ASM: An Adaptive Secure Multicore for Co-located Mutually Distrusting Processes

ABDUL RASHEED SAHNI, HAMZA OMAR, USMAN ALI, and OMER KHAN, Universty of Connecticut

With the ever-increasing virtualization of software and hardware, the privacy of user-sensitive data is a fundamental concern in computation outsourcing. Secure processors enable a trusted execution environment to guarantee security properties based on the principles of isolation, sealing, and integrity. However, the shared hardware resources within the microarchitecture are increasingly being used by co-located adversarial software to create timing-based side-channel attacks. State-of-the-art secure processors implement the *strong isolation* primitive to enable non-interference for shared hardware, but suffer from frequent state purging and resource utilization overheads, leading to degraded performance. This paper proposes ASM, an adaptive secure multicore architecture that enables a reconfigurable, yet strongly isolated execution environment. For outsourced security-critical processes, the proposed security kernel and hardware extensions allow either a given process to execute using all available cores, or co-execute multiple processes on strongly isolated clusters of cores. This spatio-temporal execution environment is configured based on resource demands of processes, such that the secure processor mitigates state purging overheads and maximizes hardware resource utilization.

CCS Concepts: • **Computer systems organization** → **Multicore architectures**; • **Security and privacy** → **Security in hardware**.

Additional Key Words and Phrases: secure architecture, hardware security, strong isolation

1 INTRODUCTION

In the era of cloud, data-center, and distributed computing, the privacy of user's code and data in computation outsourcing is a challenge. Today's microprocessors enable aggressive hardware virtualization by means of which multiple clients (users) are provided with separate virtual spaces that temporally co-execute on the processor. Data encryption, integrity, and attestation mechanisms are being adopted in commercial processors to provide users with a secure computation outsourcing experience. However, these mechanisms do not provide sufficient protection against side-channel attacks, since concurrently executing processes share hardware resources, such as on-chip caches, translation look-aside buffers (TLBs), on-chip network and off-chip memory. This resource sharing induces hardware interference that is exploited by an adversarial process to infer confidential information of a given process by monitoring the microarchitecture state via timing-based side channels [11, 26, 28, 30, 36].

Corresponding Author: Omer Khan.
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).
Authors' address: Abdul Rasheed Sahni, abdul.rasheed@uconn.edu; Hamza Omar, hamza.omar@uconn.edu; Usman Ali, usman.ali@uconn.edu; Omer Khan, omer.khan@uconn.edu, Universty of Connecticut, 371 Fairfield Way, Storrs, Connecticut, 06269 .

XXXX-XXXX/2023/3-ART $15.00
https://doi.org/10.1145/3587480

Secure processor technology is an active research area that aims to tackle the threats posed by software side-channel information leakage. The non-enclave based schemes either protect a given process' state via scrambled (randomly mapped) address accesses [29, 39], or introduce intrusive hardware extensions [40, 48, 50] to disallow a process from accessing unaffiliated data. On the other hand, enclave-based schemes [2, 9, 12, 31, 33, 38] aim to develop non-interference containers that are isolated at hardware-level from co-executing processes. The state-of-the-art MI6 secure processor [12] deploys enclave-based model to ensure strong isolation. It implements a trusted security kernel that statically and equally partitions the large state structures, such as the off-chip memory regions across the processes. However, all other microarchitecture state is temporally shared, and per-core cache and buffer resources are *purged* on each enclave entry and exit. Consequently, MI6 experiences significant performance overheads due to frequent purge and reload of private data, and imbalanced static allocations of large state structures. MI6-Optimus [35] dynamically partitions the last-level cache to improve the shared cache utilization. Although processes with high core-level parallelism benefit from the available multicore resources, processes with low resource demands suffer from under-utilization. Moreover, all processes incur frequent purge overheads on each enclave entry/exit.

The Ironhide [38] secure processor avoids purging overheads by creating two strongly isolated clusters of cores. It enables execution of secure and insecure processes in two strong isolated clusters of cores. This allows the secure process to interact with system software without incurring costly enclave exits. In this execution model, processes with high core-level parallelism suffer from performance degradation due to the distribution of cores into two clusters. However, the secure process achieves high throughput by avoiding the purging overheads.

The objective of this paper is to enable a secure processor technology that takes advantage of temporal execution of processes that demand high core-level resources (similar to MI6-Optimus), yet dynamically enable spatial execution (similar to clusters of cores in Ironhide) for co-location of processes with low resource demand. We propose adaptive secure multicore ASM that aims to reduce unwanted state purges between execution of co-located processes, while maximizing core-level parallelism. ASM considers a baseline multicore processor with many cores interconnected using on-chip networks and a multi-level cache hierarchy. It enables execution of processes using dynamically configurable *single* and *multiple cluster* modes.

The *single cluster* mode executes processes in a time-multiplexed manner. However, instead of partitioning the last-level cache among co-located processes, like MI6-Optimus, it implements a parallel purge operation for all shared cache slices. Even though this flush operation is costly, the benefits arise from improved data locality during process execution. In *multiple cluster* mode, ASM dynamically creates strongly isolated clusters of cores by utilizing hardware based reconfiguration capabilities. Each cluster is allocated a set of cores that utilize a hardware based address map table to redirect local cache accesses to those shared cache slices that are within the cluster boundary. This ensures no last-level cache slices get shared among the clusters. The on-chip network implements a deterministic routing protocol to ensure that network packets never cross the cluster boundary. The shared cache misses also utilize the hardware-level address map table to map the respective memory region accesses to memory controllers that are within the cluster boundary. The DRAM modules are statically isolated across clusters and dedicated memory controllers are utilized to enable strong isolation. The *multiple cluster* mode is beneficial for executing co-located processes when their aggregated thread-level parallelism matches the available core counts in the system. To ensure strong isolation during mode transition from single-to-multiple or vice versa, ASM flushes all shared hardware resources.

ASM extends the security kernel in the secure processor to include *process-to-mode* mapping software for the co-located processes. The idea is to provide each process with an execution mode that matches its

core-level resource demands. The process-to-mode mapping is communicated to the operating system for mode aware scheduling of processes. The processor hardware independently accesses the mappings for the scheduled processes from the security kernel. If the operating system violates the mappings determined by the security kernel, the hardware raises an exception. Otherwise, the hardware executes the scheduled processes while ensuring strong isolation among them.

The contributions of this work are summarized below:

- ASM proposes an adaptive secure multicore processor that switches between temporal and spatial execution modes based on process demands. The temporal execution mode is beneficial for applications with high parallelism, whereas spatial execution improves performance for less resource intensive applications by executing them in parallel.
- ASM introduces process-to-mode mapping in security kernel to map processes to appropriate execution mode (single or multiple clusters).
- Conduct security analysis of ASM to verify that execution of co-located processes under ASM single or multiple cluster mode is secure.
- Conduct scalability study of ASM using large core count multicore processors.

ASM is prototyped on a real *Tilera® Tile-Gx72™* multicore processor, as well as a RISC-V ISA based multicore simulator. When utilizing the 64-cores in the Tilera processor, ASM with single- and 2-cluster modes is shown to outperform MI6-Optimus by ∼49%, and Ironhide by ∼11%. However, the performance benefits diminish when *3-cluster* mode is also introduced in the 64-cores processor. To evaluate ASM for larger core count processors, the multicore simulator is configured to simulate up to 256 cores. At 128 core configuration, ASM with 4-cluster mode gives ∼20% improvement over the 2-cluster mode. This performance scalability of ASM increases to ∼30% for 6-cluster mode on 256 cores configuration.

2 THREAT MODEL

This work primarily focuses on enclave-based secure processors due to their continuing commercial integration and high-end security guarantees [2, 9, 12, 31, 33, 38]. Hence, the threat model assumptions are adopted from state-of-the-art MI6 [12] secure processor, where all microarchitecture state attacks that rely on covert and information leakage side-channels are considered. Specifically, a virtualized environment is considered, where security-critical processes belonging to different users are considered as mutually distrusting. The operating system (OS) is assumed to be untrusted as well. However, the processor package (including main memory) and a security kernel are trusted.

Adversarial Capabilities: The adversarial process can co-locate with a victim process on the shared microarchitecture structures, and conduct timing-based side channel attacks. The adversary is capable of exploiting both non-transient state (e.g., caches [11, 21, 30, 39]), as well as transient state (e.g., on-chip network routers [44] and speculative execution units [14, 26, 28]). Moreover, the adversary can monitor resource scheduling events to infer information via the timing and termination channels [19, 23, 35]. The main focus is on the software-based side-channel attacks on the microarchitecture state. Thus, adversaries with physical access to conduct attacks via thermal imaging, electromagnetic, and power based channels are not considered in this work. Moreover, physical attacks on memory are also not considered as these exploits are prevented using orthogonal mechanisms, such as memory integrity checking [20] and ORAM [37]. Attacks by compromised system software, e.g., OS refusing to allocate process resources are not considered. Lastly, hardware attacks, such as hardware trojans [10] and rowhammer [24] are considered orthogonal attack vectors.

3 BACKGROUND AND RELATED WORK

Industry has commercialized efforts to reduce the trusted computing base (TCB) to a secure processor chip [2, 9, 31, 33]. Intel's SGX [13, 33] platform maintains on-chip enclaves, which isolate processes from the untrusted OS via key management, and address partitioning. However, Intel's SGX and its successor TDX [5] have been shown to be vulnerable against cache-timing and control flow speculation attacks [14, 21]. Recent secure processor works [12, 25, 38] extend the idea of enclaves in SGX to alleviate microarchitecture state attacks. For instance, DAWG [25] utilizes protection (or security) domains to isolate secure data from malicious insecure applications.

3.1 The MI6 Secure Processor

The MI6 secure processor [12] adopts Intel's SGX enclave execution model, and implements the *strong isolation* [18] security primitive to protect against microarchitecture state attacks. The clients' mutually distrusting applications comprising of security-critical and ordinary (insecure) processes are offloaded to their respective processing nodes; where all processes execute in a time-sliced fashion and temporally share hardware resources. MI6 implements strong isolation by: (a) purging the small stateful hardware structures, such as core pipeline buffers, private caches and TLBs, and memory controller queues on each secure process context switch (or enclave entry and exit), as shown in figure 1:② and (b) statically partitioning the large stateful resources, such as shared last-level cache, shared TLBs, and off-chip memory (DRAM regions) across the processes, shown as ③ in figure 1. In MI6, from a stream of incoming mutually distrusting processes, each context switch (enclave entry/exit) requires purging of the private micro-architecture state, leading to performance overheads. These overheads further aggravate as the purged state also needs to be brought back into the hardware caches/TLBs. Additionally, the static partitioning of last-level cache resources adversely impacts performance, as each process is allocated with fixed shared cache set(s)/slice(s). Thus, any process in need of more cache capacity is expected to experience degraded data locality. If multiple secure processes are scheduled for execution, it is detrimental for performance to statically partition the shared cache slices to ensure strong isolation.

