall performance implications depend on the cache behavior of the workloads. Overall, both multi-level mitigation schemes have performance degradation of ∼2%, where RSP incurs a data access overhead of ∼8% and PSP ∼10%. Both schemes have different underlying reasons for performance degradation. The multi-level mitigation scheme with cache partitioning reduces the cache size, resulting in cache misses that inject increased traffic on NoC and the memory controller hardware. A costly mitigation scheme for NoC and memory-controller adversely affects the performance of such workloads. For example, for heavily memory bound workload, the cache partitioning scheme increases the NoC traffic by ∼40% and data access overhead of ∼10% compared to ∼5% for mixed traffic workloads. Contrary, a multi-level scheme with cache randomization disturbs the cache locality, resulting in unpredictable performance implications. The randomization penalty combined with performance implications of NoC mitigation decides the overall performance impact. For example, a workload with mixed on-chip and off-chip bound traffic shows data access overhead of ∼6% compared to ∼9% for on-chip memory traffic bound workload.

We conclude that multi-level mitigation schemes with cache randomization and partitioning have comparable performance implications. Cache randomization schemes have high hardware overhead and do not require operating system support. In comparison, the partitioning scheme incurs minimal hardware changes, but requires system support for auto tuning cache partition sizes.

II. Timing-based SCAs on Shared Channels

This section provides an overview of the data flow through shared hardware resources in a multicore processor. Further, this section discusses timing-based SCAs on individual and multi-level hardware channels, and threat model for multi-level mitigation schemes.

A. On-Chip Data Access

A multicore processor requires hardware sharing between multiple on-chip components to execute memory operations efficiently. Figure 1 shows an architecture of a 4x4 multicore processor consisting of 16 tiles. Each tile consists of a private core, private caches, shared cache slice, and network router. Certain tiles also include hardware for the memory controller. A shared cache is a fast memory accessible to all tiles in a multicore processor. NoC connects all tiles of a multicore processor to perform data accesses.

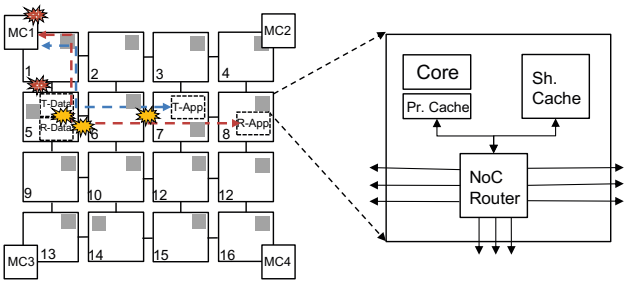

Fig. 1. A block diagram of multicore processor and two applications sharing on-chip hardware resources and point of contentions including 1) caches, 2) network-on-chip, and 3) memory-controllers.

The memory controllers enable access to off-chip main memory, and the retrieved data is moved on the NoC interconnect until it reaches its destination. An application data is homed at certain shared cache locations on-chip, and requires multiple hops on the NoC to reach the target tile. Similarly, shared cache misses require NoC to access the assigned memory controller to complete main memory accesses. For example, R-App (i.e., receiver application) in figure 1 uses a shared cache of tile 5, MC1 memory controller, and NoC hardware of tiles 1, 5, 6, and 7 with a T-App (i.e., transmitter application). This dependency on shared hardware causes timing variations that enable T-App to establish a covert-communication channel with R-App using timing-based SCAs. R-App can measure timing variations based on contention status on these shared hardware resources. Consequently, it uses the timing variations to infers secret data values.

B. Individual Channel SCAs

In timing-based SCA, an adversarial application performs software operations and measures the completion time of the operation under different states of the target shared hardware channel (i.e., cache, NoC, or the memory controller). The efficacy of attack depends on timing-variations, i.e., the timing difference to perform same operation under contended or non-contended access to the hardware channel. Modern processor enables precise timing-measurement using cycle accurate counters. In this paper, the *Tilera TileGx-72* multicore processor is utilized, where the get_cycle_count() api is used for timing measurements [18]. For a state-of-the-art cache attack [2], an adversarial application attempts to access a shared cache block. If the cache block exists in a shared cache, it results in a fast timing measurement due to cache hit access. The *Tilera TileGx-72* cache read hit operation takes ∼10-30 cycles, depending on the number of NoC hops between the tiles hosting the application code and the shared cache slice. On the contrary, if the cache block does not exist in the shared cache slice, a main memory request is generated. The cache miss operation involves cache tag check, memory controller requests, and data fetch from the main memory (i.e., DRAM). On *Tilera TileGx-72*, a cache miss takes ∼130-150

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

cycles to read a cache block from main memory. This ∼100 cycle timing-variation is sufficient for reliable timing-based SCAs to create a covert-communication channel.

Figure 1 shows an attack setup for covert-communication. It consists of a receiver application (R-App) and transmitter application (T-App). Transmitter application modifies the state of shared cache hardware, where receiver application measures the timing of cache hit versus miss using the cycle accurate counter. Similar to cache timing attacks, the transmitter application in [3] occupies the NoC resources (i.e., links, buffers, and crossbars), and creates timing-variations for the receiver application to measure. On the Tilera machine, a remote cache access involving non-contended NoC hardware takes ∼40-42 cycles. In case the NoC hardware is contended by an adversary application, the same operation takes ∼45-47 cycles. This timing variation of 5 to 7 cycles is sufficient for low efficacy timing-based SCAs but require additional optimizations (i.e., repetition codes [19]) for practical attacks. Similar to NoC hardware, memory controller shared hardware queues [4] are contended by the adversary application to create timing variations. A contended memory controller operation requires an additional 5 to 7 cycles compared to non-contended memory controller operations.

C. Multi-Channel SCA

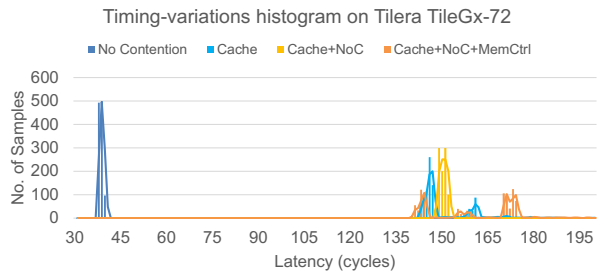

Fig. 2. Contention induced timing variations on shared hardwares.

