OPTIMUS: A Security-Centric Dynamic Hardware Partitioning Scheme for Processors that Prevent Microarchitecture State Attacks

Hamza Omar, Brandon D'Agostino, and Omer Khan

Abstract—Hardware virtualization allows multiple security-critical and ordinary (insecure) processes to co-execute on a processor. These processes temporally share hardware resources and endure numerous security threats on the microarchitecture state. State-of-the-art secure processor architectures, such as MI6 and IRONHIDE enable capabilities to execute security-critical processes in hardware isolated enclaves utilizing the *strong isolation* security primitive. The MI6 processor purges small state resources on each enclave entry/exit and statically partitions the last-level cache and DRAM regions to ensure strong isolation. IRONHIDE takes a spatial approach and creates two isolated clusters of cores in a multicore processor to ensure strong isolation for processes executing in the enclave cluster. Both architectures observe performance degradation due to static partitioning of shared hardware resources. OPTIMUS proposes a *security-centric* dynamic hardware resource partitioning scheme that operates entirely at runtime and ensures strong isolation. It enables *deterministic* resource allocations at the application level granularity, and limits the number of hardware reconfigurations to ensure *bounded* information leakage via the timing and termination channels. The dynamic hardware resource partitioning capability of OPTIMUS is shown to co-optimize performance and security for the MI6 and IRONHIDE architectures.

Index Terms—Secure Processor, Multicore, Strong Isolation, Dynamic Hardware Partitioning, Performance

1 Introduction

Modern microprocessors enable hardware virtualization by means of which multiple security-critical and ordinary processes temporally co-execute on the processor and share hardware resources, such as caches, translation look-aside buffers (TLBs), on-chip networks, and even memory controllers. This hardware sharing results in timing access variations due to *interference* that can by exploited by an attacker process to infer secret data value(s) [1], [2], [3], [4]. To guarantee *non-interference*, various software and hardware based solutions have been proposed. At the software level, process-level isolation (e.g., Intel's SMAP and KASLR) is traditionally adopted across co-executing processes to guarantee memory isolation. However, it falls short as hardware resources still remain shared across temporally executing processes [1]. On the other hand, hardware based solutions broadly fall into two categories: The first category comprises of *non-enclave based* mitigation schemes, where secure and insecure processes temporally co-execute on the processor. The microarchitecture state is protected via scrambled (randomly mapped) address accesses [5], [6], or intrusive hardware extensions are introduced to mitigate unauthorized access to secure data [7], [8], [9]. The second category involves *enclave-based* architectural mechanisms [10], [11], [12], [13], [14], where secure processes execute in containers that are isolated at the hardware-level from temporally executing ordinary processes. Given their continuing commercial integration and strong security guarantees, this paper primarily focuses on *enclave-based* secure processors.

Intel's Software Guard Extensions (SGX) [10] and ARM's TrustZone [12] introduce processor extensions that allow security-critical processes to execute in isolated *enclaves*, while the ordinary (insecure) processes co-execute temporally in the clear. However, SGX and TrustZone fall short of mitigating microarchitecture state attacks [15], [16]. This is primarily due to the absence of *strong isolation* across the secure enclave and ordinary (insecure) processes, which is essential to ensure that secure enclave's data does not leak through the temporally shared hardware resources [17].

To ensure strong isolation, secure processor works [13], [14] have proposed *partitioning-based* mechanisms to protect against microarchitecture state attacks. Indeed, the performance of these works is expected to suffer if the system resources are not partitioned (distributed) proportionally based on the demands of the underlying secure and insecure processes. For instance, the secure MI6 processor [13] adopts the traditional *temporal* execution model and enables strong isolation by statically partitioning the large state structures, such as last-level caches and off-chip memory (DRAM) region(s) across processes. Static partitioning of the shared cache resources adversely impacts performance, as the cache capacity requirements vary based on the demands of a process. Another recently proposed secure processor architecture, IRONHIDE [14] proposes a *spatio-temporal* execution model and enables strong isolation by spatially partitioning the core-level resources of a multicore to construct secure and insecure clusters of cores. It attempts to address the MI6 performance degradation due to static partitioning by dynamically reconfiguring clusters' core-level resources for load balanced execution using a core reallocation heuristic. It uses an offline performance monitoring metric, shared cache misses per kilo-instructions (MPKI) as a function of core counts to decide how many core-level resources allocate to the respective clusters. However, it requires *offline pre-computation* of the MPKI trends for all processes that compose an application, which burdens the underlying security monitor/kernel. Thus, it is imperative to devise an online *security-centric* scheduler that ensures strong isolation for security, while also dynamically partition resources to guarantee high performance.

To dynamically partition hardware resources in an enclave-based secure processor, the resource demands of the processes belonging to an application are continuously moni-

• *H. Omar, B. D'Agostino, and O. Khan are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT USA 06269. E-mail: {hamza.omar, brandon.d'agostino, khan}@uconn.edu. (Corresponding Author: Omer Khan)*

This research was supported by the National Science Foundation under Grant No. CNS-1929261. The Research Experiences for Undergraduates (REU) supplement award supported Brandon D'Agostino.

tored. However, continuous resource scheduling allows an adversary to infer secret information through the timing and termination channels. In essence, employing a continuous resource reconfiguration mechanism violates strong isolation. Prior works [18], [19] have shown that this leakage can be *bounded* by limiting the number of unique reconfiguration events. This paper proposes OPTIMUS, an online *security-centric* resource distribution framework for secure processor architectures that dynamically distribute hardware resources across the underlying processes of an application. It bounds the timing and termination channel leakage by limiting the number of reconfiguration events to *two* for each application invocation.