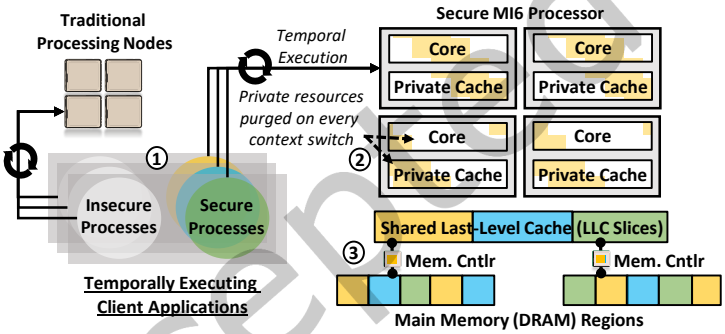


Fig. 1. The execution model for secure processes on MI6.

The *MI6-Optimus* [35] dynamically distributes the shared cache slices among processes while ensuring strong isolation. It extends the security kernel to sample MPKI (misses per kilo-instructions) for each process to decide how many shared cache resources should be allocated for near-optimal performance. Consequently, it deploys a deterministic and bounded allocation of shared cache among secure processes. By matching the cache capacity demands of processes, it improves performance over MI6. However, as the number of co-located processes increases, the dynamic and proportional cache distribution still remains a performance bottleneck. In addition, the temporal execution of processes in MI6 and MI6-Optimus incur frequent purging overheads.

3.2 The IRONHIDE Secure Processor

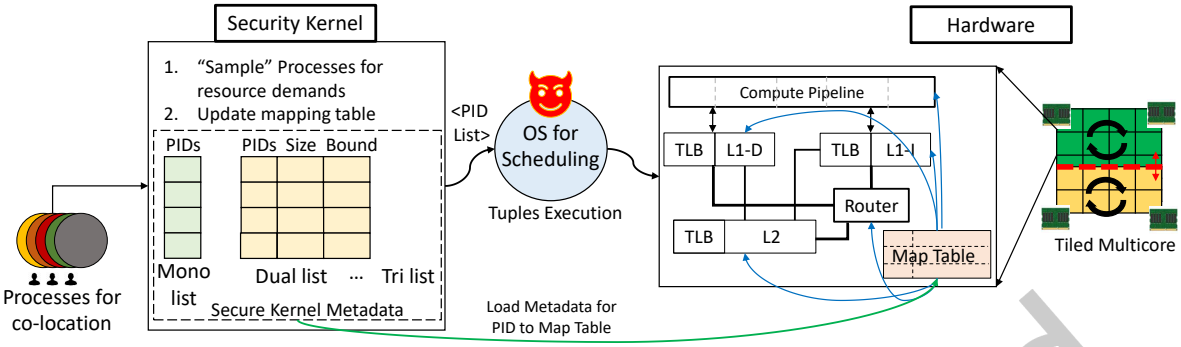


Fig. 3. ASM architecture interactions between hardware, security kernel and the operating system for a set of co-located processes.

The Ironhide [38] secure processor enables strong isolation by spatially distributing core-level resources in a shared memory multicore to support two clusters of cores. Each cluster is provided with spatially partitioned cores, private and shared caches, TLBs, network-on-chip, memory controller, and DRAM regions. For co-located processes, Ironhide pins each process threads to their allocated cluster's cores. For load balanced execution, it implements a dynamic resource distribution scheme that reconfigures the size of each cluster while ensuring strong isolation. To ensure isolation, Ironhide purges the core pipeline and private and shared caches, shown as ① in figure 2. When co-located processes exceed the number of supported clusters, Ironhide adopts temporal execution among clusters while adjusting the size of cluster as per incoming process demands. On such scheduling event, it purges the core level resources (both private and shared) and reconfigures the cluster size for incoming processes, as shown in figure 2. This spatial co-location of processes improves performance over MI6/ MI6-Optimus in two ways, (1) the overall number of state purges is reduced due to spatial execution of processes, and (2) the core-level parallelism is improved due to better hardware resource utilization. However, spatial execution leads to performance degradation in processes that demand higher core-level parallelism.

This paper makes a key observation that processor resources can be better utilized if a method is enabled that dynamically and securely utilizes the temporal execution model (MI6-Optimus) for processes with high core-level parallelism, and the spatial execution model (Ironhide) for low resource demanding processes. We propose ASM, an adaptive secure multicore processor that offers mutually distrusting secure processes with a dynamic execution model with improved resource utilization, while satisfying strong isolation guarantees.

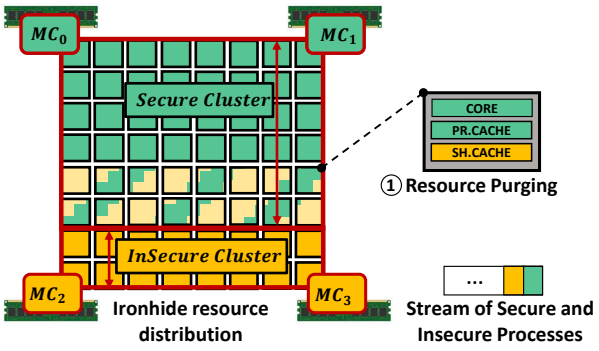


Fig. 2. The execution model of Ironhide.

4 THE ADAPTIVE SECURE MULTICORE (ASM)

The proposed ASM secure processor is illustrated using a representative shared memory tiled multicore consisting of numerous cores (in-order or out-of-order with speculative execution support). The tiles are interconnected using on-chip network routers, a per-core private–shared cache hierarchy including

translation look-aside buffers (TLBs), and multiple memory controllers connected to distributed memory channel modules (DIMMs). When a core makes local access to its private L1-cache, it results in a cache and TLB hit or miss. The L1 cache and TLB misses are routed (using the on-chip network) to the shared last-level cache and TLB based on a *homing policy* determined using the address mapping. The last-level cache and TLB are physically distributed, and a TLB slice as well as a L2-cache slice with an integrated directory for cache coherence is implemented in each tile. The on-chip network routes the miss request to a dedicated L2 or TLB that results in a hit or miss. A hit retrieves the cache line and returns it to the requesting core, while a miss request is forwarded to the memory controller that maps the respective DRAM region. The L2 misses are also routed based on a *homing policy* determined using the address mapping. The cache line is retrieved from the DRAM and returned to the requesting L2 cache slice.

During execution of a multi-threaded process, the private resources of a core i.e. compute pipeline, private caches and TLBs are temporally shared with other processes. Moreover, the on-chip network routers and wires, as well as memory controller queues are shared as well. To prevent the microarchitecture state attacks, strong isolation is needed among the co-located mutually distrusting processes. The strong isolation requires all shared hardware resources to ensure non-interference. The MI6 and MI6-Optimus achieve temporal strong isolation by purging/flushing all shared hardware, except for large stateful resources (last-level cache and DRAM) that are partitioned among the co-located processes. On the other hand, Ironhide achieves spatial isolation by creating strong isolated clusters of cores to co-execute multiple processes concurrently. When the number of co-located processes exceeds the number of clusters, Ironhide resorts to purging/flushing of all shared hardware, except the partitioned DRAM regions.

ASM achieves strong isolation by utilizing both temporal and spatial execution of the co-located processes. The temporal execution mode (termed as single cluster) allocates all hardware resources (except DRAM regions) to a process, instead of partitioning the shared resources like in MI6-Optimus. The single cluster mode ensures strong isolation by performing flush/purge on each process invocation. The spatial execution mode reconfigures the processor into multiple clusters of cores for concurrent execution of multiple processes, and performs flush/purge to ensure strong isolation on each invocation of processes for co-execution. The hardware supports secure transitions between the temporal and spatial execution modes, as well as purging/flushing logic to ensure strong isolation. The spatial execution mode may implement 2 or more clusters of cores (termed as multiple clusters).

The ASM execution model envisioned in Figure 3 comprises of co-located processes that are first intercepted by the security kernel. The security kernel is extended with a process-to-execution mode mapping capability that creates per-process metadata for efficient yet secure execution. It first samples each process to determine its core-level resource demands. Next, the mapping phase utilizes the sampled information to map each process for execution under the single- or multi-mode execution. It iteratively allocates processes to single or multi-cluster list data structures. For example, Figure 3 shows dual-list in a configuration that supports single- and 2-cluster mode execution under ASM. The security kernel is part of the TCB; hence the process-to-execution mode mappings are protected from adversarial attacks on ASM.

The security kernel passes the single and multiple-cluster lists to the operating system (OS) for scheduling of processes on the processor. Although the OS is outside the TCB, the sharing of this metadata does not reveal any private information from one process to another. These lists only reveal the execution mode for each process, and the OS is allowed the freedom to schedule their execution as per the system level constraints, such as throughput, fairness and quality-of-service. When the OS schedules a particular mode of execution for a given process or tuple of processes, the hardware loads the metadata into a per-core map table by directly accessing it from the security kernel. In case of a compromised OS, the malicious process may modify the metadata in the OS. This inconsistency in metadata between

the OS and security kernel's data structures results in a hardware exception to indicate a potential compromise by the OS.

The per-core hardware logic is added to enforce single or multi-cluster mode execution of the scheduled process(es). For single cluster mode execution, all core-level hardware resources are allocated to the scheduled process. However, multi-cluster mode execution supports the allocation of shared hardware resources to configure multiple strongly isolated clusters of cores. The size and bounds metadata for each cluster is read from the hardware map table, and utilized by both L1 and L2 cache/TLB misses. On an L1 cache/TLB miss, the map table uses the address and size/bound to determine the L2 cache/TLB within the cluster. Since all previous process execution is purged/flushed for strong isolation, this method results in allocation of requested data to the prescribed L2/TLB slice within the cluster. Similarly, on a L2/TLB miss, the map table again uses the address and size/bound to determine the memory controller allocated to the cluster. To ensure no packet drifts from its allocated cluster of cores, the on-chip networks implement non-interference routers. The details to ensure strong isolation during multi-cluster mode execution are described later in this section. When the OS prescribed time slot expires, the security kernel is invoked to purge/flush all shared hardware resources to ensure strong isolation between mode transitions. Consequently, the OS is allowed to schedule the next time slot and the execution continues until all co-located processes complete their assigned tasks.

The next subsections outline the ASM process-to-mode policy that performs mapping of processes to appropriate execution modes, the architectural details of single and multi-cluster modes and secure transitions between modes.