A multi-channel timing-based SCA aggregates timing variations of two or more channels (i.e., cache, NoC, and memory controller), and thus have 1) high timing-variations compared to individual channels, and 2) shows timing variations for cases where mitigation schemes are in-place for certain shared hardware, but not all. Figure 1 shows two application codes and data placed such that they share the cache, NoC, and memory controller hardware. A multi-channel attack works similar to existing individual channel attacks [2] [3] [4]. In the multi-channel attack, a receiver application places data in a shared cache line (i.e., a variable), and let the transmitter application modify the state of the cache line. In the next step, the receiver application measures the latency to access the data. Suppose the transmitter application does not modify the state of the cache line. The receiver application finds the data in the cache line, and memory access involves cache hit and NoC accesses through tiles 5 and 7, resulting in a fast access latency. On the contrary, if the transmitter application flushes or evicts the cache line, the data access results in the main memory access through the memory controller. This operation involved cache (i.e., tag check), NoC accesses on tiles 1, 5, 6, and 7, and memory controller access on tile 1. The contention on these shared hardware channels results in a slow access latency observed by the receiver application. Although cache access (i.e., a cache miss) creates a significant timing variation by itself, the additional NoC and memory controller channels contribute to the timing variations. If the processor protects the cache hardware channel, the adversarial application can still use the aggregate timing variations due to NoC and memory controller channels to create a successful timing-based SCA. Figure 2 shows the timing variations histogram for cache channel, a combination of cache and NoC channels, and a multi-channel configuration consisting of cache, NoC and memory controller channels on the *Tilera TileGx-72* processor. A multi-channel contention aggregates the timing variations, and increases the efficacy of timing-based side-channel attacks.

D. Motivation for Multi-Level Mitigation

Existing mitigation schemes [8] [13] [4] protect against individual channel attacks while leaving remaining channels vulnerable for timing-based SCAs. For example, cache address randomization [8] or cache isolation-based mitigation schemes [11] eliminate timing variations from cache hardware. However, contention on unprotected NoC and/or memory controller is sufficient for timing-based SCAs. Similarly, NoC protection scheme [13] temporally isolates the NoC hardware, but shared memory controller queues can contend for timing variations. The improved efficacy of multi-channel attacks demand a holistic mitigation scheme that considers all sources of timing variations. This papers explores multi-level mitigation schemes, and evaluates their performance implications using security-critical graph and machine learning workloads.

E. Threat Model

The threat model is adopted from timing-based SCAs on shared hardware resources [1] [3] [4]. It is assumed that the adversary is capable of executing a co-located malicious application on the processor. The adversary also controls co-location of data on shared hardware (i.e., caches, NoC, and memory controller) since it has system level access. Additionally, the adversary is capable of measuring the timing information using cycle-accurate hardware counters. The threat vector only focuses on software-based timing SCAs and considers physical attacks such as power analysis, thermal monitoring, and electromagnetic attacks as orthogonal attack vectors.

III. Mitigation Schemes

The timing variations are essential to enable timing-based SCAs on shared hardware resources. Existing miti-

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

gation schemes attempt to eliminate timing variations on individual shared hardware channels, and are categorized into obfuscation and isolation-based schemes. Obfuscation schemes remove timing variations by introducing non-deterministic placement of data, and attempt to make constant time for critical security operations. Whereas, an isolation-based scheme partitions the critical security applications from non-critical applications. A cache randomization scheme, CEASER [8] introduced address randomization based on the principles of encryption/decryption for generic cache architectures. Other mitigation schemes, such as MIRAGE [20], ScatterCache [21], PhantomCache [22], CEASER-S [23], and CaSA [24] focus on specialized caches, such as skewed caches. Similarly, state-of-the-art isolation cache isolation schemes [10], [11] propose partitioning of caches for security and maximum performance.

The NoC and memory controller hardware are protected against timing attacks using isolation-based schemes. This section focuses on the 1) randomization and spatial isolation-based schemes [8], [10], [11] for Cache, 2) temporal isolation schemes i.e., TDM and SurfNOC for network-on-chip hardware, and 3) temporal and spatial partitioning schemes for the memory controller. These mitigation schemes have been shown in literature to provide strong security guarantees for individual shared hardware resources. However, these schemes leave the remaining channels vulnerable for multi-channel attacks. This section explains mitigation schemes for individual channels, and multi-level mitigation schemes.

A. Cache Mitigations

Shared cache is protected using a randomization or isolation scheme. Although, both schemes guarantee security, they have different implementation costs and execution requirements.

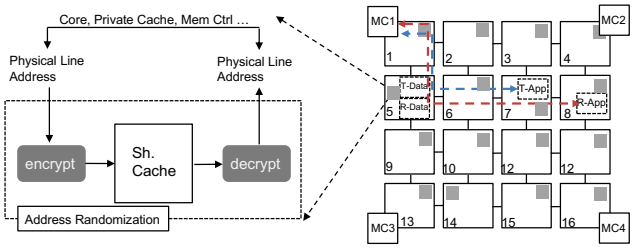

Fig. 3. A representation of randomization mitigation scheme in multicore system.

1) Randomization: A cache timing attack requires the identification of a unique cache line or set in cache memory, and the literature has shown such attacks. A randomization based mitigation scheme encrypts cache line addresses to randomize the placement of cache lines, thus mitigating the attack. Figure 3 shows the architecture of a randomization based mitigation scheme. It consists of an encryption hardware, and all cache operations are performed using the encrypted addresses. It also includes the decryption hardware to access outside data, i.e., write-back operation, and performs re-mapping. A single use of encryption/decryption hardware for address randomization incurs two additional cycles [8]. The randomization scheme is hardware intrusive and requires encryption/decryption hardware in all tiles. However, it is a hardware-only scheme and does not require system software support at runtime.

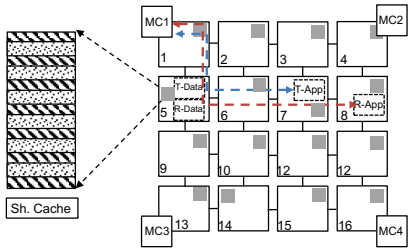

Fig. 4. A representation of cache partitioning scheme for two domains in a multicore system.

2) Partitioning: Spatial isolation scheme distribute the physical cache memory into clusters to create isolated domains for the co-located applications. Cache implementations are organized in sets and ways, and they can be either set-partitioned or way-partitioned. Figure 4 shows a 4x4 multicore setup with cache set-partitioning implemented on all tiles. The performance implications are dependent on the number of domains, and optimal size of the cache sets for each domain is depend on the workloads. The size of a set is computed either statically [9]–[11] or dynamically [25], and depends on the co-located workloads. Cache partitioning is relatively easier to implement by modifying each cache controller. However, it requires system software support to auto-tune set-partition size for each domain and workload.