First, the system is reconfigured to a state where all hardware resources are allocated to the secure kernel (or security monitor of MI6). OPTIMUS samples the MPKI for each process as a function of the resource configurations. Later, these MPKI trends (or profiles) of the secure and insecure processes are utilized to find configurations that balance the hardware resource utilization for performance. The system then undergoes the second reconfiguration with the new computed resource allocations, and the application is allowed to continue execution. Moreover, OPTIMUS always provides the same resource distribution binding for any given application whenever it is scheduled on the system, assuring determinism. During the resource reconfiguration events, OPTIMUS ensures strong isolation, while maximizing hardware resource utilization and system performance. In this work, we show that two state-of-the-art secure processor architectures benefit from OPTIMUS. For MI6, OPTIMUS assures near-optimal distribution of shared cache resources for performance, while keeping strong isolation guarantees intact. It shows performance improvements of 16% compared to MI6 for a set of user and OS level interactive applications. Moreover, OPTIMUS enables a completely *online* mechanism for dynamic resource distribution of core-level resources across the secure and insecure clusters in IRONHIDE. Compared to IRONHIDE without dynamic hardware isolation, OPTIMUS is shown to improve performance by 24%.

2 Enclave-based Secure Processors

The threat model assumptions are adopted from state-of-the-art enclave-based MI6 [13] and IRONHIDE [14] secure processor architectures. All microarchitecture state attacks that rely on covert and side information leakage channels are considered. The operating system (OS) and user level processes are untrusted. However, the processor package (including the main DRAM memory) and a security monitor (or kernel) are trusted. An adversarial process is assumed to be capable of co-locating with a victim process on the shared microarchitecture structures, such as the pipeline buffers, private and shared caches and TLBs, on-chip networks, and memory controller buffers. The adversarial process conducts various state exploits, such as cache timing or access based attacks [1], [2], [5], [15], [20], as well as on-chip network attacks [21]. The adversary is also capable of training hardware speculative execution units, such as branch predictors and store-to-load forwarding logic that is used to leak speculative microarchitecture state of the victim process [3], [4], [9], [16]. Finally, the adversary can monitor resource scheduling events to infer secret information via the timing and termination channels [18], [19].

OPTIMUS caters for the above threats by adopting a novel *security-centric* dynamic hardware resource partitioning framework. It exclusively focuses on software-based microarchitecture state attacks, and excludes adversaries with physical access to the processor. Thus, physical thermal imaging, electromagnetic, and power based channels are not considered in this work. Moreover, physical attacks on memory are prevented using orthogonal mechanisms, such as memory integrity checking [22] and ORAM [23]. Attacks by compromised system software, e.g., OS refusing to allocate process resources are not considered. Lastly, hardware attacks outside the microarchitecture state are orthogonal vectors, such as hardware trojans and DRAM row-hammer attacks.

2.1 Commercial Secure Processor Extensions

Intel's SGX [10] is a recent secure processor extension that allows processes to execute in isolated containers, called *enclaves*. These enclaves are used to execute security-critical processes that temporally co-execute with ordinary (insecure) processes, such as an untrusted OS. For each enclave entry, the processor is switched to *enclave mode*, where the secure process is first attested and authenticated using cryptographic primitives [24]. Upon gaining access, the secure process's data is decrypted for processing in the enclave. On the contrary, for every secure enclave exit, the processor encrypts all enclave related data and flushes the core pipeline. Lastly, the processor is switched to the *normal mode* to execute ordinary processes. Due to temporal execution of the secure enclave with insecure processes, an attacker process can either directly monitor accesses made by the enclave [15], or befuddle the system in making speculative accesses [16] to leak secure enclave's data. Intel's SGX is vulnerable to microarchitecture state attacks as it falls short in providing strong isolation guarantees, which is essential to ensure that secure process's data does not leak through temporally shared hardware resources [17].

2.2 The MI6 Secure Processor Architecture

The MI6 architecture [13] adopts the SGX enclave execution model, and enables *strong isolation* to protect against microarchitecture state attacks. A *security monitor* is deployed that attests and authenticates processes to be executed in an enclave. Similar to SGX, all processes time-share hardware resources. To mitigate sharing across large stateful resources, such as shared last-level cache and DRAM memory, MI6 statically partitions these resources across the secure and insecure processes. Moreover, small stateful hardware structures, such as core pipeline buffers, private caches and TLBs, and memory controller queues are purged on each secure enclave entry and exit.

Figures 1: (a)–(c) shows an illustrative example of processing the secure enclave entry and exit on MI6. Assuming the execution starts with the secure process, the enclave data (after decryption) is read into the time-shared hardware resources and dedicated shared last-level cache sets/slices (c.f. Figure 1:(a)). When finished with its execution, the enclave exit procedure is initiated to switch the secure enclave out of the system. However, to ensure strong isolation, the private pipeline buffers, caches, TLBs, and memory controller

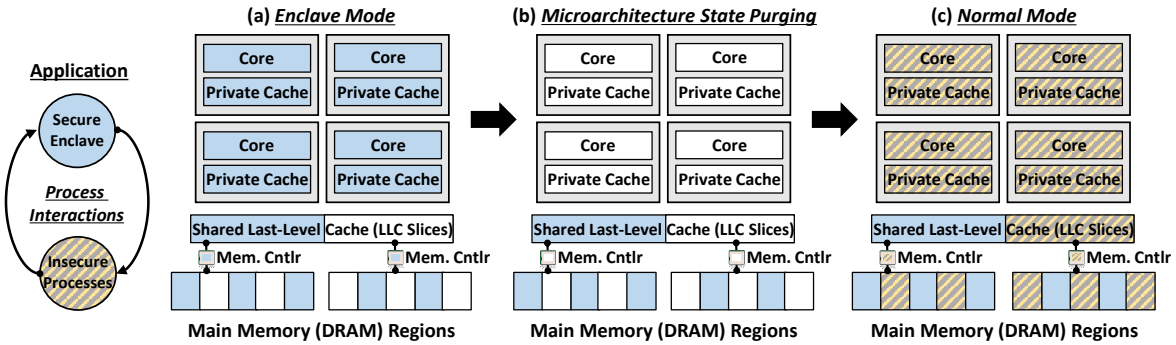

Fig. 1. The execution cycle of an interaction across secure and insecure processes on MI6. For strong isolation, the shared cache and DRAM regions are statically partitioned, and the private resources are purged on every transition between enclave and normal modes.

queues are purged (c.f. Figure 1:(b)) before the processor enters the *normal* mode to execute the ordinary process. Similar to the secure enclave, the ordinary process utilizes its dedicated shared last-level cache sets and DRAM regions along with the pristine (already purged) private resources (c.f. Figure 1:(c)). Lastly, when the ordinary process finishes its execution, the time-shared resources are purged again (similar to Figure 1:(b)) before initiating the enclave entry procedure to execute the secure process.