4.1 Process-to-Mode Mapping in Security Kernel

4.1.1 Sampling Phase. The purpose of sampling phase is to obtain the number of cores required by each process for near optimal performance. There are multiple potential metrics to correlate the performance of a process with the number of allocated cores, e.g. cache hit rate, cycles per instruction (CPI) and last-level cache misses per kilo instructions (MPKI). Prior works have shown that MPKI exhibits correlation for performance scaling as a function of increasing number of cores [8, 35]. In high core count multicore processors, the last-level cache is logically shared but physically partitioned among all the cores. Therefore, allocating more cores to a process results in allocation of more shared cache slices, which reduces the cache misses due to lesser address conflicts. For the workloads evaluated in this paper, MPKI shows a good correlation between completion time and number of cores allocated to a process. However, more elaborate metrics may be considered if the underlying workloads do not show good correlations with MPKI. For example, [27] utilizes a combination of MPKI and CPI to obtain optimal cache partitioning size for processes.

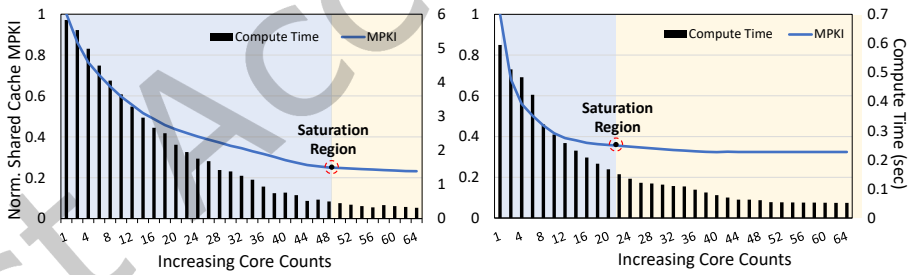


Fig. 4. Normalized MPKI and Completion time as a function of core count.

The security kernel in ASM computes the MPKI trend as a function of allocating different number of cores. First, the security kernel maps process threads to all available cores and executes a representative input. The hardware performance counters are used to capture the MPKI. This procedure is repeated by varying the allocation of core counts in a monotonically decreasing manner to obtain the MPKI trend

curve. As shown in figure 4, the completion time of process directly relates to the normalized shared cache MPKI. From these curves, it is evident that increasing the number of cores after a certain point does not improve the completion time or MPKI. This point is called the saturation point. The saturation point is a characteristic of the workload, e.g., a process with less dependence on the last-level cache saturates early.

To calculate this saturation point, each MPKI curve is scanned by comparing the discrete pairwise MPKI values from one end of the curve to the other, and compute the respective slope values. As the slope decreases, the MPKI trend flattens with increasing core counts. This suggests that allocating cores beyond a saturation point does not yield further performance benefits. A threshold slope value of 0.1 as utilized in prior work [35] determines the per-process saturation point, which is then passed to the mapping phase. The number of sampling points impacts the accuracy of the MPKI trend curve, and prior work has shown 64-point MPKI curve as optimal [35]. Therefore, ASM also uses 64-point MPKI curve for each process.

4.1.2 Mapping Phase. In the mapping phase, processes with high core-level parallelism are scheduled for execution under single cluster mode. Whereas, processes with combined core-level parallelism supported by the underlying multicore are scheduled together as a tuple for multi-cluster mode execution. Each multi-cluster mode tuple also computes the resource distribution for each cluster.

The security kernel performs the mapping phase, where it starts by iteratively creating two process tuples for 2-cluster mode execution. In each iteration, it performs a search by considering all combinatorial two-process pairs of unassigned processes. For example, for 8 processes there exist ${}^8C_2 = 28$ two-process pairs. For all pairs, it calculates the sum of saturation points obtained in the sampling phase. If there exists a viable two process tuple that has sum of saturation points less than available cores in the processor (R), that pair is pushed to *dual list* data structure with a proportional distribution of cores per cluster. For a given process, the ratio of the saturation point core count and the total resource needs of both processes is scaled using R. For n process tuple, the proportional distribution for any P_x process is calculated as follows:

$$P_x = \frac{R * Sat.pt_x}{\Sigma_{i=1}^{n} Sat.pt_i} \tag{1}$$

When a two-process tuple is pushed in the *dual list*, these processes are removed from mapping during next iterations. Each *dual list* entry contains the process IDs and their respective cluster size and boundary metadata. Similar process is followed when underlying hardware supports 3 or more clusters, where mapping phase generates *tri*, *quad*, *penta* or *hexa lists*. At the end of the iterative search, the remaining processes are classified as the ones requiring high core-level parallelism. These process IDs are pushed in the *mono list* data structure.

Consider an example where multicore processor has 64-cores and there are 3 co-located processes, A, B and C. First, the saturation point of each process is calculated in the sampling phase. Assume that the saturation points are 18, 15, and 37 respectively. The mapping phase calculates the cumulative saturation point of all combinations of two-process tuples, such that {A,B}(33), {A,C}(55) and {B,C}(52). The pair with the highest sum value that fits within the available cores in the system is selected, i.e., {A,C} (55). This {A,C} tuple resource allocation proportionally distributes the 64 core among them, i.e., A gets 21 and C gets 43 cores. This two-process tuple is pushed into the *dual list*. Since there is only one process B remaining after this iteration, it is pushed into the *mono list*.

4.2 Architectural Support for Single and Multiple Cluster Mode

To illustrate the architectural requirements for the ASM secure processor, consider Figure 5 that shows a stream of ten co-located processes. Here, two processes map to the mono list, while two tuples of

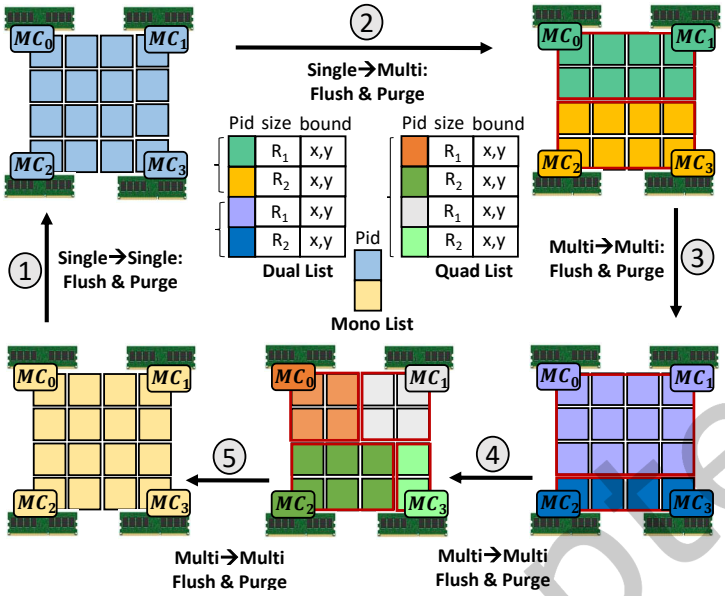


Fig. 5. Secure Transitions between ASM Execution Modes.

two-processes map to the dual list and a four process tuple maps to quad list. The representative OS scheduler first schedules mono list processes, each for a fixed time quantum. It then schedules the dual list tuples, each for a fixed time quantum and finally executes four process tuple. This schedule repeats until all processes complete their assigned work.

4.2.1 Single Cluster Mode. ASM single cluster mode temporally executes processes with high core-level parallelism. In this mode, all per-core resources (compute pipeline, L1-cache, private TLBs) are allocated to the scheduled process. However, these resources are purged (similar to MI6 [12]) when the executing process needs to exit. To provide single cluster mode with maximum cache resources, the last-level shared cache is fully allocated to the scheduled process. Again, to ensure strong isolation, the shared cache is also purged at each process exit. Since DRAM regions are large stateful resources, single cluster mode partitions the DRAM regions, similar to MI6 and Sanctum [12, 15]. The shared cache L2 misses are intercepted by the ASM hardware map table that determines the memory controller that maps the respective DRAM region. The memory access request is then routed via on-chip network to the respective memory controller. Therefore, memory controller buffers also need to be flushed when the executing process exits.

When a single cluster mode process finishes its execution for the allocated time quantum, the next mono list process is scheduled for execution, shown as ① in Figure 5. ASM requires the processor to perform **single-single transition** for strong isolation by performing a purge of all shared hardware resources. The representative multicore has two types of shared resources:

- Non-persistent resources like load-store buffer queues, memory controller queues and on-chip network buffers are drained by issuing fence operations. For example, each core issues a MEMORY_FENCE( ) operation to drain all queued memory requests.

- Persistent resources like private and shared caches/TLBs are purged by performing a flush-and-invalidate operation for each cache line. These flush and invalidate operations are invoked on per core granularity to clean up the micro-architecture state.

The overhead added to clean-up the execution state of both persistent and non-persistent resources is termed as ***flushing overhead***.

4.2.2 Multiple Cluster Mode. To spatially execute a multiple process tuple in strongly isolated clusters in the multiple cluster mode, all the persistent and non-persistent hardware resources must be spatially distributed. Following steps are performed in ASM to distribute resources:

- **Distribution of Per-Core Resources:** For strong isolation, core resources are distributed such that clusters do not overlay with each other. Each cluster is assigned a set of cores, dictated by size and bound values (cf. Section 4.1.2) in the hardware map table, and all threads of the process are pinned to cores within the cluster. Doing so allows processes to execute across multiple clusters with strong isolation.
- **Distribution of Shared Cache:** The partitioning of the shared cache is unique because each cache line has a single L2-slice allocated to ensure the cache coherence invariant. Since the shared cache is logically shared but physically partitioned across the chip, it is distributed such that each cluster gets to access a dedicated set of shared cache slices. Consequently, all cache lines of a process are allocated to the cache slices dedicated to the cluster. This is accomplished via the hardware hash function that uses the memory access request address and cluster boundary from map table to allocate shared cache slices from within the cluster.
- **Isolation of NoC Traffic:** For each cluster, the network traffic needs to be routed in a way that packets from one cluster remain isolated from packets of the other cluster. This isolation can be achieved either temporally using time division multiplexing [45], or spatially using deterministic routing protocol that confines each packet source to destination path to never exit its allocated cluster of cores [38]. For example, X-Y routing in a 2-D mesh can be used along with clusters of cores confined within X-Y dimension, such that any source to destination of a packet never exits the cluster. The multiple cluster mode in ASM is evaluated using spatial isolation of the NoC traffic.
- **Distribution of Memory Controllers:** The memory controllers are distributed across the clusters to avoid interference in controller queues. This isolation can also be achieved temporally and/or spatially. In temporal isolation, the memory controllers are shared by clusters using time division multiplexing [43]. In spatial isolation, the memory controllers are statically partitioned across the clusters such that they never overlap each other [38]. The multiple cluster mode in ASM is evaluated using statically partitioned memory controllers.
- **Distribution of DRAM:** DRAM is partitioned such that each process is allocated a dedicated region [12, 15]. Processes scheduled to execute in any given cluster map their data to that cluster's dedicated DRAM regions, which are inaccessible to the other cluster. The shared cache misses are routed to the memory controller(s) that map the respective DRAM region(s).