B. Network-on-Chip Mitigations

Temporal isolation schemes guarantee security in NoC hardware.

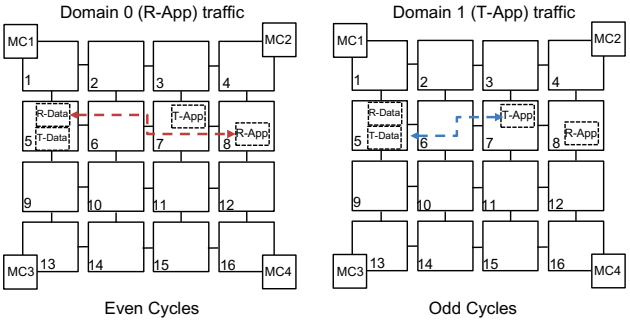

Fig. 5. TDM scheduling of NoC router in a 4x4 multicore system with two domains.

1) TDM: In time-division multiplexing (TDM) based temporal mitigation scheme, all NoC routers are allocated temporally to different domains at once. Figure 5 shows

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

temporal scheduling of NoC hardware allocation for two domains i.e., receiver and transmitter applications. The domain 0 application traffic uses NoC routers, queues, and links in odd cycles, whereas domain 1 traffic uses NoC hardware in even cycles. For example, Figure 5 shows an X-Y routing scheme where a domain 1 packet originating from tile 5 in an even cycle waits for an odd cycle to move to tile 7. For a one-cycle per-hop NoC hardware, the packet requires a total of 4 cycles to reach its destination compared to 2 cycles without TDM. This temporal isolation scheme eliminates the contention occurrence due to interference from other domain's application, thus guaranteeing protection against timing-based SCAs. However, the performance overheads in TDM are directly proportional to the number of domains and tiles. For example, every packet must wait for an additional cycle, proportional to tiles and domains to traverse through the target NoC routers.

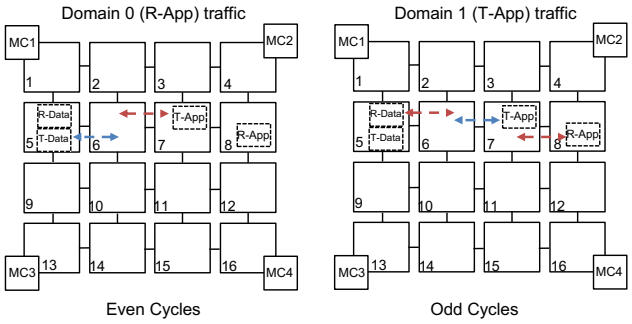

Fig. 6. SurfNoC scheduling of NoC router in a 4x4 multicore system with two domains.

2) SurfNOC: In SurfNOC [13], data from different domains flow like surfs. This wave-like packet traversal guarantees non-interference while reducing latency overhead compared to the TDM scheme. Figure 6 shows surf-like scheduling of two domains using SurfNOC. For example, the domain 0 data flows on red waves, whereas domain 1 data flows on blue waves. For an X-Y routing scheme, the figure 6 shows a domain 0 packet originating at tile 5 and having a destination at tile 8 observes a one cycle delay at start time while waiting for a red wave. The packet reaches its designation on cycle 4. The SurfNOC takes one cycle during traversal compared to two additional cycles of TDM (i.e, total 6 cycles for TDM). Each domain observes a delay directly proportional to the $number of domains - 1$ at start time, and once data rides the wave, there are no further delays. The only exception is for a higher number of domains (i.e., greater than 2) where an additional delay is possible at the turn while the packet changes the wave (i.e., dimension change). The target SCA attacks focus on two domains. Thus, SurfNoC reduces up to 75% performance implications compared to TDM, where the overhead depends on the number of tiles.

C. Memory Controller Mitigations

Memory controllers are protected using temporal or spatial isolation schemes.

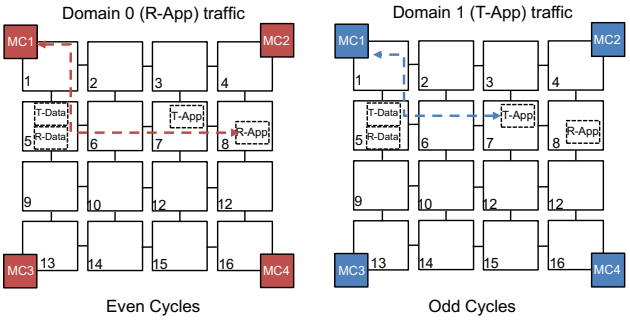

Fig. 7. TDM scheduling of Memory-Controller traffic in a 4x4 multicore system with two domains.

1) TDM: Similar to the temporal isolation of NoC hardware, all memory controllers are allocated to an individual domain at once. For example, Figure 7 shows a domain 0 workload utilize all available memory controllers in odd cycles, and domain 1 workloads use memory controllers in even cycles. This temporal allocation protects against timing attacks but is performance intrusive. The performance implications are directly dependent on the number of memory controllers and domains.

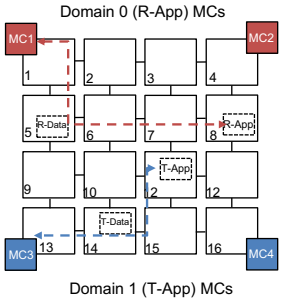

Fig. 8. A representation of memory-controller partitioning in a multicore system with two domains.

2) Spatial Partitioning: Spatial partitioning distributes memory controllers in two domains to guarantee security. Figure 8 shows a setup with four memory controllers, and two memory controllers are allocated per domain. For example, the MC1 and MC2 are allocated to domain 0 workloads, and MC3 and MC4 are allocated to domain 1. All data of a domain is stored in the corresponding main memory modules, and accessed using the assigned memory controllers. Compared to spatial partitioning of cache memories, spatial partitioning of memory controller is less sensitive to re-allocation of memory controllers. Thus, static allocation performs efficiently.

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

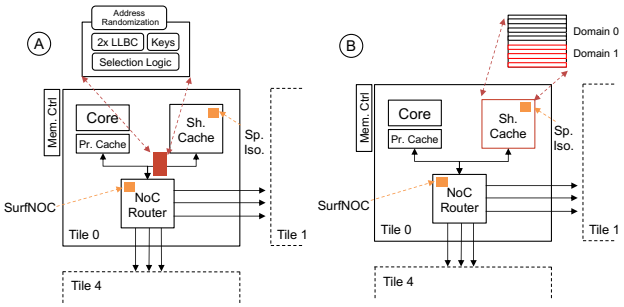

Fig. 9. Two combinations of multi-level mitigation scheme including SurfNOC for NoC, Spatial Partitioning for Memory Controller and Cache with a) Randomization based address obfuscation, and b) Spatial Partitioning of cache sets for two domains.