In MI6, each secure enclave entry/exit requires purging of the private microarchitecture state, leading to performance overheads. The purged state also needs to be brought back into the hardware caches/TLBs, which further exacerbates performance. The static partitioning of last-level cache also impacts performance. Since each application is allocated fixed shared cache set(s)/slice(s), any process in need of more cache capacity experiences degraded data locality.

2.2.1 Security-Centric Dynamic Cache Partitioning

OPTIMUS enables dynamic partitioning of shared cache resources across temporally executing secure and insecure processes of an interactive application. It allows a process to give away or gain shared cache resources to/from the co-located process(es). This essentially caters for the varying shared cache requirements of processes, and overcomes the performance bottlenecks due to imbalance in resource utilization. However, when shared cache resources are reconfigured across processes, the on-chip network routers get shared. Thus, the insecure process can infer secure enclave's secret data by contending on these shared routers.

To regain MI6 strong isolation, OPTIMUS *re-allocates* the process's data structures (memory pages) for all dynamically reallocated shared cache slices. This mechanism unmaps the data structures from their current home (cache slice), by which all dirty data is propagated to the off-chip memory. Lastly, the data structures are re-mapped to the dynamically reconfigured shared cache slice(s). This procedure is performed for every interactive application invocation as a part of the *security-centric design* to ensure bounded leakage [18], [19] (details in Section 3). Moreover, for any given application, the same deterministic shared cache distribution is employed when it is scheduled for execution.

2.3 The IRONHIDE Secure Processor Architecture

IRONHIDE [14] adopts a *spatio-temporal* enclave execution model by spatially isolating core-level resources of a multicore processor to form secure and insecure clusters of

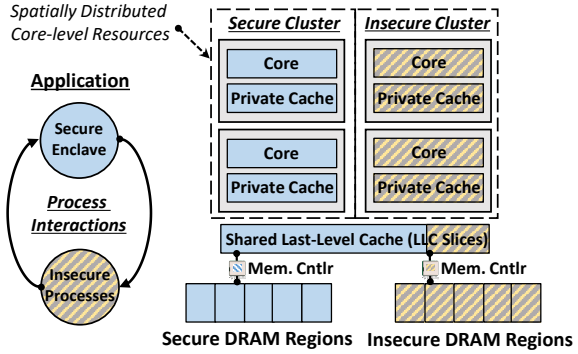

Fig. 2. IRONHIDE architecture and its core-level strong isolation of shared hardware resources. Resources are spatially distributed across clusters.

cores. The application processes temporally execute within their respective clusters, such that the secure process(es) are pinned to the secure cluster where they execute and interact with the processes executing in the insecure cluster. The process interactions across clusters take place without incurring enclave entry/exit purging overheads, thus giving IRONHIDE a significant performance advantage over MI6.

IRONHIDE spatially partitions the shared cache slices and DRAM regions across secure and insecure clusters. Moreover, the per-core private resources, such as pipeline buffers and private caches and TLBs also get spatially distributed across the two clusters. The on-chip networks support a deterministic routing protocol to ensure no packets that originate in one cluster drift to the other cluster. Only network packets intended for application process interactions are allowed to drift from one cluster to the other. Figure 2 shows secure and insecure clusters, where respective secure and insecure processes execute and utilize their dedicated spatially partitioned core-level resources.

Static allocation of core-level resources in IRONHIDE adversely impacts performance, as the resource requirements for any given process vary dynamically [25]. Thus, IRONHIDE implements a core-reallocation heuristic to reconfigure core-level resources across clusters for load-balanced application performance. For each interactive application, it computes the core-level resource distribution by analyzing the per-process last-level cache misses per kilo-instruction (MPKI) trends as a function of varying core counts. However, the heuristic adopts an *offline* approach that requires the MPKI trends to be pre-computed for all secure and insecure processes of an interactive application. This requires IRONHIDE to possess a priori knowledge and storage for

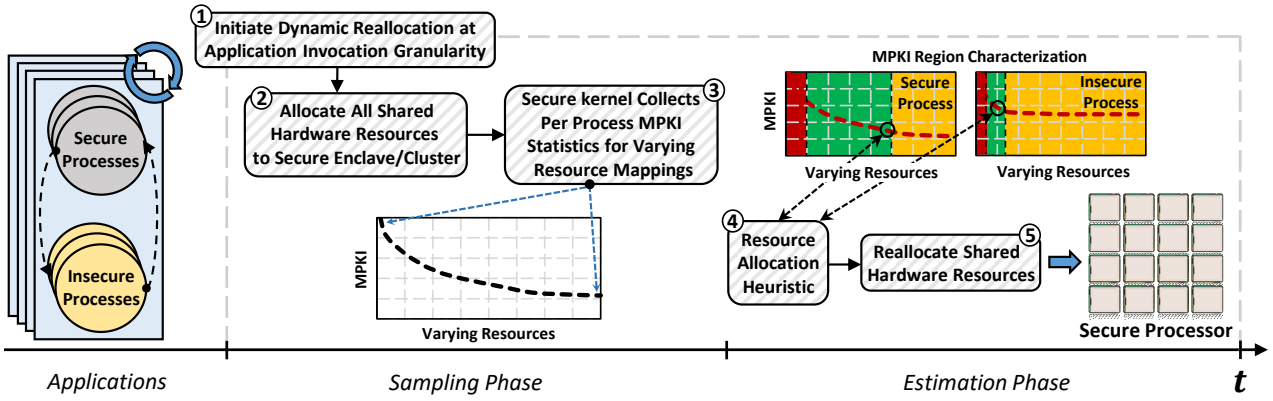

Fig. 3. OPTIMUS security-centric procedure for load-balanced dynamic reallocation of shared hardware resources at application granularity.

the per-process MPKI trends, which burdens the capabilities needed for the security monitor.