When the second single cluster mode process finishes its execution for the allocated time quantum, the first dual list tuple is scheduled for execution, shown as ② in Figure 5. ASM requires the processor to perform **single-multi transition**. To maintain strong isolation, the persistent and non-persistent shared hardware resources are purged, incurring the flushing overhead similar to the **single-single transition**. Next, the security kernel activates the processor to use multiple cluster mode execution. The threads for dual list processes are pinned to their respective cores in the allocated cluster. Moreover, all L1/TLB and

L2/TLB cache misses are redirected to the hardware map table to perform the multiple cluster mode specific routing of requests and their replies from the on-chip cache hierarchy.

When the second multi-cluster mode tuple is scheduled for execution (③ in Figure 5), the processor must perform a **multi-multi transition**. To maintain strong isolation, the flushing overhead similar to the **single-single transition** is incurred. In addition, the number of cores allocated to each cluster may require adjustment to accommodate the demands of the incoming tuple. This is achieved by updating the hardware map table with the new size and boundary values for the clusters. Next, the processor commences with multi mode execution as described earlier.

When second dual list tuple has executed for its fixed time quantum, the four process tuple is scheduled to execute. As indicated by ④ in figure 5, a multi-multi transition occurs, which includes purge of all private and shared hardware resources. The hardware map table is updated with new cluster sizes and boundary values for incoming tuple of four processes, to create four strongly isolated clusters.

When transitioning from multiple to single cluster mode (⑤ in Figure 5), the processor must perform a **multi-single transition**. Again, the flushing overhead similar to the **single-single cluster transition** is incurred. Next, the processor is transitioned to use the default mechanism that maps all shared hardware resources to the incoming mono list processes.

4.3 Hardware and Security Kernel Interactions with the OS

The security kernel invokes the operating system to schedule the mono and multi-cluster lists. For ASM, the OS must be enhanced to support scheduling of processes and tuples from these lists. On each scheduling event, the security kernel gets invoked to purge the shared hardware resources, and set mode specific flags for the hardware. In case of I/O requests, if one of process in tuple switches to idle state, the other process continues to execute until the next scheduling interval. When a process terminates, the security kernel removes the process from its respective mono or multi-cluster list. In case of multiple cluster mode, if one process in a tuple terminates, the security kernel marks that process as inactive. In this paper, the hardware continues execution with active processes in their clusters, while the other cluster(s) are idle. When all processes in the tuple complete, the security kernel removes the tuple from the respective list. Depending on system requirements, different scheduling algorithms (e.g. FCFS, SJF, MLFQ etc) can be enhanced accordingly to handle I/O and termination scenarios.

5 METHODOLOGY

5.1 Secure Processors on Tilera Tile-Gx

The ASM architecture is prototyped on a real machine, *Tilera® Tile-Gx72™* [46]. The reason for choosing this specific processor is due to the availability of the necessary hardware primitives to implement strong isolation methods required for the MI6, Ironhide, and the proposed ASM architectures. Tile-Gx72™ is a tiled multicore architecture comprising of 72 tiles, where each tile consists of a 64-bit multi-issue core pipeline, private level-1 (L1) data and instruction caches of $32KB$ each, private instruction and data TLBs of 32 entries each, and a $256KB$ slice of the shared level-2 (L2) cache (LLC capacity of $18MB$). It comprises 2-D mesh networks with *X-Y routing* for the on-chip cache coherence and memory controller traffic.

A customized CentOS Linux distribution runs on Tilera with additional capabilities of Tilera Multicore Components (TMC) library API. This library includes facilities to form clusters of cores, manage network traffic across clusters, regulate on-chip and off-chip data access controls, and manage shared cache data placement. The off-chip memory is accessible using four on-chip 72-bit ECC protected DDR memory

controllers attached to independent physical memory channels. To model all evaluated secure processors, 64 cores were utilized out of the 72 available cores.

5.1.1 MI6 and MI6-Optimus Secure Processors. In MI6 [12] the co-located processes are executed temporally. Each process is provided with statically partitioned shared L2 cache slices, and DRAM memory regions. Therefore, based on the number of co-located processes, the L2 cache and DRAM are partitioned proportionally. For example, for a two process execution, 32 L2 slices and half of the DRAM regions are allocated to each process. The L2 cache partitioning is achieved by using the *local homing* capability enabled on the Tilera machine [1]. Here, each page in a process is mapped to a specific L2 cache slice using the `tmc_alloc_set_home(&alloc, core_id)` API call. This enables an address mapping in hardware that translates an address based on the core ID of the L2 cache set as the home location.

For MI6, all other shared on-chip hardware resources are purged on each process exit. To *purge* the private L1 cache, a dummy buffer of size equal to the cache size is read into each L1 cache. Reading this buffer removes all secure process' data from the private L1 cache. A similar purging technique is used for the TLBs. However, all L1 caches and TLBs are purged on a per-core granularity in parallel. Then, a memory fence operation (`tmc_mem_fence()` call) is performed to ensure the propagation of dirty data to the respective L2 cache slice. Finally, the queues/buffers of all memory controllers are purged using `tmc_mem_fence_node(controller_id)` call that writes back all modified data to DRAM. These flush operations are reported as *flushing overheads* in the evaluation.

For MI6-Optimus [35], the only difference is in the distribution of L2 slices. MI6-Optimus deploys a sampling phase in the security kernel to calculate the appropriate number of L2 cache slices per process. The saturation point of each process is calculated using a 64-point MPKI curve. Then, L2-slices are proportionally distributed among all co-located processes using equation 1 from Section 4.1.

5.1.2 Ironhide Secure Processor. The Ironhide secure processor executes two processes concurrently using two strongly isolated clusters of cores [35, 38]. It first invokes the security kernel to determine the schedule of co-located two-process tuples on the clusters. To model Ironhide on Tilera, the threads of both processes are pinned to the cores of their assigned clusters via `tmc_cpus_set_my_cpu(tid)` API calls. Setting the thread affinity distributes private core-level resources, L1 caches and TLBs among two clusters, whereas distribution of L2 cache slices requires a specific homing policy. The L2 cache slices are allocated to a given cluster of cores using the *hash-for-home* hardware capability in Tilera [1]. Here, the size and bound of each cluster of cores is used to configure the per-core TLBs. During the virtual to physical address translation in each core, the physical address uses a special hashing feature to determine a user specific tile for L2 cache homing. On each two-process tuple exit, the cluster size and bound may change for the next tuple. In addition to purge discussed in Section 5.1.1, *all* L2 cache slices are flushed and invalidated using the `tmc_mem_flush_l2()` API call to ensure strong isolation of L2 caches across execution of co-located process tuples. This purge operation involving flushing of private core level resources, as well as, L2 cache slices is reported as flushing overhead for Ironhide in section 7.

The L2 cache misses of a cluster are routed to their respective DRAM regions via dedicated memory controllers. This capability is enabled by using `tmc_alloc_set_nodes_interleaved (&alloc, pos)` API calls. Here, `pos` represents the bit-mask representation of memory controllers to be selected, e.g., `pos = 0b0011` is used to dedicate MC_0 and MC_1 to one cluster, whereas `pos = 0b1100` (MC_2 and MC_3) for second cluster. *Tile-Gx72*^{TM} implements deterministic *X-Y routing* with 2-D mesh network topology, which is used to isolate the network traffic by routing each packet to/from the allocated clusters' memory resources.

5.1.3 The ASM Secure Processor. For a set of co-located processes, ASM first invokes the security kernel that performs the sampling for each process (cf. Section 4.1.1). The computed saturation core count for each process is used in the mapping phase to map processes to their respective execution modes.

The *single cluster* mode of ASM is modeled in a similar fashion as MI6-Optimus, as described in Section 5.1.1. However, the *single cluster* mode does not statically partition the shared L2 cache slices. Instead, it provides all L2 cache slices to the process under execution, and *flush-and-invalidate* them by invoking the `tmc_mem_flush_l2()` API calls in each core. The *multiple cluster* mode of ASM creates two strongly isolated clusters of cores to execute two-process tuples, as described in Section 5.1.2. The purge operations in ASM are identical to Ironhide, with similar flushing cost as Ironhide.

ASM is evaluated separately using four configurations on Tilera. Mono executes all co-located processes in single-cluster only, irrespective of their resource demands. Dual is identical to the Ironhide secure processor, i.e. it always run in dual cluster formation, without any adaptive transitions between mono or dual cluster. ASM-2 Cluster is the proposed adaptive secure processor that maps and executes processes using both mono and dual cluster formations. ASM-3 Cluster utilizes tri-cluster formation in addition to the mono and dual clusters. For tri-cluster, 3-process tuples execute spatially in three clusters of cores that have sum of saturation points less than 64.

5.2 The ASM Secure Processor on Multicore Simulator

To study the scalability of ASM on high core count with multiple clusters, a RISC-V multicore simulator is used [4]. The simulator uses performance models of MIT's Graphite simulator [34] for multi-level cache hierarchy, NoC and memory controller modeling. The RISC-V functional model i.e. Instructions implementation, register states and MMU in simulator are described using Architecture Description Language (ADL) [3] models.

Architectural Parameter	**Simulator**	**TILE-Gx72**
Number of Cores	up to 256 @ 1 GHz	72 @ 1 Ghz
Compute Pipeline per Core	In–Order, Single–Issue	64-bit VLIW
Memorgy Subsystem		
L1–I Cache per core	32 KB, 2–way Assoc.	32 KB, 2–way Assoc., 2 cycles
L1–D Cache per core	32 KB, 2–way Assoc.	32 KB, 2–way Assoc., 2 cycles
L2 Inclusive Cache per core	256 KB, 4–way Assoc.	256 KB, 4–way Assoc., 10 cycles 2 cycle tag, 4 cycle data
Directory Protocol	Invalidation–based MESI	Invalidation–based
DRAM Bandwidth	64 GBps	64 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	64 bits

Table 1. Architectural parameters for evaluation

Table 1 shows the architectural parameters of simulator. These parameters are tuned to run 128 and 256 core simulations while matching Tilera architecture specifications. Similar to ASM implementation on Tilera, the shared cache model and NoC models on simulator are updated to create clusters of cores. The ASM process-to-mode mapping creates schedule for co-located processes on tri, quad or penta cluster configuration. The simulator creates multiple clusters as mentioned in section 4.2.2. ASM is evaluated on simulator using 128 and 256 cores while creating up-to 6 cluster configuration. ASM-4 cluster represents mono, dual, tri and quad cluster modes, whereas, ASM-6 represents mono, dual, tri, quad, penta and hexa cluster modes.