D. Multi-level Mitigations

Timing-based SCAs targeting shared cache, NoC, and memory controller can be protected using any combination of mitigation schemes. An ideal combination of mitigation schemes should have 1) minimum performance overhead for the workloads and 2) minimum software and hardware implementation cost. Cache randomization and spatial isolation schemes have comparable performance overheads but differ in implementation cost. Randomization requires additional new hardware and modification in the hardware of cache controllers, where isolation requires operating system support. Due to comparable performance and implementation costs of cache mitigation schemes, both schemes are considered. NoC mitigation schemes, TDM and SurfNOC have comparable implementation requirements, but SurfNOC outperforms TDM in terms of overall performance. On the other hand, memory controller temporal and spatial isolation schemes have comparable performance overheads. Temporal isolation requires hardware modification, whereas spatial isolation needs operating system support. Due to low numbers of memory controllers as compared to the number of cores in multicore processors, static spatial isolation eliminates the need for operating system support.

Based on these insights, we have selected two representative combinations that both utilize the best performing schemes for the NoC and memory controller but considered both options for cache protection. The two schemes are 1) RSP (Randomization for caches, SurfNOC for NoC hardware, and Spatial Partitioning for memory controllers) and 2) PSP (Partitioning for caches, SurfNOC for NoC hardware, and Spatial Partitioning for memory controllers). Figure 9 shows the required changes in each tile of the multicore to support the proposed multi-level mitigation schemes. RSP and PSP schemes differ in hardware intrusiveness and software support for implementation. For caches, randomization requires implementations of hardware intrusive encryption/decryption modules in each tile alongside key generation and control logic. The partitioning scheme requires modification in the cache controller to allocate limited addresses for the security-critical applications, where the remaining addresses are assigned to non-secure applications. Further, it requires system software support for allocation, re-allocation of cache sets, and calculation of the optimal size of the cache sets. The randomization scheme is oblivious to the system software. For SurfNoC, a modification is needed in the NoC router and crossbar to implement the surf schedular. For memory-controller spatial partitioning, the cache controller needs a modification to confine the memory address to allocated memory controllers. Section V evaluates the performance implications of these multi-level mitigation schemes. RSP and PSP schemes show comparable performance overhead, and any scheme can be used for practical purposes.

TABLE I
ARCHITECTURAL PARAMETERS FOR EVALUATION

Architectural Parameter	Simulator
Number of Cores	64 @ 1 GHz
Compute Pipeline per Core	Single–Issue
Memory Subsystem	
L1–I Cache per core	32 KB, 4–way Assoc.
L1–D Cache per core	32 KB, 4–way Assoc.
L2 Inclusive Cache per core	256 KB, 8–way Assoc.
Directory Protocol	Invalidation–based MESI
Num. of Mem Cntrl	4
DRAM Bandwidth per Cntrl	10 GBps
Electrical 2–D Mesh with XY Routing	
Hop Latency	2 cycles (1–router, 1–link)
Contention Model	Only link contention (Infinite input buffers)
Flit Width	64 bits

IV. METHODOLOGY

Hardware intrusive mitigation schemes (i.e., cache randomization and SurfNoC) require modifications in hardware, thus all schemes are modeled on a multicore simulator. We utilize an in-house RISC-V based large core count application level multicore simulator [17]. The simulator uses performance models from the MIT Graphite multicore simulator [15], and Architecture Description Language (ADL) [26] to describe the functional aspects of the RISC-V instruction set architecture, e.g., implementation of instructions, register states, MMU, and memory hierarchy models.

Table I configuration parameters are used in a 64 tiles multicore simulator. By default, each tile implements a single-issue in-order core pipeline with a 32 entry store buffer. However, to evaluate out-of-order cores, the evaluation is also performed with a core pipeline that implements 32-entry load and store queues, and a 128 entry reorder buffer and speculative execution. Each tile implements private level 1 (L1) instruction and data caches, $32KB$ each. A $16MB$ shared cache is physically distributed among tiles, where each tile implements a $256KB$ L2 cache slice. The tiles are kept cache coherence for their data accesses

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

using a directory-based hardware cache coherence protocol. The on-chip network implements a 2-D mesh topology using X–Y routing protocol.

A. Implementation of Mitigation Schemes

The simulator L2 cache models are modified to implement randomization, partitioning functionality and performance counters. A 40-bit encryption and decryption functions [8] are implemented to randomize cache addressing. All incoming traffic to L2 cache from Core, L1, and NoC router (includes memory controller) is encrypted before allocating and placing a cache line. The decryption is used for outward traffic including write-back of dirty cache lines and responses for synchronization purposes. The partitioning is implemented by reducing overall per tile L2 cache size, i.e., by half (for 50%) or quarter (for 25%) and assign it to a given domain.

The simulator NoC performance models are modified to include additional latency penalties for TDM and SurfNOC. In TDM implementation, the latency of routers doubles for the two domains setup at every NoC router. Contrary to TDM, the SurfNOC requires additional latency while waiting for surf and changing a surf (i.e., dimension shift). To model SurfNOC performance implications for two domains, the router latency is doubled at first NoC router, and later an additional router latency if an X to Y turn is detected. To model spatial isolation of memory controller on the simulator, L2 misses are routed to memory controllers mapped to their respective DRAM regions.

B. Benchmarks

The performance implications of the mitigation schemes are evaluated using six applications from two different security-critical domains. Three graph benchmarks, Single Source Shortest Path (SSSP), PageRank, and Triangle Counting (TC), are executed using a real-world California Road Network graph [27]. A machine learning workload AlexNet is also considered. ImageNet [28] is provided as an input to this benchmark. A sparse matrix-vector workload (SpMV), and a dense matrix-vector multiplication workload (GeMV) is ported from the PrIM benchmark suite [29]. SpMV and GeMV have extensive applications in graph and machine learning domains.

C. Evaluation Metrics

The workload performance is measured using the overall completion time and data access latency metrics. The overall completion time is reported using breakdowns of the time spent on performing operations in core (i.e., compute time) and time spent on data path (i.e., memory time). The data-access time is further broken down into time spent in L1 Cache, access between private L1 and shared L2 cache components (L1-L2 Home, L2Home-Waiting, L2Home-Sharers), and access between L2 cache and memory controllers (L2Home-Off-chip).