2.3.1 Security-Centric Dynamic Hardware Isolation

OPTIMUS enables an *online* capability for IRONHIDE to reconfigure its core-level resources across the clusters at application granularity. However, when core-level resources are dynamically reconfigured across clusters, all re-allocated cores get temporally shared across both clusters. Thus, the insecure process(es) can contend and monitor these hardware resources to leak the state of the secure process(es). To regain strong isolation, IRONHIDE *flushes-and-invalidates* the core pipeline buffers, as well as the private and shared caches (and TLBs) of all re-allocated cores. Moreover, the shared cache/TLB resources are re-allocated (remapped) for the cores given up or gained by the secure cluster (c.f Figure 2).

These core-level resource re-allocations are invoked on every reconfiguration event per interactive application invocation. Similar to MI6, OPTIMUS limits the dynamic core-level resource reconfiguration events to a small bounded factor per each scheduled application (details in Section 3). This method bounds leakage via the timing and terminal channels due to limited number of unique reconfiguration events [18], [19]. Furthermore, OPTIMUS employs a fixed resource binding whenever the same application is scheduled to ensure determinism.

3 Security-Centric Resource Partitioning

The objective of this work is to ensure load-balanced distribution of shared hardware resources at runtime without violating strong isolation guarantees. Ideally, the resource demands of processes must be continuously monitored to decide optimal resource allocations. However, this strategy results in sharing of hardware resources between the secure and insecure processes, resulting in timing and termination channels [18], [19]. An adversary can also perform replay attacks due to non-deterministic resource distribution decisions [26]. The proposed OPTIMUS framework takes a *security-centric* approach by limiting the number of hardware resource reconfigurations to *bound* the information leakage channels. It executes under the trusted secure kernel/monitor, and computes the load-balanced hardware resource allocations at the application invocation granularity. The resource distribution decisions are ensured to be *deterministic* across application invocations, essentially disallowing an adversary to gain useful information by conducting replay attacks. The shared cache misses per-kilo instruction (MPKI) as a function of varying hardware resources is shown to highly correlate with the performance scaling trends of the underlying processes. Therefore, OPTIMUS utilizes runtime classification and characterization of the per-process MPKI information to derive its security-centric hardware resource partitioning.

The overall OPTIMUS framework is shown in Figure 3. Applications executing under a secure processor comprise of secure and insecure processes that frequently interact with each other to assure application progress. Since applications are considered mutually distrusting (see Section 3.1), OPTIMUS computes a bounded and deterministic set of hardware resource allocations on each application invocation (Figure 3:①). The *sampling* phase first reconfigures all hardware resources to the secure kernel (Figure 3:②). The secure kernel varies the hardware resource mappings to obtain an MPKI curve at discrete sampling points for each application process (Figure 3:③). The *estimation* phase performs a resource allocation heuristic using the per-process MPKI information, and computes a deterministic allocation of hardware resources (Figure 3:④). The resource mappings and sampling inputs for each process in an application are kept consistent, ensuring no change in resource allocations across multiple application invocations. OPTIMUS reconfigures the hardware resources with the computed resource bindings, and allows the application to execute on the secure processor (Figure 3:⑤). The hardware reconfigurations are bounded since they are performed once before the sampling, and once at the end of the estimation phase. The deterministic and bounded nature of OPTIMUS disallows an adversary to gain useful information, since the resource distribution decisions are invariant of how an application's requirements vary during its execution.

3.1 Application Execution Model under OPTIMUS

The MI6 secure processor implements a time-multiplexed execution model that can be seamlessly adopted for traditional OS and hypervisor systems. In MI6, multiple secure and insecure processes of an interactive application temporally execute on the processor. When multiple applications are scheduled for execution, they are also considered *mutually distrusting*. MI6 performs a private resource purge operation to ensure strong isolation across each enclave entry/exit.

Moreover, to ensure strong isolation, MI6 statically partitions the shared last-level cache and off-chip memory resources across all application processes. The static partitioning of shared cache resources leads to degraded performance (c.f. Section 2.2). OPTIMUS addresses this performance challenge by enabling a security-centric dynamic partitioning of shared cache resources across the mutually distrusting applications.

Contrarily, IRONHIDE adopts a *spatio-temporal* execution model that is more suitable for secure processors with high core count. It forms two spatially isolated clusters of cores. All *mutually trusting* secure processes of an application temporally execute in the secure cluster, while the insecure processes execute in the insecure cluster. The core-level resources are spatially distributed among the two clusters to ensure strong isolation. However, each cluster must proportionally allocate core-level resources to ensure load-balanced application performance. IRONHIDE also considers multiple secure processes as *mutually distrusting* when they belong to different interactive applications. Thus, the core-level resources are purged on each application context switch. OPTIMUS addresses the performance challenge of load-balanced execution of clusters by enabling a security-centric dynamic partitioning of core-level resources across the mutually distrusting application invocations.

3.2 OPTIMUS Sampling Phase

OPTIMUS first captures the MPKI trends for the application's secure and insecure processes using representative inputs, as shown in Figure 3, steps ② and ③. In MI6, the MPKI trends are computed as a function of allocating different number of shared last-level cache slices (or sets) to the process being profiled. However, in IRONHIDE these trends are computed as a function of sampled core counts. The sampling granularity offers a tradeoff. Fine granularity sampling implies more accurate capture of the MPKI trends, however each sampling point implies additional computation overheads.

For the MI6 architecture, OPTIMUS first reconfigures the processor to a state where all shared cache resources are dedicated to the security monitor (or secure kernel), as shown in Figure 3:②. The security kernel collects the MPKI at this sample point using hardware counters [27] and representative (sample) inputs. It periodically takes away shared cache resources and remaps its data structures (memory pages) to the remaining shared cache slices (Figure 3:③). For all sampled shared cache mappings, OPTIMUS captures the MPKI data points. These sampling steps are repeated for every secure and insecure process of the interactive application. To ensure a bound on resource reconfigurations, OPTIMUS disallows the secure kernel to context switch during the sampling phase. This enables a single reconfiguration of all processor resources to the security kernel before the sampling phase commences.