5.3 Security Kernel and OS Scheduler

The security kernel interacts with the processor hardware, and it is invoked at (1) startup time when co-located processes are scheduled, and (2) on each process/tuple exit when the microarchitecture state is purged and/or mode transition incurs. The security kernel hands off the process level metadata to the operating system incurring a few cycles overhead. Moreover, the interactions between security kernel and the hardware take several processor cycles to update the map table entries. A round-robin scheduler with fixed time slice execution per process is used for evaluation of all secure processors. An interval timer is configured to trigger an interrupt every $500ms$ to schedule processes. This time slice quantum is empirically determined in this paper. For ASM, all single cluster mode processes are scheduled before the multiple cluster mode processes.

5.4 Benchmarks and Metrics

Seven parallel processes from four different security-critical computing domains are considered. Three parallel graph algorithms, Single Source Shortest Path (SP), PageRank (PR), and Triangle Counting (TC) are acquired from the CRONO [6] benchmark suite. They are all executed using a real world California Road Network graph [16]. A mission planning algorithm, Artificial Bee Colony [47] (AB) is also analyzed, which is adopted from the advanced driver-assistance system (ADAS). Two machine learning benchmarks, AlexNet (AN) and Squeeze-Net (SN) are also used. ImageNet [17] is provided as an input to these benchmarks. Lastly, Advanced Encryption Standard (AE) is considered for analysis due to its importance in the domain of encryption. We consider all combinations of the aforementioned 7 processes to evaluate various 2, 3, 4, 5, 6, and 7 co-located processes. For example, two co-located processes results in ${}^7C_2 = 21$, and four give ${}^7C_4 = 35$ combinations. However, we randomly pick a subset of these combinations, and execute each set of co-located processes with 100 inputs for evaluation.

ASM performance against MI6, MI6-Optimus, and IRONHIDE secure processors is evaluated by measuring completion time of different co-located processes. The performance metric used for evaluation is completion time of the co-located processes until each process completes 100 inputs. The completion time also includes the overhead occurred during the sampling phase and flushing occurring on each process-level context switch.

6 SECURITY ANALYSIS

In a multicore processor, shared hardware resources such as cache, network-on-chips, and memory controllers are targeted for timing-based side-channel attacks. For example, cache-based timing attacks [49][22][30] exploit timing variations due to sharing of cache states within virtually isolated applications. Similarly, NoC [7][44] and Memory controller [42] shared hardware are carefully contented to create timing-based covert-communication attacks. This section discusses the implementation of state-of-the-art Flush+Reload [49] attack on cache, ConNOC [7] on NOC hardware, and Contention attack [42] on memory controller for Tilera Tile-Gx processor. These attacks are conducted without any strong isolation implementation and for temporal execution based single cluster mode and spatial execution based multiple cluster mode. Without protection enabled, all of these attacks show a True Positive (TP) rate of above 90% and a high Discrimination index (DI) value. TP rate and DI quantify the efficacy of timing-variation attacks [41] [32]. We have conducted a security analysis of ASM processor in single cluster mode, multiple cluster mode, and transitions between them i.e 1) single-single, 2) single-multi, 3) multi-multi, and 4) multi-single.

6.1 Timing Attacks on Unprotected Hardware

Without any mitigation scheme enabled, the cache, NoC and memory controller channels in a multi-core processor are exploited by co-located processes for timing-based side channels using practical attacks. It is assumed that the attacker can co-locate data on individual hardware and a combination of shared hardware resources.

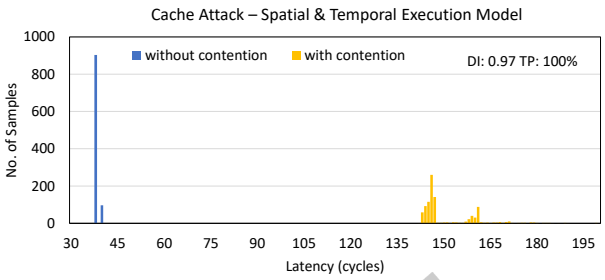


Fig. 6. Timing variations for cache attack without any mitigation.

6.1.1 Cache attack. Shared cache memory significantly reduces the memory access time. Data access from shared cache memory (i.e. cache-hit) takes ∼10-30 cycles, whereas data access from outside of cache (i.e. cache-miss) takes ∼110-150 cycles, as observed in Figure 6. This high timing difference of ∼120 cycles can be exploited to create a covert-communication attack or leak secret information. For example, a transmitter application sharing the cache can control the state of the cache by placing the data or flushing the data. A receiver application accessing shared data and measuring the time can use this timing information to infer secret information. A cache-hit is translated to secret bit 1, whereas a cache-miss translates to secret bit 0. Figure 6 shows timing-variation histogram on unprotected hardware with temporal and spatial execution model with a TP rate of 100% and high DI value.

6.1.2 NoC attack. NoC shared resources (i.e., links, buffers, crossbars) are shared by all cores and processes. An adversary can carefully contend on this hardware to create timing variations and consequently use these timing variations for covert communication or secret information leakage. For example, a transmitter application can carefully occupy these shared hardware resources and create contention for other applications. A receiver application sharing these NoCs with a transmitter application, can observe contention or no-contention situations based on the time to access data over NoC. Fast access over NoC translates to secret bit 1, whereas slow access over contented NoC hardware translates to secret bit 0. This NoC contention attack works on unprotected temporal and spatial execution models. In temporal execution models, pre-occupied shared resources cause timing delays. Contrary, in the spatial execution model, the scheduling of NoC routing is a primary reason for timing variations. Although NoC has low timing variations of ∼5-7 cycles, the information theory technique of repetition codes [41] are used to improve the efficacy of the attack. Figure 7 shows timing variation histogram for temporal and spatial execution model on unprotected NoC hardware with a TP rate of above 90% and high DI value in absence of any mitigation scheme.

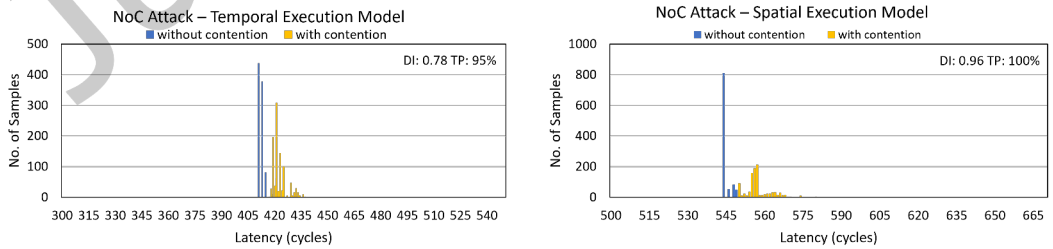


Fig. 7. Timing variations for NoC attack under Spatial/Temporal Execution Model

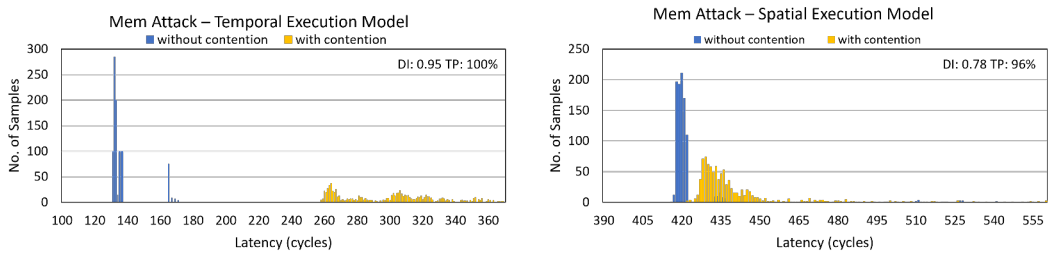


Fig. 8. Timing variations for memory controller attack under Spatial/Temporal Execution Model

6.1.3 Memory Controller attack. Memory controller queues are shared resources within all processes. An adversary application can occupy memory controller queues to create contention and timing variations for other applications. These timing variations are exploited to create information leakage and covert communication attacks. For example, a transmitter application intentionally occupies shared memory controller hardware. A receiver application sharing the same memory controller observes the timing variation of ∼5-10 cycles based on the state of memory controller queues. A contented memory controller results in a slower data access compared to a non-contended one. Fast access (i.e., less time) translates to secret bit 1, and slow access (i.e., high time) translates to secret bit 0. This process is repeated to covertly leak secrets. Similar to the NoC attack, information theory concepts of repetition codes are used to improve the efficacy of the attack. Memory controller attacks are practical for temporal and spatial execution models on unprotected hardware. Figure 8 shows the timing histogram of memory controller attack on temporal and spatial execution models with a TP rate of above 90% and high DI value in absence of any mitigation scheme.

6.2 The ASM Protected Multicore Hardware

When ASM is enabled on multicore hardware, it ensures strong isolation between co-located processes by flushing shared hardware resources and/or creating strongly isolated clusters of cores. Cache-based timing attacks exploit timing variations due to the sharing of cache states between virtually isolated processes. ASM in single cluster executes two processes in temporal execution model and purges cache state on entry/exit of each process. The purging of cache state results in cache miss for otherwise cache hit and protects against cache-based timing attacks. Similarly, ASM purges NoC and memory shared resources (i.e., queues) to protect against NoC and Memory Controller attacks. Whereas, ASM multiple cluster mode enforces strong isolation by creating clusters and partitioning cache, NoC, and memory controller hardware. This strong isolation prevents timing-attacks.

Attacks	Targeted Hardware	Single Cluster Mode	Multiple Cluster Mode	Transitions			
				Single -Single	Single – Multi	Multi – Multi	Multi - Single
Flush+Reload	Cache	TP: 50%, DI: 0.01	TP: 50%, DI: 0.01	TP: 51% DI: 0.01	TP: 50% DI: 0.02	TP: 50% DI: -0.01	TP: 49% DI: -0.02
ConNOC	Network-on-Chip	TP: 50%, DI: -0.02	TP: 50%, DI: 0.01	TP: 49% DI: -0.01	TP: 51% DI: -0.02	TP: 51% DI: -0.02	TP: 50% DI: -0.01
Memory-Controller	Memory-Controller	TP: 49%, DI: 0.01	TP: 50%, DI: 0.01	TP: 51% DI: -0.03	TP: 49% DI: 0.01	TP: 50% DI: -0.02	TP: 51% DI: 0.01

Table 2. TP/DI for cache, network-on-chip, and memory controller attacks with ASM.

The ASM execution model includes four transitions: single-single, single-multi, multi-multi and multi-single. These transitions are performed by security kernel which ensures the strong isolation by flushing all shared hardware resources before bringing-in next process/tuple for execution. Table 2 summarizes the TP rate and DI for cache, NoC, and memory controller hardware attack with ASM protection enabled.

Table 2 shows the efficacy of ASM while executing under single cluster mode, multiple cluster mode and transitions between single and multi cluster modes. A TP rate of ∼50% shows that attacks are fully mitigated, and DI of ∼0 confirms negligible timing variations.