To measure the accuracy and reliability of mitigation schemes against timing-based SCA, the True Positive (TP) rate and Discrimination Index (DI) [30] metrics are used. The TP rate indicates the correct inference of contention or no-contention situation on the shared hardware. For example, if a transmitter application creates a contention situation on shared hardware, the receiver application successfully measures and infers a contention situation. We calculate the TP rate using $TP = \frac{d_c+d_n}{t_c+t_n}$, where d_c is total correctly classified contention cases, and d_n is the correctly classified no-contention case. The t_c is the total number of contention situations, and t_n is the total number of no-contention situations. The timing variations are plots for contention and no-contention situations, and TP is not sufficient to measure reliability of mitigations scheme. The DI metric quantifies the difference between two distributions of timing variations. DI uses the statistical mean and the variance of each distribution. We calculate the DI using $DI = \frac{\mu_c+\mu_n}{\sqrt{\sigma_c^2+\sigma_n^2}}$, where μ_c is the statistical mean of distribution in contention case, and μ_n is the statistical mean of distributed in no-contention case. The σ_c^2 and σ_n^2 are the variance in contention and no-contention cases, respectively.

V. Evaluation

Individual and multi-level mitigation schemes are evaluated. Initially, mitigation schemes are evaluated for individual hardware resources, and the efficient performance scheme for each hardware component is selected. The SurfNOC scheme for NoC hardware and spatial partitioning scheme for memory controller is found to be efficient. On the contrary, the efficiency of the randomization and cache partitioning scheme depends on benchmarks. We evaluate two multi-level mitigation scheme variants that both include SurfNOC for NoC hardware, Spatial Partitioning for memory controller, and randomization or partitioning of cache hardware. Finally, we show a security verification study of both cache schemes against conflict-based cache timing SCAs.

A. Cache mitigation performance implications

We evaluate randomization and the partitioning scheme to protect cache timing side-channel. Figure 10 shows the time spent on data access path of benchmarks without cache mitigation scheme, with two variants of partitioning scheme (i.e., 25% and 50% sets) and randomization mitigation scheme. The partitioning reduces cache size, thus causes an increase in cache misses, consequently increasing waiting time from the remote cache and off-chip memory. The partitioning has a negligible performance effect on benchmarks with minimum cache requirements (e.g., Tri-Count, PageRank), where the effect is visible on benchmarks with efficient utilization of shared cache (e.g., GeMV and AlexNET). On the other side, randomization has constant encryption/decryption overhead, and remapping affects the locality of cache placement due to address randomization. Randomization affects the performance of the L2 cache, and all workloads show an increased time for

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

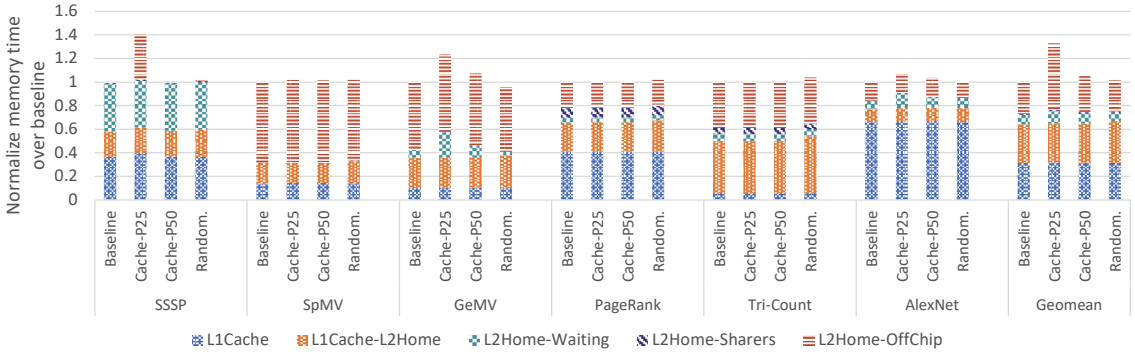

Fig. 10. Data access path performance implication of baseline, 25%, 50% cache randomization and partitioning

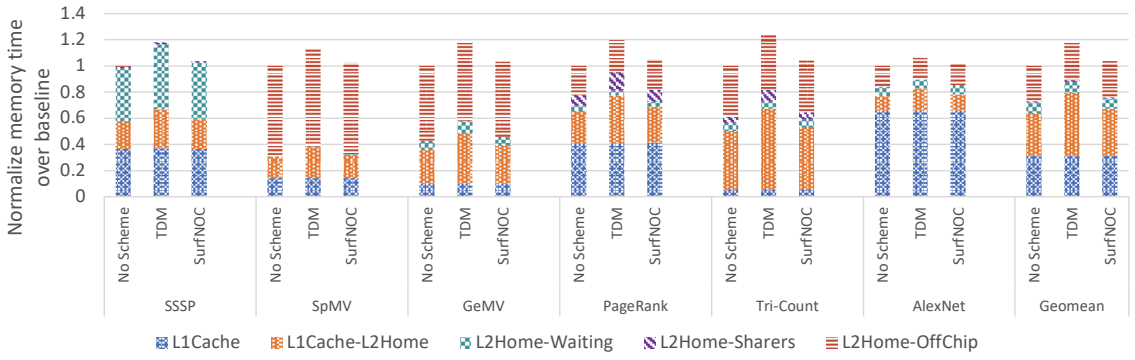

Fig. 11. Data access path performance implications of baseline, TDM and SurfNOC for network-on-chip

L2 home accesses. Partitioning shows batter performance for Tri-Count and PageRank, and randomization performs batter for remaining benchmarks.

B. NoC mitigation performance implications

The NoC performance implications are evaluated using two temporal isolation schemes. Figure 11 shows the time spent on data access path using the TDM and SurfNOC schemes. For a two domain threat model, the TDM scheme schedules one domain traffic in the even cycle and the other in the odd cycle. Thus, each domain traffic must wait for an extra cycle on every NoC router. It increases the 100% penalty on NoC router hardware. Figure 11 shows a visible increase in time spent on L2-Home time and L2-Sharers. Contrary, SurfNoC proposes an efficient performance scheduling for the NoC router. SurfNoC schedules traffic on NoC hardware in wave type pattern instead of scheduling traffic in even/odd cycle for two domains. This wave-like scheduling incurs delays at the start of the wave and on change of wave. For example, it results in a maximum two-cycle delay for any routing. SurfNoC shows a ∼3% increase in time spent on the data access path, compared to the TDM ∼18% increase. We propose to use SurfNOC in our multi-level mitigation schemes due to its lower performance overhead for all benchmarks.