Consider the *64-point* MPKI trend. OPTIMUS first allocates all shared cache slices to the secure kernel, which maps all process's pages to the 64 cache slices and collects the MPKI data point by executing the sample inputs. Next, one of the cache slices is selected to remap all its mapped pages to be re-mapped to the remaining 63 cache slices. However, all threads of the process remain mapped to the 64 cores. This MPKI data point is captured by executing the sample inputs again. These steps are repeated until the secure kernel collects the last MPKI data point at one mapped shared cache slice. Note, upon switching from one process to the other for capturing their respective MPKI trends, the processor resources are purged. Figure 4 shows the normalized MPKI trend (*64-point*) of a security-critical machine learning process, ALEXNET. Two other variants of the sampled MPKI trends, *4-point* and *6-point* are also shown. The *4-point trend* profiles the process at 64, 48, 16, and 1 shared cache slices, while *6-point trend* is captured at 64, 52, 40, 24, 8, and 1 cache slices. Increasing the number of MPKI data points results in high correlation (accuracy) with *64-point*. However, obtaining more MPKI data points results in higher sampling overheads.

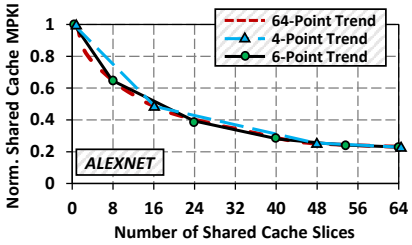

Fig. 4. MPKI trends of the ALEXNET process for various sampling data points under OPTIMUS.

For the IRONHIDE architecture, all discussed operations for MI6 are performed for the *sampling* phase. However, the MPKI data points are captured by first reconfiguring all available core-level resources (core pipeline, private–shared caches and TLBs, and on-chip network routers) to the secure kernel. This implies that all processor core-level resources are allocated to the secure cluster, and the secure kernel executes the sampling phase. For each process, the core-level resources are periodically taken away to collect subsequent MPKI data points. These sampling points not only involve remapping of the process's pages, but the process threads are also re-spawned to map to the newly available cluster's core-level resources. These steps are repeated until the last MPKI data point at one core-level resource is captured.

3.3 OPTIMUS Estimation Phase

After the sampling phase, OPTIMUS analyzes the per-process MPKI trends to find a *single, deterministic* mapping of shared hardware resources, as shown in Figure 3, steps ④ and ⑤. In MI6, this results in the re-allocation of shared cache slices at the per-process granularity of the application being scheduled. However, in IRONHIDE the proportional re-allocation of core-level resources is determined for the respective clusters. The goal of the *estimation* phase is to find a resource allocation binding, such that all shared hardware resources are utilized while the aggregate MPKI is minimum (maximum performance). For a given application, the MPKI trends are expected to remain consistent since all the processes are sampled using pre-determined representative inputs. Furthermore, these MPKI trends are evaluated using a heuristic [14] that leads to *deterministic* resource bindings across multiple invocations of a given application.

The heuristic is first described for the MI6 architecture using an example of the *6-point* MPKI trends for the ALEXNET and VISION processes. As shown in Figure 5, a given MPKI trend comprises of three regions, (1) *non-linear* region (from left to point A), (2) *linear* region (between points A and B),

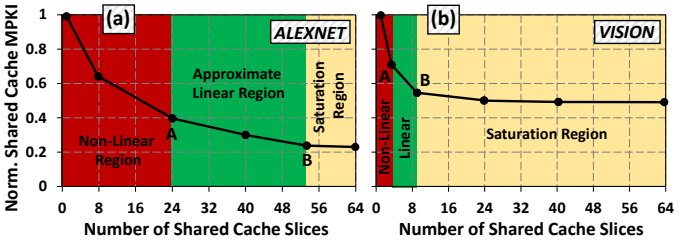

Fig. 5. Classification of MPKI trends for the estimation phase of OPTIMUS. *6-point* MPKI trends for the ALEXNET and VISION processes are shown.

and (3) *saturation* region (from point B to the right). For maximum shared cache utilization (and performance), each application process must operate on the edge of *linear* and *saturation* regions, or beyond. At the minimum, each process must operate in the *linear* region, closer to the saturation point B, making the linear region as the *region of interest*. To retrieve the linear region of an MPKI trend, OPTIMUS first captures the saturation point B by scanning from end of the trend to the point where the absolute slope value becomes greater than 0.1. Then, it captures point A by checking for points from the start of the trend to the point where the slope becomes lesser than 0.5. The *linear* region for an MPKI trend is determined by selecting a linear line between points A and B.

The heuristic computes the ideally required shared cache slices by adding the cache slices acquired for each process at point B. Certainly, the required shared cache slices must not exceed the total available cache slices in the processor. Therefore, a thresholding mechanism is adopted to satisfy the aforementioned constraint. If the required cache slices are equivalent to the total available cache slices, then no resource adjustment is needed and the heuristic terminates by forwarding the computed shared cache distribution for each process at point B. However, if the required cache slices are either less or greater than the total available shared cache slices, then there is a need to perform resource adjustment. In the former case, near-optimal performance is already achieved since all processes are allocated enough resources to operate at their MPKI saturation points. However, to maximize shared cache utilization, OPTIMUS equally distributes the remaining unoccupied cache slices among both processes. Contrarily, when the required shared cache slice count exceeds the total available cache slices, a fixed number of cache slices must be removed from one process and assigned to the other. OPTIMUS proportionally reconfigures (removes or adds) cache slices across processes based on the relative difference between the slope values of each MPKI trend's linear region. This is done while ensuring that fewer cache slices are removed from the process with higher rate of MPKI change (slope value) in the linear region, compared to the process with a smaller slope value. Lastly, the adjusted shared cache distribution is forwarded to the *security monitor* of MI6, where each process's data is remapped before allowing the application to execute.

The slope-based estimation heuristic is applicable to the IRONHIDE architecture in a similar fashion. The MPKI trends obtained from the *sampling* phase are a function of core-level resources for IRONHIDE. Therefore, the resource adjustments are done across the core-level resources for the secure and insecure clusters. The heuristic finds the resource distribution, such that all core-level resources are utilized while assuring minimal aggregate MPKI. The core-level resource binding is forwarded to the secure kernel of IRONHIDE, which performs the cluster reconfiguration before allowing the application to execute.