7 PERFORMANCE EVALUATION

7.1 Capture the Shared Cache MPKI Trend

In a shared memory multicore setup, shared cache is logically shared but physically partitioned. Therefore, allocating more cores to a process results in allocation of more shared cache slices. A sweep study of shared cache MPKI for various number of cores captures the behavior of processes on increasing the allocated shared cache slices. Figure 9 shows the normalized MPKI values against core count for all workloads mentioned in section 5.4. Using the gradient based approach mentioned in section 4.1.1, the saturation points are calculated for the workloads. These MPKI curves show that after saturation point, there is negligible variation in MPKI values on further increasing core count. For example, the performance of ABC saturates at 54 cores, and allocating more cores does not yield further improvements. Therefore, in multiple cluster mode ASM can execute another process in parallel using the remaining 10 cores. If there is no such process with saturation point of ≤10, ASM executes ABC in the single cluster mode.

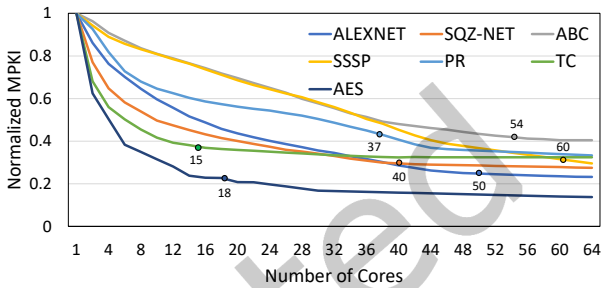


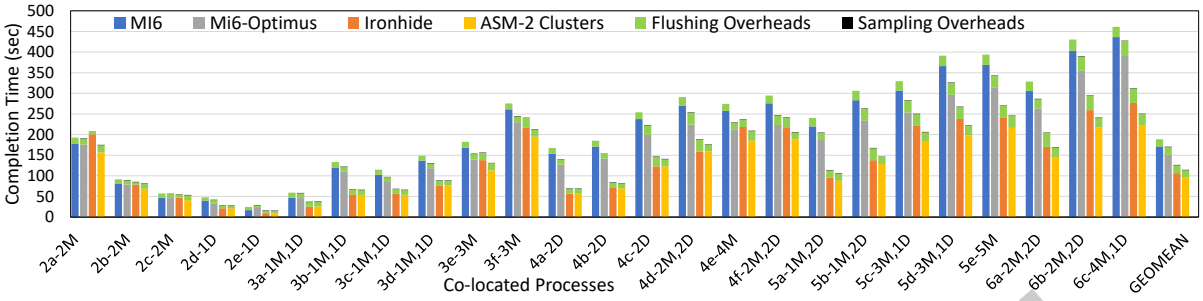
Fig. 9. Benchmark applications MPKI trends

This sampling phase is part of security kernel, such that whenever a process is created, ASM has to bear this one-time penalty to calculate the saturation point by invoking security kernel. Later in evaluation, it is shown that for long running processes, sampling phase does not add considerable overhead.

7.2 Various Secure Processes versus ASM-2 Cluster on 64-Core Tilera

Figure 10 shows completion times of 25 selected combinations of 2, 3, 4, 5 and 6 processes. These combinations are assigned ID based on number of co-located processes. Figure 10 also shows the ASM mappings generated by using the mapping phase. If cumulative saturation point of all the processes in a tuple is ≤ total resources, then ASM maps that tuple into multiple cluster mode, i.e., 4-cluster mode is created for 4 co-located processes. If cumulative saturation point is > total core count, then an iterative search is performed, as mentioned in section 4.1.2. A single process in braces identifies a single cluster mode process, while multiple processes in braces represent multiple cluster mode tuple of processes. The last column shows the mapping of processes for the Ironhide processor, where X represents an idle cluster of cores. Figure 11a shows normalized geometric means of completion time for 2, 3, 4, 5 and 6 co-located processes. Total number of flushes observed in different secure processor setups are reported in figure 11b. Figure 10 shows that MI6-Optimus performs ∼10% better than MI6 due to proportional distribution of L2-cache slices versus static partitioning. However, with increased number of co-located processes, the proportional distribution of L2-slices allocates small chunks of the shared cache to each process that results in increased compute time due to capacity misses.

Ironhide executes 2-process tuples in two clusters of cores to reduce the overall number of flushes (shown in 11b). However, the flushing overheads reported in figure 10 are similar to MI6-Optimus because flushing cost of Ironhide is higher than MI6-Optimus due to additional purging of shared L2-cache. With increased number of co-located processes, Ironhide performs better than MI6-Optimus. However, due to its 2-cluster



	ASM-2 Cluster	Ironhide
2a	{AB},{SP}	{SP-X},{AB-X}
2b	{AN},{SN}	{AN-X},{SN-X}
2c	{SN},{PR}	{SN-PR}
2d	{SN-AE}	{SN-AE}
2e	{TC-AE}	{AE-TC}

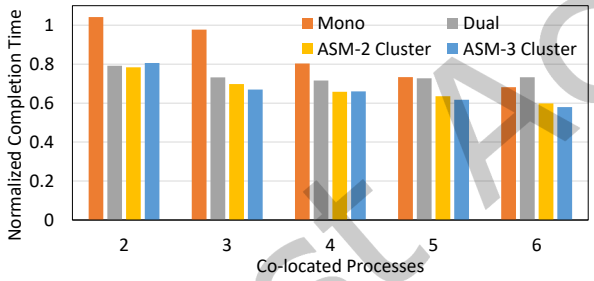
	ASM-2 Cluster	Ironhide
3a	{TC},{AE-SN}	{AE-SN},{TC-X}
3b	{SP},{AE-TC}	{AE-SP},{TC-X}
3c	{AN},{AE-SN}	{AE-AN},{SN-X}
3d	{SP},{TC-AN}	{TC-SP},{AN-X}
3e	{AB},{AN},{AE}	{AE-AB},{AN-X}
3f	{AN},{SP},{AB}	{AN-X},{AB-X},{SP-X}

	ASM-2 Cluster	Ironhide
4a	{TC-PR},{AE-SN}	{SN-PR},{AE-TC
4b	{TC-AN},{AE-PR}	{AE-AN},{TC-PR}
4c	{SN-AB},{TC-AN}	{TC-AB},{AN-X},{SN-X}
4d	{SP-AB},{TC-AE}	{AE-SP},{TC-AB}
4e	{AN-SN},{PR-SP}	{SN-PR},{AN-X},{SP-X}
4f	{SP-AB},{TC-AN}	{TC-SP},{AN-X},{AB-X}

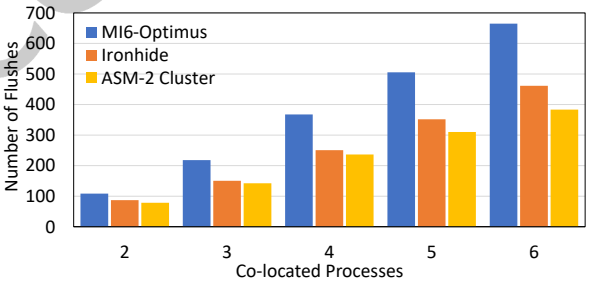
	ASM-2 Cluster	Ironhide
5a	{PR}, {AE-SN}, {TC-AN}	{SN-PR},{AE-AN},{TC-X}
5b	{AB},{AE-SN}, {TC-AN}	{AE-AB},{AN-TC},{SN-X}
5c	{PR},{SP},{SN}, {TC-AN}	{SN-PR},{TC-SP},{AN-X}
5d	{SP},{AE},{AB}, {TC-AN}	{AE-SP},{TC-AB},{AN -X}
5e	{AN},{SN},{PR}, {SP},{AB}	{SN-PR},{AN-X},{SP-X},{AB-X}

	ASM-2 Cluster	Ironhide
6a	{PR},{AB},{AE-SN},{TC-AN}	{SN-PR},{AE-AB},{TC-X},{AN-X}
6b	{SP},{AB},{AE-SN},{TC-AN}	{AE-SP},{TC-AB},{AN-X},{SN-X}
6c	{PR},{AB},{SP},{SN},{TC-AN}	{SN-PR},{TC-SP},{AN-X},{AB-X}

Fig. 10. Completion time comparison of co-located processes under the MI6, MI6-Optimus, Ironhide and ASM architectures.



(a) ASM variants relative to Mi6-Optimus on Tilera processor.



(b) Fig.Flush count for various secure processor setups.

Fig. 11. ASM variants and flush counts on Tilera

only execution, it suffers from performance degradation for processes with high core-level parallelism. For example, in figure 10, tuple IDs 2a, 3e, 4e show that Ironhide is at-par or worse than MI6-Optimus, due to higher flushing cost and under utilization of resources as compared to MI6-Optimus. ASM-2 cluster enables an adaptive execution model via single and 2-cluster modes. Processes with high core-level parallelism (high saturation points) are executed in single cluster mode to fully utilize all available cores. On the other hand, processes with relatively low saturation points are combined to execute in 2-cluster mode, improving performance by better utilizing the available cores and reducing the overall number of purges. All secure processor setup except MI6 incur sampling overhead for capturing the MPKI trend, but it is a one-time cost with low overhead incurring at each process creation.

For 2 co-located processes, ASM-2 cluster benefits from single-cluster mode execution for 2a, 2b, 2c where it executes both processes with all available cores. Thus, improving their competition time which in turn decreases the number of flushes by consuming less time slices. For tuples 2d and 2e, ASM-2 cluster executes in 2-cluster mode where it matches the performance of Ironhide. For 3 processes, ASM-2 cluster benefits from its transitions between single and 2-cluster modes, such that for tuples 3a to 3d it executes two processes in 2-cluster mode and one process in single-cluster mode. The processes that are combined to run in 2-cluster mode are shown in table under figure 10. The multiple cluster mode execution reduces the number of flushes for less resource intensive processes, and single-cluster mode improves completion time of processes with high core-level parallelism. For 4 processes, ASM-2 cluster either matches Ironhide or out-performs it when there are resource intensive processes. Similarly, for 5 and 6 processes, due to better resource utilization via transitions between single and 2-cluster modes, ASM-2 cluster improves the completion time of tuples and also reduces the number of flushes. Figure 11b shows that as the number of co-located processes is increased, the difference in number of flushes for Ironhide and ASM-2 cluster is continuously increasing. Overall, ASM-2 cluster shows ∼58%, ∼49%, and ∼11% improvement over MI6, MI6-Optimus, and Ironhide by better utilizing the available core-level resources.

To further evaluate the benefits of ASM, a tri-cluster mode is added. Here, one cluster is provided with 2 dedicated memory controllers, whereas the remaining two clusters utilize one memory controller each. Five out of 25 tuples take advantage of the tri-cluster mode to exploit the exposed parallelism. The geometric mean performance gains presented in figure 11a show that ASM with 3-clusters either performs at-par or gives slight benefit over ASM-2 cluster. We postulate that as the number of cores are increased in the processor, ASM-3 cluster will become more effective.