C. Memory Controller mitigation performance implications

Figure 12 shows the time spent on the data access path with temporal mitigation (i.e., TDM) and three configurations of spatial partitioning mitigation scheme using 1, 2, and 3 memory controllers. The original configuration has 4 memory controllers. The memory controllers have high bandwidth, and the majority of benchmarks use 2 memory controllers to work efficiently. Figure 12 shows that all benchmarks perform efficiently with 2 memory controllers and have negligible overhead on the memory path of <1%. The TDM scheme schedules memory traffic in even/odd cycles for two domain configurations. This scheduling forces each domain traffic to wait for additional cycles while other domain uses the memory controller resources and consequently doubles the time spent on off-chip hardware. Figure 12 shows a 100% increase in off-chip time spent. Although selecting optimal number of memory-controller requires system support, we observe that 2 memory controllers are sufficient to fulfill all benchmark's requirements. Therefore, we pick 2 memory controllers per domain (i.e., 50% spatial partitioning) for the multi-level mitigation schemes.

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

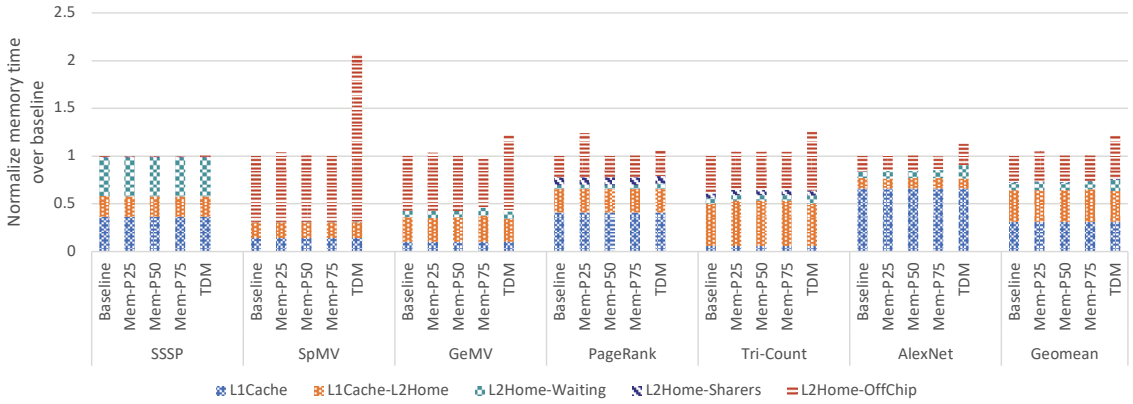

Fig. 12. Data access path performance implications of baseline, TDM and Spatial Partitioning for memory controller

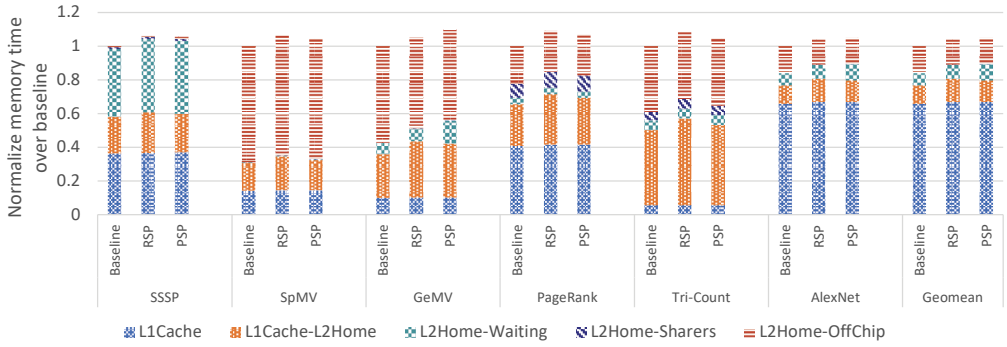

Fig. 13. Data access path performance implications of multi-level scheme with RSP (Randomization, SurfNOC, Spatial Partitioning) and PSP (Spatial Partitioning, SurfNOC, and Spatial Partitioning) for Cache, NoC, and Memory Controller.

D. Performance implications of multi-level mitigation

Multi-level mitigation protects on-chip data access against timing-based SCAs on multiple shared hardware resources. The performance of randomization and isolation-based mitigation schemes for shared cache depend on the benchmarks, as discussed in Section V-A. We use both cache schemes alongside SurfNOC for NoC hardware and 50% spatial partitioning of memory controllers. Figure 13 shows time spent on the data access path for RSP (i.e., Randomization, SurfNOC, and Spatial Partitioning) and PSP (Partitioning, SurfNOC, and Spatial Partitioning) multi-level mitigation schemes. Randomization incurs a fixed encryption/decryption overhead of 2 cycles, affecting the cache locality due to the randomization of the cache address. The encryption/decryption overhead increases time spent on the home L2 cache portion, and address randomization affects the time spent on L2-sharers (remote slices of shared L2 cache). Figure 13 shows that RSP performs worst in benchmarks including Tri-Count, and PageRank. These benchmarks involve considerable on-chip operations that involve L2 cache accesses. In PSP, the number of cache sets reduces to 25% or 50% based on the demand of the benchmarks. For example Tri-Count, and Page-Rank need minimum L2 cache to work efficiently, whereas SpMV, GeMV, SSSP, and AlexNet require a minimum of 50% cache sets to work optimally. The partitioning reduces the size of the slice of home L2, which triggers a trickle-down effect on other shared hardware. For example, L2-waiting time and L2-Sharers usage increases, which involves NoC hardware and memory controllers. For off-chip bound benchmark, GeMV, the time spent on data access path shows a ∼10% degradation compared to the geometric mean of ∼5%.

Although both RSP and PSP affect the time spent on the data access path, the overall completion times (i.e., time spent within the core, core synchronization, and memory path) are similar for all benchmarks. Figure 14 shows both RSP and PSP multi-level mitigation schemes degrade performance when each core implements an in-order pipeline. It shows a geometric mean performance overhead of ∼2%.