4 Methodology

A real multicore processor, *Tilera®Tile-Gx72™* [27] is utilized to prototype the MI6 and IRONHIDE architectures, as well their OPTIMUS *security-centric* hardware resource scheduler implementations. Tile-Gx72™ offers hardware support to enable *strong isolation*, i.e., form clusters of cores, manage network traffic across clusters, regulate on-chip and off-chip data access controls, and manage shared cache data placement. It is a tiled multicore architecture comprising of 72 tiles, where each tile consists of a 64-bit multi-issue in-order core, 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 $18MB$). The off-chip memory is accessible using four on-chip 72-bit ECC protected DDR memory controllers attached to independent physical memory channels.

4.1 Secure Processor Modeling on Tile-Gx72™

4.1.1 Modeling MI6 and OPTIMUS

The MI6 secure processor is modeled on *Tilera®Tile-Gx72™* using 64 out of 72 available cores, which are time-shared across the secure enclave and insecure processes. All time-shared cores and their respective L1 caches (and TLBs) are purged on every secure enclave entry and exit. To purge the private L1 cache, a *flush-and-invalidate* procedure reads a dummy buffer of size equal to the cache size into each L1 cache. This essentially removes all secure process's data from the private L1 cache. Then, a memory fence operation (`tmc_mem_fence()` call) is performed to ensure propagation of dirty data to L2 slices. Similarly, the TLBs are flushed using Tilera specific user commands. Each process of the application is provided with statically partitioned L2 slices by overriding the default caching scheme with the *local homing* that maps each process's data on specific L2 slices using `tmc_alloc_set_home(&alloc, core_id)` API call. Moreover, *L2-replication* is disabled to allow only one process to access any given L2 cache slice. For instance, each application comprising of an insecure and a secure process, 32 L2 slices are allocated to each process alongside half of the (statically partitioned) DRAM regions. The memory controllers are also purged on every secure enclave entry/exit, which is done using `tmc_mem_fence_node(controller_id)` call that writes back all modified data to the DRAM.

When an application is deployed on the system, OPTIMUS halts the application and initiates its *sampling* phase, where all processor resources (including the shared L2 slices) are first *re-allocated* to the secure kernel. To construct the MPKI trend, the secure kernel systematically allocates L2 cache slices to each process being profiled. To re-map data structures (pages) to certain L2 cache slices, the pages are first un-mapped from their current L2 home cache slices using `tmc_alloc_unmap (*addr, size)` API call, followed by setting the new home for each page

using `tmc_alloc_set_home (&alloc, core_id)`. Finally, each page is mapped to the new L2 home cache slice using `tmc_alloc_remap (&alloc, size, new_size)` call. This re-map procedure is performed by varying the L2 cache allocations to construct the desired MPKI trends of each application process. Note, the prototype only contains private TLBs, thus only shared L2 cache slices are re-mapped. Once the *sampling* phase completes, the secure kernel performs the *estimation* phase, where it executes the resource allocation heuristic and deploys the computed allocation of L2 cache slices for the application processes. Although the MPKI data points can be sampled at any arbitrary granularity of shared cache partitions, the *Tile-Gx72*^{TM} limits this capability to a per-core L2 cache slice granularity. Once these deterministic, bounded reconfigurations are performed, OPTIMUS allows the application to continue its execution.

4.1.2 Modeling IRONHIDE and OPTIMUS

To model IRONHIDE on *Tilera*®*Tile-Gx72*^{TM}, the secure and insecure clusters of cores are formed by pinning process's threads to respective cores via `tmc_cpus_set_my_cpu(tid)` call. The L2 cache slices are allocated to their respective cluster using the *local homing* scheme. The clusters' accesses to their respective DRAM regions are realized by forwarding their respective L2 cache miss traffic to dedicated memory controllers via `tmc_alloc_set_nodes_interleaved (&alloc, pos)`. 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 the secure cluster, whereas `pos = 0b1100` (MC_2 and MC_3) for the insecure 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.

The procedure similar to MI6 is followed when OPTIMUS is integrated with IRONHIDE. However, the allocation of core-level resources during the *sampling* phase involves not only L2 cache re-mappings but also re-spawning of the process threads and mapping them to the varying number of cores to construct the MPKI trends. Moreover, OPTIMUS performs the private L1 cache and TLB *flush-and-invalidate* mechanism for the re-allocated cores at the end of the *estimation* phase to ensure strong isolation.

4.2 Benchmarks & Execution Settings

Three different classes of user-level interactive applications and two different classes of OS-level interactive applications are evaluated in this work.

4.2.1 User-Level Interactive Applications

- Real-time Graph Processing: This interactive application uses safety-critical graph algorithms to perform decision analytics on the graph input generated by an insecure graph generation algorithm [28] (GRAPH). The insecure GRAPH generation process reads values at various time intervals from distributed sensors, and generates temporal graph inputs for California road network [29]. Three secure graph algorithms [30] are considered, i.e., Single Source Shortest Path (SSSP), PageRank (PR), and Triangle Counting (TC).
- Real-Time Perception and Mission Planning: This interactive application deploys an insecure vision pipeline [31] (VISION) that processes RAW images, and consequently feeds them to several secure perception and mission planning algorithms. The mission planning Artificial Bee Colony [32] (ABC) algorithm is adopted from advanced driver-assistance system with inputs from a real-world road scenario. Moreover, two perception algorithms [33], ALEXNET and SqueezeNet (SQZNET) process inputs communicated via the VISION pipeline.
- Query Encryption: This interactive application uses a secure encryption algorithm from Advanced Encryption Standard (AES) to encrypt database queries periodically generated by an insecure QUERY generation algorithm [34].

Each user-level interactive application is executed with 500, $1K$, $5K$, $10K$, and $50K$ inputs, and the reported completion time is the average across these runs.

4.2.2 OS-Level Interactive Applications

A set of interactive applications are considered that require frequent support from an untrusted OS for generating and processing requests, such as *fread*, *fcntl*, *close*, and *writev* [35].