7.2.1 Case Study of All Co-located Processes. A detailed evaluation of all 7 co-located processes is shown in Figure 12 using a single input for each process. The sampling phase performed by the security kernel is omitted since it is a common overhead in each evaluated secure processor setup. The security kernel first purges the secure processor, and invokes the operating system to schedule processes with 500 *ms* time slice.

The partitioning of L2-slices in MI6-Optimus results in small chunks of shared cache being allocated to each process. This significantly impacts performance due to capacity misses resulting in more compute time. High compute time means more time slices are consumed to complete the process, which results in a high number of purges. However, each flushing cost in MI6-Optimus is less than Ironhide and ASM-2 cluster because it does not require L2-cache purging.

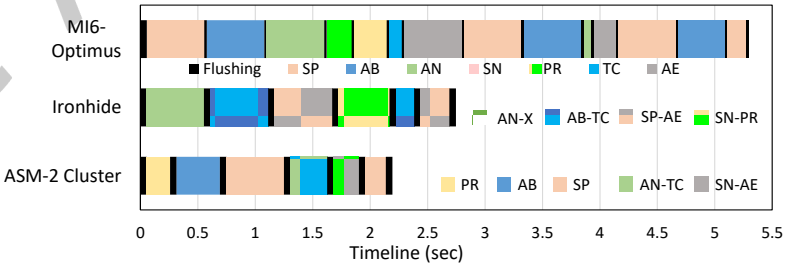
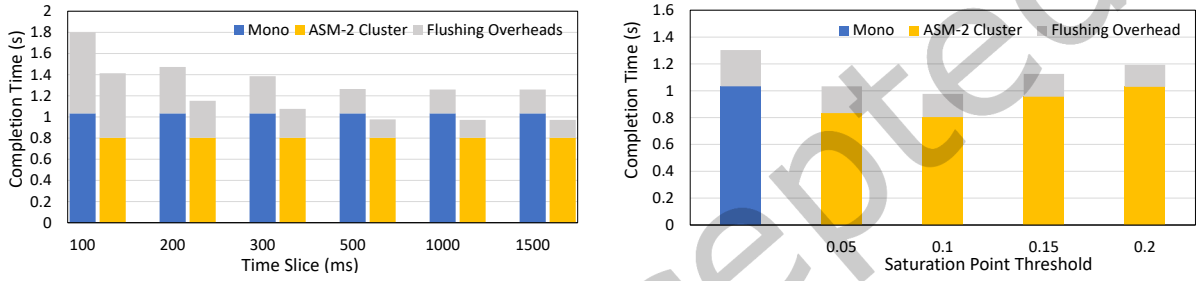


Fig. 12. Execution breakdown for all co-located processes.

Ironhide combines the AB-TC, SP-AE and SN-PR to form 2-process tuples for execution in 2 clusters, which also helps reduce the number of flushes. Both AB and SP have high core-level parallelism and high saturation points, i.e., 54 and 60 respectively. Their resource demands are not met when they are executed with TC and AE, thus resulting in over-utilization of cores which impacts performance. Ironhide also executes AN with its saturation point cores in one cluster, while the other cluster remains idle. This results in under-utilization of the available cores. ASM-2 cluster resolves this over- and under-utilization

of hardware by combining AN with TC, and executing AB and SP in single cluster modes. This results in a 14% performance improvement in ASM-2 cluster relative to Ironhide.

7.2.2 Time Slice Selection. So far, the evaluation is presented with a time slice (T) setting of $500ms$. However, this parameter impacts performance since for high time slice values the processor may incur load imbalance, and for small values the context switch and associated flushing overheads may overwhelm performance. Figure 13a shows the geometric mean completion time results for a range of time slice values using selected combinations of 2, 3, 4, 5 and 6 processes. ASM-2 cluster and Mono-only (worst case scenario with always single-cluster) both incur higher flushing overheads when time slice parameter decreases. This is expected since the addition of context switches requires flushing of private-shared resources to ensure strong isolation. The $1500ms$ incurs a handful of context switches, resulting in best performance, while $100ms$ incurs 14 switches on average. The $500ms$ time slice shows best performance and this value is at par with modern operating systems, thus it is chosen as the default.



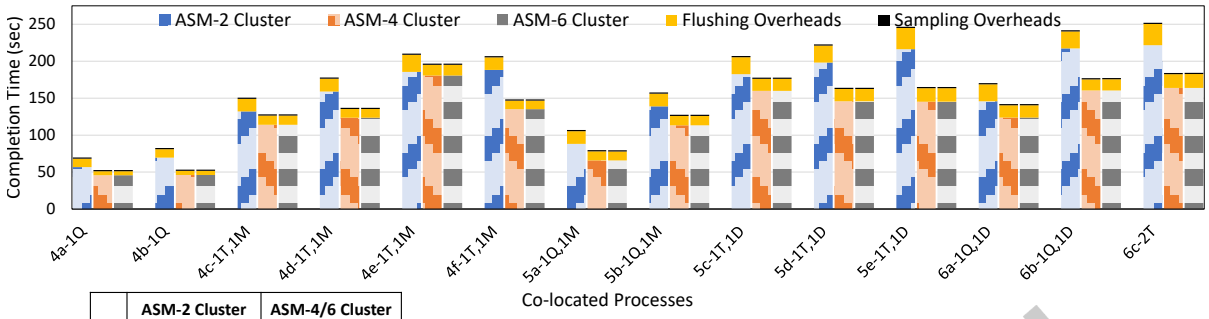
(a) ASM with various Time Slices. (b) ASM with various saturation thresholds.

Fig. 13. Sensitivity Studies on Tilera

7.2.3 Sensitivity to Saturation Point Selection. The saturation point from a per-process MPKI trend is captured by scanning the trend from the end to the point where the absolute slope value reaches or becomes greater than a defined threshold. The evaluation has so far considered the saturation point selection threshold value of 0.1. Figure 13b shows the impact of varying threshold values. The higher saturation points select higher core counts for processes to operate at. This reduces the number of multi-cluster mode tuples, thus not allowing the system to take advantage of this mode for better parallelism across two processes. Therefore, higher than 0.1 threshold value decreases performance. It is also observed that a lower threshold value selects lower than optimal core counts for a process, resulting in too many 2-cluster mode tuples. This also adversely affects performance, since single-cluster mode execution is better for processes that have higher core-level parallelism. Therefore, lower than 0.1 threshold also decreases performance.

7.3 The ASM Secure Processors with 128 and 256 Cores

The scalability of ASM at higher core counts is evaluated on the multicore simulator with 128 and 256 cores. Figures 14 and 15 represent the completion times and ASM mappings for various co-located processes on the 128 and 256 cores simulated multicores. The co-located processes shown in these figures are same as mentioned in section 7.2.. These simulations deploy ASM variants with up to 2, 4 and 6 clusters, i.e., ASM-4 cluster can execute in mono, dual, tri and quad cluster formation based on schedule generated by mapping phase, whereas ASM-6 cluster can execute up to penta and hexa cluster configurations. Both figures 14 and 15 do not show 2 co-located processes, because there is no variation in completion times as compared to figure 10. This is because 2 co-located processes either get scheduled to

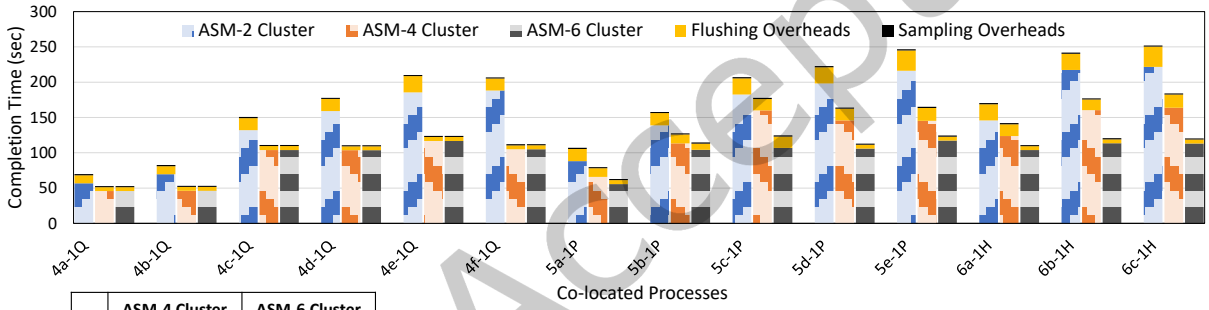


	ASM-2 Cluster	ASM-4/6 Cluster
4a	{TC-PR},{AE-SN}	{TC-PR-AE-SN}
4b	{TC-AN},{AE-PR}	{TC-AN-AE-PR}
4c	{SN-AB},{TC-AN}	{TC-AN-SN},{AB}
4d	{SP-AB},{TC-AE}	{SP-TC-AE},{AB}
4e	{AN-SN},{PR-SP}	{AN-SN-PR},{SP}
4f	{SP-AB},{TC-AN}	{SP-TC-AN},{AB}

	ASM-2 Cluster	ASM-4/6 Cluster
5a	{PR},{AE-SN},{TC-AN}	{PR-AE-TC-AN},{SN}
5b	{AB},{AE-SN},{TC-AN}	{AB},{AE-SN-TC-AN}
5c	{PR},{SP},{SN},{TC-AN}	{PR-SN-AN},{SP-TC}
5d	{SP},{AE},{AB},{TC-AN}	{SP-TC},{AE-AB-AN}
5e	{AN},{SN},{PR},{SP},{AB}	{AN-SN-PR},{SP-AB}

	ASM-2 Cluster	ASM-4/6 Cluster
6a	{PR-AB},{AE-SN},{TC-AN}	{AE-TC-PR-AN},{SN-AB}
6b	{SP-AB},{AE-SN},{TC-AN}	{SP-AB}, {AE-SN-TC-AN}
6c	{PR-SN},{AB-SP},{TC-AN}	{AB-SP}, {SN-PR-TC-AN}

Fig. 14. Completion time comparisons of ASM 2, 4 and 6 clusters variants on 128 core processor.