To evaluate the performance sensitivity of the SCA mitigation schemes for more powerful cores, each tile implements an out-of-order core. Figure 15 shows perfor-

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

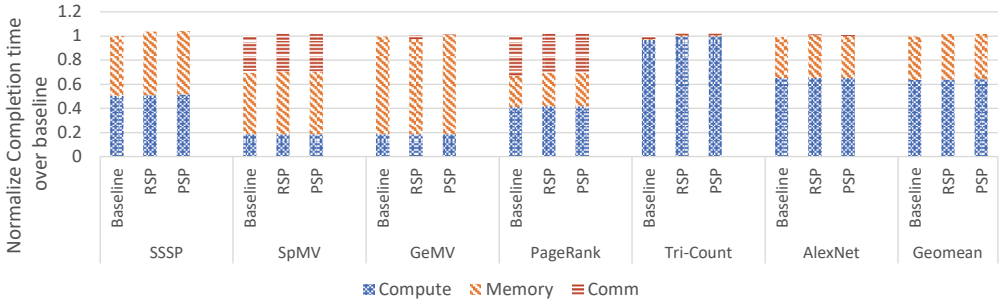

Fig. 14. Overall performance implications of multi-level scheme with RSP (Randomization, SurfNOC, Spatial Partitioning) and PSP (Partitioning, SurfNOC, and Spatial Partitioning) for Cache, NoC, and Memory Controller on an in-order processor

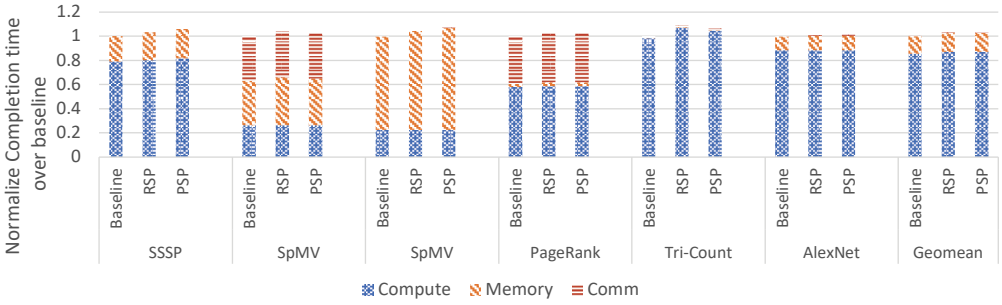

Fig. 15. Overall performance implications of multi-level scheme with RSP (Randomization, SurfNOC, Spatial Partitioning) and PSP (Partitioning, SurfNOC, and Spatial Partitioning) for Cache, NoC, and Memory Controller on an out-of-order processor

mance degradation of RSP and PSP multi-level mitigation schemes. Out-of-order cores reduce the overall memory stalls by exploiting instruction and memory level parallelism. It is evident that the time spent in computations increases as compared to memory stalls. However, the overall performance impact on the mitigation schemes is similar to in-order cores, resulting in a geometric mean performance overhead of ∼3%. Although both schemes i.e., RSP and PSP have the same performance overhead and guarantee security, they differ in hardware implementations and software support needs.

E. Security Analysis of Cache Mitigation Schemes

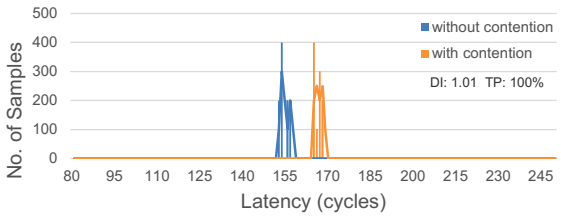

Fig. 16. Timing-variation histogram with Cache protection enabled

To validate the efficacy of our mitigation schemes, we implement a contention-based attack template with transmitter and receiver applications. As discussed earlier, timing variations observed by receiver application with and without contention on the multiple (cache, NoC and memory controller) shared hardware channels is sufficient for practical attacks. A positive DI of 1 and TP rate of 100% confirmed the efficacy of the attack vector. Figure 16 shows timing-variations of attack when cache partitioning is enabled to remove the timing variations from cache hardware. However, the NoC and memory controller are still used to create timing variations. The DI of 1 and TP rate of 100% confirm that an adversary is still successful with their attack.

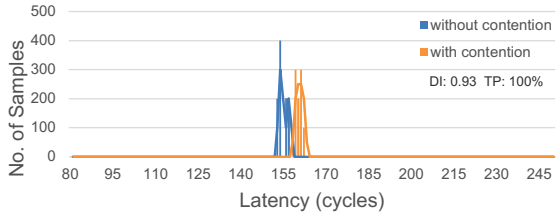

Fig. 17. Timing-variation histogram with Cache and NoC protection enabled

Figure 17 shows the timing-variation histogram with cache and NoC hardware protection schemes, but the

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

shared memory controller can still be exploited for timing attacks. A positive DI of 0.93 and TP rate of ∼100% confirms the inadequate mitigation. A multi-level mitigation scheme, i.e., RSP/PSP, completely removes timing variations. Figure 18 shows histogram with overlapped timing-variations. A negative DI and TP rate of ∼49% confirms successful mitigation of timing-based SCA with a multi-level mitigation scheme.

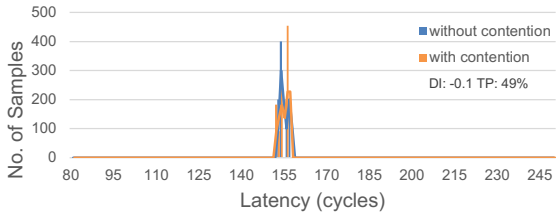

Fig. 18. Timing-variation histogram with multi-level protection enabled

VI. Related Work

A. Timing-based SCA

Cache hardware is extensively studied for timing-based SCA [1] [2] [31] [12] [3] [4] to leak security-sensitive information. Cache attacks such as Prime+Probe [1] and Flush+Reload [2] are shown to leak private keys of AES and RSA cryptosystems and are used alongside other vulnerabilities to develop newer attacks, including Spectre [32]. In the Prime+Probe attack, the adversary application occupies a cache set and lets the victim application evict a cache line from the occupied cache set. This victim-induced eviction forces adversaries to load data from primary memory, which takes extra time. An adversary can infer interference based on this extra time and creates a timing profile to leak secret information or develop practical attacks. A recent work, ConNOC [3] shows that the NoC hardware could be contented to develop practical covert-communication attack based on timing variations. In ConNOC attack, a receiver application and its data are pinned so that it shares NoC hardware with the victim application and its data. The transmitter and receiver applications are otherwise virtually isolated. The receiver application occupies all NoC hardware resources and starts monitoring packet arrival times. The transmitter application accesses its data, which affects the timing of receiver applications that create timing variations. Similarly, literature shows that memory controllers [4] [33] are exploited for timing-based SCA.