- Web Servers: Three web server applications: LIGHTTPD [36] (version 1.4.41), NGINX [37], and APACHE [38] (version 2.4.18) are considered. LIGHTTPD fetches 1 million pages (each of $20KB$ size) through 100 concurrent client connections via the *http_load* tool, whereas, the remaining two web server applications use the ApacheBench tool to download 2 million web pages.
- Databases: MEMCACHED [39] (version 1.4.31) processes 2 million requests generated via the *memtier benchmark*. A SET:GET ratio of 1:1 is employed.

Each user and OS level interactive application is first invoked under the OPTIMUS security-centric resource partitioning *sampling* and *estimation* phases, followed by their execution. For all considered applications, the interactions across secure and insecure processes are carried out via the shared inter-process communication buffer. In case of user-level interactive applications, the secure process interacts with the insecure process at an interactivity rate of $\sim$400 secure process entry/exit events per second. However, the average interactivity rate for OS interactive applications is measured as $\sim 220K$ secure process entry/exit events per second, similar to the rate observed in HotCalls [35]. The *purging* overheads for MI6 at every enclave entry/exit are included in the completion time. The overheads associated with *sampling* and *estimation* phases of OPTIMUS are also added to the application's completion time breakdown.

5 Evaluation

5.1 OPTIMUS and the MI6 Architecture

Figure 6:(a) shows the completion time comparison of MI6 and OPTIMUS. The *online* OPTIMUS is evaluated using the *64-point* MPKI trend generated in the *sampling* phase. The prototype processor exposes 64 shared L2 cache slices for reconfiguration. Hence *64-point* MPKI trend represents a fine grain configuration that enables highest accuracy, while incurring the largest sampling overhead. This exposes an overhead and accuracy tradeoff that is empirically evaluated to reveal the *64-point* MPKI trend as default (c.f. Figure 7). An *offline* OPTIMUS scheme is also evaluated, where the

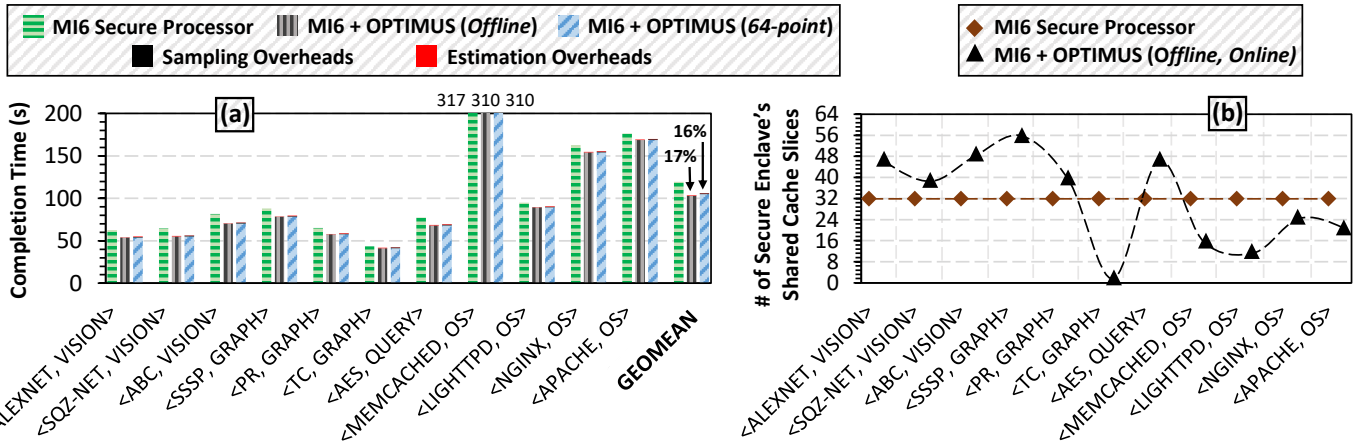

Fig. 6. (a) Per-application completion time for MI6 and various OPTIMUS configurations. (b) Dynamic allocation of shared cache resources for MI6 and OPTIMUS.

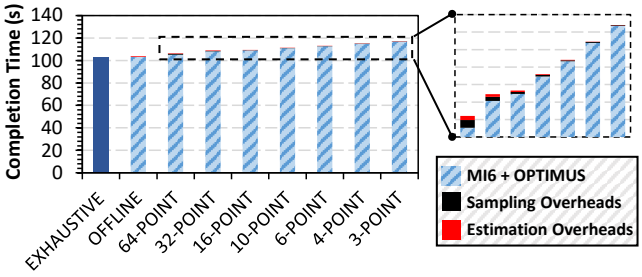

Fig. 7. Completion time analysis of various online and offline configurations for MI6 with OPTIMUS.

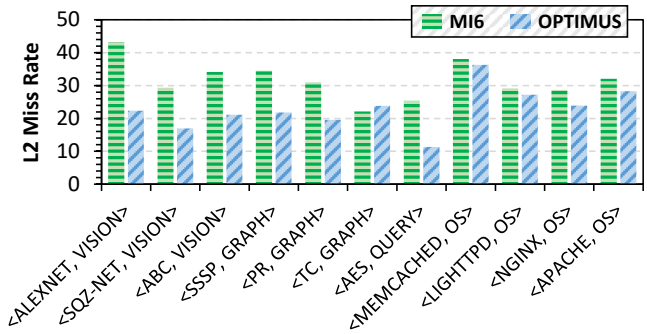

Fig. 8. Per-application shared L2 cache miss rates for MI6 and *64-point* OPTIMUS.

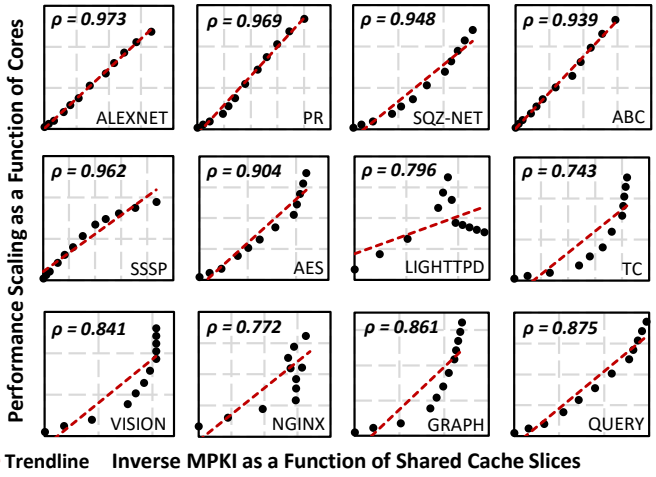

Fig. 9. The scatter plot represents the Pearson's correlation (ρ) between inverse shared cache MPKI and performance scaling at a given cache slice and core count tuple. Higher the coefficient, higher is the correlation.