	ASM-4 Cluster	ASM-6 Cluster
4a	{TC-PR},{AE-SN}	{TC-PR-AE-SN}
4b	{TC-AN},{AE-PR}	{TC-AN-AE-PR}
4c	{SN-AB},{TC-AN}	{TC-AN-SN-AB}
4d	{SP-AB},{TC-AE}	{AE-TC-SP-AB}
4e	{AN-SN},{PR-SP}	{AN-SN-PR-SP}
4f	{SP-AB},{TC-AN}	{TC-AN-SP-AB}

	ASM-4 Cluster	ASM-6 Cluster
5a	{PR},{AE-SN-TC-AN}	{AE-TC-AN-SN-PR}
5b	{AB},{AE-SN-TC-AN}	{AE-TC-AN-SN-AB}
5c	{SP},{PR-SN-TC-AN}	{TC-AN-SN-PR-SP}
5d	{SP},{AE-AB-TC-AN}	{AE-TC-AN-SP-AB}
5e	{AN-SN-PR-SP},{AB}	{AN-SN-PR-SP-AB}

	ASM-4 Cluster	ASM-6 Cluster
6a	{PR-AB-AE-SN},{TC-AN}	{AE-TC-AN-SN-PR-AB}
6b	{SP-AB-AE-SN},{TC-AN}	{AE-TC-AN-SN-SP-AB}
6c	{PR-SN-AB-SP},{TC-AN}	{TC-AN-SN-PR-SP-AB}

Fig. 15. Completion time comparisons of ASM 2, 4 and 6 clusters variants on 256 core processor.

mono or dual cluster modes that are supported by all variants of ASM. Both 128 and 256 cores simulated multicores schedule all combinations of 3 co-located processes on 3 isolated clusters and execute them in parallel, which results in similar execution times. To magnify the scalability of ASM on higher core count, 4 or more co-located processes are shown in figures 14 and 15.

Both ASM-4 and ASM-6 cluster variants always outperform ASM-2 cluster configuration on 128 and 256 core processors, due to parallel execution of more processes in multiple clusters. Supporting more clusters results in increased overhead during the mapping phase, where the security kernel performs an iterative search on more number of clusters to find the best mode for each tuple. But this overhead is compensated by the parallel execution of multiple processes. Figure 14 shows that formation of tri and quad clusters for ASM-4 and ASM-6 results in more parallel execution. But more clusters (i.e., penta or hexa) are not possible on 128 cores processor, because there are no 5 or 6 process tuples with cumulative

saturation point ≤128. Therefore, ASM-6 cluster does not show any further improvement over ASM-4 cluster. The overall performance improvement for ASM-4 over ASM-2 on 128 cores is ∼20%.

On increasing the total number of cores to 256, the ASM mappings on figure 15 show that 5 and 6 co-located processes are mapped to 5- and 6- cluster modes. This is possible because the total core count is always higher than the cumulative saturation point of processes for all the tuples. For 256 core processor, ASM-4 and ASM-6 cluster variants observe a performance improvement of ∼20% and ∼30% over ASM-2 cluster, respectively. This shows that ASM architecture is highly scalable for high core count processors, where more clusters can be supported.

8 CONCLUSION

Computation outsourcing has raised concerns regarding the privacy of users' sensitive data. Researchers have explored the use of secure processors, where multiple mutually distrusting processes are co-located for secure computations. MI6 secure processor implements strong isolation of shared hardware resources to protect against timing-based software side-channel attacks. However, it causes performance degradation for co-located processes due to frequent purging overheads and inadequate resource partitioning among processes. Ironhide secure processor resolves the unnecessary purging by improving the overall utilization of core-level resources in a multicore processor. However, processes with high core level parallelism perform sub-optimally on Ironhide due to its resource partitioning. ASM is an adaptive secure multicore architecture with demand-aware execution environment for mutually distrusting processes. ASM ensures privacy by creating single or multiple strongly isolated clusters on multicore processor. ASM shows performance improvement over all prior secure processor architectures.

REFERENCES

[1] [n.d.]. Programming the Tile-gx processor. http://www.mellanox.com/repository/solutions/tile-scm/docs/UG505-Programming-Tilegx-Processor.pdf.

[2] 2009. ARM Security Technology Building a Secure System using TrustZone Technology. ARM white paper.

[3] 2013. FreescaleADL: An Industrial-Strength Architectural Description Language For Programmable Cores. http://opensource.freescale.com/fsl-oss-projects/.

[4] 2017. QUARQ: A Novel General Purpose Multicore Architecture for Cognitive Computing. https://khan.engr.uconn.edu/pubs/quarq-techcon17.pdf.

[5] 2021. Intel® Trust Domain Extensions (Intel® TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html

[6] Masab Ahmad, Farrukh Hijaz, Qingchuan Shi, and Omer Khan. 2015. CRONO: A Benchmark Suite for Multithreaded Graph Algorithms Executing on Futuristic Multicores. In *IISWC*.

[7] Usman Ali and Omer Khan. 2021. 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)*.

[8] Marco Alves, Henrique Freitas, and Philippe Navaux. 2009. Investigation of Shared L2 Cache on Many-Core Processors. 1 – 10.

[9] AMD. 2019. AMD Secure Encrypted Virtualization (SEV). (2019). https://developer.amd.com/sev/

[10] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. 2014. Hardware Trojan Attacks: Threat Analysis and Countermeasures. *Proc. IEEE* 102, 8 (Aug 2014), 1229–1247. https://doi.org/10.1109/JPROC.2014.2334493

[11] Joseph Bonneau and Ilya Mironov. [n.d.]. Cache-Collision Timing Attacks Against AES. In *CHES 2006*, Louis Goubin and Mitsuru Matsui (Eds.).

[12] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, Arvind, and Srinivas Devadas. 2019. MI6: Secure Enclaves in a Speculative Out-of-Order Processor. In *IEEE/ACM MICRO*. 42–56.

[13] Somnath Chakrabarti, Matthew Hoekstra, Dmitrii Kuvaiskii, and Mona Vij. 2019. Scaling Intel® Software Guard Extensions Applications with Intel® SGX Card. In *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy* (Phoenix, AZ, USA) *(HASP '19)*. Association for Computing Machinery, New York, NY, USA, Article Article 6, 9 pages. https://doi.org/10.1145/3337167.3337173

[14] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2018. SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. *arXiv e-prints*, Article arXiv:1802.09085 (Feb 2018), arXiv:1802.09085 pages. arXiv:cs.CR/1802.09085

[15] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX*.

[16] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson (Eds.). 2009. *The Shortest Path Problem*.

[17] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE CVPR*.

[18] Andrew Ferraiuolo, Mark Zhao, Andrew C. Myers, and G. Edward Suh. 2018. HyperFlow: A Processor Architecture for Nonmalleable, Timing-Safe Information Flow Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. ACM, New York, NY, USA, 1583–1600. https://doi.org/10.1145/3243734.3243743

[19] C. W. Fletcher, L. Ren, X. Yu, M. Van Dijk, O. Khan, and S. Devadas. 2014. Suppressing the Oblivious RAM timing channel while making information leakage and program efficiency trade-offs. In *IEEE HPCA*.

[20] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas. [n.d.]. Caches and hash trees for efficient memory integrity verification. In *HPCA'13*.

[21] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *EuroSec*.

[22] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez (Eds.). Springer International Publishing, Cham, 279–299.

[23] Syed Kamran Haider and Marten van Dijk. [n.d.]. Revisiting Definitional Foundations of ORAM for Secure Processors. ([n. d.]). http://arxiv.org/abs/1706.03852.

[24] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[25] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In *IEEE/ACM MICRO*.

[26] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *IEEE S&P*.

[27] Xing Lin and Rajeev Balasubramonian. 2011. Refining the utility metric for utility-based cache partitioning. *Proc. WDDD* (2011).

[28] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *USENIX*.

[29] F. Liu, H. Wu, K. Mai, and R. B. Lee. 2016. Newcache: Secure Cache Architecture Thwarting Cache Side-Channel Attacks. *IEEE Micro* (2016).

[30] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks Are Practical *(IEEE S&P)*. 605–622. https://doi.org/10.1109/SP.2015.43

[31] Tarjei Mandt, Mathew Solnik, and David Wang. 2016. Demystifying the secure enclave processor. *Black Hat Las Vegas* (2016).

[32] Yehia Massoud, Jamil Kawa, Don MacMillen, and Jacob White. 2001. Modeling and Analysis of Differential Signaling for Minimizing Inductive Cross-Talk. In *Proceedings of the 38th Annual Design Automation Conference* (Las Vegas, Nevada, USA) *(DAC '01)*. Association for Computing Machinery, New York, NY, USA, 804–809. https://doi.org/10.1145/378239.379070

[33] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *HASP@ ISCA*. 10.

[34] Jason E. Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *HPCA*. 1–12.

[35] H. Omar, B. D'Agostino, and O. Khan. 2020. OPTIMUS: A Security-Centric Dynamic Hardware Partitioning Scheme for Processors that Prevent Microarchitecture State Attacks. *IEEE Trans. Comput.* (2020), 1–1.

[36] H. Omar, H. Dogan, B. Kahne, and O. Khan. 2018. Multicore Resource Isolation for Deterministic, Resilient and Secure Concurrent Execution of Safety-Critical Applications. *IEEE Computer Architecture Letters* 17, 2 (July 2018), 230–234. https://doi.org/10.1109/LCA.2018.2874216

[37] H. Omar, S. K. Haider, L. Ren, M. van Dijk, and O. Khan. 2018. Breaking the Oblivious-Ram Bandwidth Wall. In *IEEE ICCD*.

[38] H. Omar and O. Khan. 2020. IRONHIDE: A Secure Multicore that Efficiently Mitigates Microarchitecture State Attacks for Interactive Applications. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 111–122.

[39] Moinuddin K. Qureshi. 2019. New Attacks and Defense for Encrypted-address Cache *(ACM ISCA '19)*. 360–371.

[40] Gururaj Saileshwar and Moinuddin K. Qureshi. 2019. CleanupSpec: An "Undo" Approach to Safe Speculation. In *IEEE/ACM MICRO*. 73–86. https://doi.org/10.1145/3352460.3358314

[41] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. 2019. MicroScope: Enabling Microarchitectural Replay Attacks. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 318–331. https://doi.org/10.1145/3307650.3322228

[42] Yao Wang, Andrew Ferraiuolo, and Edward Suh. 2014. Timing Channel Protection for Memory Controllers. In *Proceedings of the International Conference on High-Performance Computer Architecture (HPCA '14)*.

[43] Y. Wang, A. Ferraiuolo, and G. E. Suh. 2014. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 225–236. https://doi.org/10.1109/HPCA.2014.6835934

[44] Y. Wang and G. E. Suh. 2012. Efficient Timing Channel Protection for On-Chip Networks. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. 142–151. https://doi.org/10.1109/NOCS.2012.24

[45] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: A Low Latency and Provably Non-interfering Approach to Secure Networks-on-chip. *SIGARCH Comput. Archit. News* 41, 3 (June 2013), 583–594. https://doi.org/10.1145/2508148.2485972

[46] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown III, and A. Agarwal. 2007. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro* (2007).

[47] Yu Xue, Jiongming Jiang, Binping Zhao, and Tinghuai Ma. 2018. A self-adaptive artificial bee colony algorithm for global optimization. *Soft Computing* (2018).

[48] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas. 2018. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. *MICRO* (2018).

[49] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 719–732. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[50] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher. 2019. Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 954–968. https://doi.org/10.1145/3352460.3358274