B. Mitigation Schemes

Timing-based SCAs are mitigated at the software level and hardware levels. Various techniques are proposed in the literature to detect the timing-based SCA that targets cache hardware at the software level. FlowTracker [34] uses an information flow tracking approach to analyze the code to detect timing attacks statically. The system software level schemes such as KASLR [5] are used to enforce software isolation to guarantee security against such attacks, but software-based mitigations failed to fix the core problem. The hardware/architecture level scheme removes timing variations by either making access constant time/randomization or enforcing hardware-level isolations. Cache randomization scheme RPCache [6] uses permutation tables to randomly moves cache set, and NewCache [7] randomizes cache lines and track randomized cache lines based on a Random Mapping Table. On the other hand, Ceaser [8] and randomization-based mitigation schemes [23] [20] [21] encrypt cache addresses using a pair of keys are performance efficient. The isolation schemes such as Intel CAT [9], DAWG [10], and IRONHIDE [11] protect cache hardware against timing attacks by enforcing strong isolation at cache set levels. This distribution of cache resources can be done statically [11] or dynamically [25]. NoC hardware is shown to be protected using either spatial isolation [12] or temporal isolation [13] schemes. Ed. Suh [12] proposed a one-way spatial isolation-based scheme that tags secure traffic as low priority traffic on NoC hardware with a fixed bandwidth quota. This approach removes interference at NoC but limits the usage of NoC hardware. Later SurfNOC [13] propose a new scheduling scheme that enforces temporal isolation but minimum performance implications. The memory-controller hardware is shown to be protected using either temporal isolation [4] scheme or spatial isolation [11] [14] schemes. TDM schedules allocation of resources to secure traffic on even cycles and non-secure applications on odd cycles. Contrary, the spatial isolation schemes allocate a fixed amount of memory controllers to secure and non-secure applications.

VII. Conclusion

Timing-based SCAs leak on-chip data that traverses various shared hardware resources in multicore processors. Existing software level mitigation schemes provide inadequate security, whereas hardware schemes protect on-chip data for individual shared hardware channels. This paper evaluates existing hardware-level mitigation for each hardware resource on the on-chip data access path, and implements two combinations of multi-level mitigation schemes (i.e., RSP and PSP) that provide holistic protection against timing-based SCAs. The performance implications of the multi-level mitigation schemes depend on the cache behavior of workloads. Our evaluation shows that multi-level mitigation schemes incur performance overheads of ∼2% for real-world graph and machine learning workloads. Although both schemes protect against timing side-channel attacks and have similar performance implications, they differ in hardware implementation cost. RSP scheme has high hardware overhead but is oblivious to the system software, whereas the PSP scheme has low hardware overhead but requires the system to implement auto-tuning for set-partitioning of the shared cache.

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.

VIII. Acknowledgments

This research was supported by the National Science Foundation under Grant No. CNS-1929261. This research was also supported in part by the Semiconductor Research Corporation (SRC).

References

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Topics in Cryptology – CT-RSA 2006* (D. Pointcheval, ed.), (Berlin, Heidelberg), pp. 1–20, Springer Berlin Heidelberg, 2006.

[2] 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)*, (San Diego, CA), pp. 719–732, USENIX Association, Aug. 2014.

[3] U. Ali and O. Khan, "ConNOC: A practical timing channel attack on network-on-chip hardware in a multicore processor," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021.

[4] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 225–236, 2014.

[5] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, (New York, NY, USA), p. 481–493, Association for Computing Machinery, 2020.

[6] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, (New York, NY, USA), p. 494–505, Association for Computing Machinery, 2007.

[7] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pp. 83–93, 2008.

[8] 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)*, pp. 775–787, 2018.

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

[10] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, p. 974–987, IEEE Press, 2018.

[11] H. Omar and O. Khan, "Ironhide: A secure multicore that efficiently mitigates microarchitecture state attacks for interactive applications," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 111–122, 2020.

[12] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pp. 142–151, 2012.

[13] H. Wassel, Y. Gao, J. Oberg, T. Huffmire, R. Kastner, F. Chong, and T. Sherwood, "Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip," in *ISCA*, 2013.

[14] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding information leakage in the memory controller with fixed service policies," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 89–101, 2015.

[15] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–12, 2010.

[16] H. Dogan, M. Ahmad, B. Kahne, and O. Khan, "Accelerating synchronization using moving compute to data model at 1,000-core multicore scale," *ACM Trans. Archit. Code Optim.*, vol. 16, pp. 4:1–4:27, Feb. 2019.

[17] "QUARQ: A Novel General Purpose Multicore Architecture for Cognitive Computing." https://khan.engr.uconn.edu/pubs/quarq-techcon17.pdf, 2017.

[18] T. Corporation, "Tile-gx72 processor," 2014.

[19] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," *IEEE Micro*, vol. 40, no. 3, pp. 91–98, 2020.

[20] 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)*, pp. 1379–1396, USENIX Association, Aug. 2021.

[21] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 675–692, USENIX Association, Aug. 2019.

[22] Q. Tan, Z. Zeng, K. Bu, and K. Ren, "Phantomcache: Obfuscating cache conflicts with localized randomization.," in *NDSS*, 2020.

[23] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 360–371, 2019.

[24] T. Bourgeat, J. Drean, Y. Yang, L. Tsai, J. Emer, and M. Yan, "Casa: End-to-end quantitative security analysis of randomly mapped caches," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1110–1123, 2020.

[25] H. Omar, B. D'Agostino, and O. Khan, "Optimus: A security-centric dynamic hardware partitioning scheme for processors that prevent microarchitecture state attacks," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1558–1570, 2020.

[26] "FreescaleADL: An Industrial-Strength Architectural Description Language For Programmable Cores." http://opensource.freescale.com/fsl-oss-projects/, 2013.

[27] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *IISWC*, 2015.

[28] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE CVPR*, 2009.

[29] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture," 2021.

[30] Y. Massoud, J. Kawa, D. MacMillen, and J. White, "Modeling and analysis of differential signaling for minimizing inductive cross-talk," in *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, (New York, NY, USA), p. 804–809, Association for Computing Machinery, 2001.

[31] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, DIMVA 2016, (Berlin, Heidelberg), p. 279–299, Springer-Verlag, 2016.

[32] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[33] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-cpu attacks," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 565–581, USENIX Association, Aug. 2016.

[34] B. Rodrigues, F. M. Quintão Pereira, and D. F. Aranha, "Sparse representation of implicit flows with applications to side-channel detection," in *Proceedings of the 25th International Conference on Compiler Construction*, CC 2016, (New York, NY, USA), p. 110–120, Association for Computing Machinery, 2016.

Authorized licensed use limited to: UNIVERSITY OF CONNECTICUT. Downloaded on July 06,2023 at 15:19:15 UTC from IEEE Xplore. Restrictions apply.