MPKI trends for all application processes are pre-computed. Therefore, no *sampling* phase is performed for *offline* OPTIMUS. The *64-point* OPTIMUS improves the geometric completion time by ∼16% over the baseline MI6 secure processor. On one hand, it incurs an average performance overhead of <2% for performing the dynamic L2 cache reconfiguration using the *sampling* and *estimation* phases. On the other hand, dynamic L2 cache allocations match the demands of the underlying application processes and result in significant data access improvements. Overall, the performance gains from improved data locality significantly outweigh the overheads of OPTIMUS. The *offline* OPTIMUS scheme filters the *sampling* phase from the overall OPTIMUS overheads. It is shown to improve performance by ∼17% over the MI6 baseline, which is 1% better compared to the *64-point* OPTIMUS. This shows that the contributions of the *sampling* phase using fine grain *64-point* MPKI trends do not overwhelm OPTIMUS. However, pre-computations for the MPKI trends result in storage burden for the security monitor, which is undesirable. Overall, from Figure 6:(a), it is clear that static partitioning of the shared L2 cache in the baseline MI6 adversely impacts the cache utilization across application processes. To visualize the distribution of shared cache resources, Figure 6:(b) shows the number of L2 cache slices that are given up (below 32), or gained (above 32) by the security monitor for both baseline MI6 and OPTIMUS optimized MI6. Note, a single marker is shown for OPTIMUS *offline* and *online* schemes as they both result in exactly similar resource distribution decisions. OPTIMUS improves performance by load-balancing the shared L2 cache capacity across application processes by re-allocating L2 cache slices at the application level granularity.

To further investigate the *online* OPTIMUS capabilities and tradeoffs, Figure 7 shows the performance comparisons of different *estimation* phase heuristic variants, i.e., *exhaustive search*, and OPTIMUS with *offline* and *online*. Moreover, the *sampling* phase tradeoffs are evaluated by comparing *online* MPKI trends from *64-point* to a coarse-grain *3-point* sampling granularity. The *exhaustive search* reports the best shared L2 cache allocations at the application granularity, and also excludes the computation overheads of the search. Thus, it

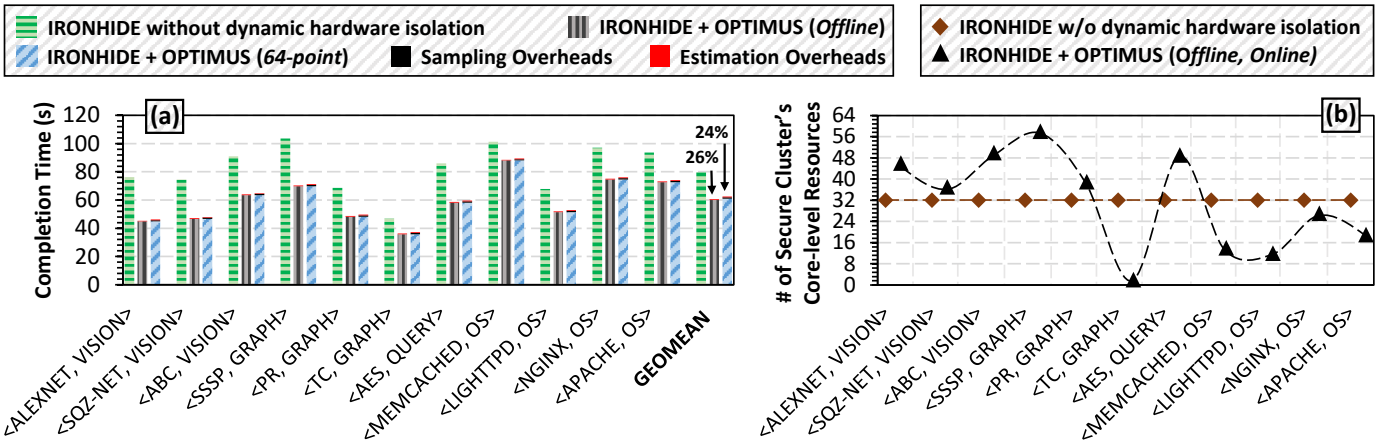

Fig. 10. (a) Per-application completion time for IRONHIDE and various OPTIMUS configurations. (b) Dynamic allocation of clusters' core-level resources for IRONHIDE and OPTIMUS.

serves as a baseline to understand the efficacy of various overheads from the *sampling* and *estimation* phases of OPTIMUS. The *offline* OPTIMUS configuration performs ∼2% worse in terms of completion time when compared to the *exhaustive* scheme. Relatively, the default *64-point* online OPTIMUS scheme is shown to degrade performance by ∼1% compared to the *offline* scheme. However, it does not require any pre-computations, and computes the load-balanced shared L2 cache distributions with negligible runtime overheads.

Figure 7 also compares the completion time breakdowns for various MPKI sampling points. As the *sampling* phase captures lesser number of MPKI data points, its overheads reduce. However, the overall completion time increases as sampling granularity is varied from *64-point* to *3-point*. In fact, the performance degrades from ∼3% to ∼16% when the *3-point* variant is considered rather than deploying the *64-point* variant compared to the *exhaustive* scheme. As the accuracy of capturing the MPKI trends reduces, the *estimation* heuristic end up making sub-optimal shared L2 cache allocation decisions. OPTIMUS favors accurate sampling of MPKI trends by utilizing the *64-point* as its default scheme to optimize the performance and accuracy tradeoff.

The data locality improvements from OPTIMUS are evaluated by investigating the shared L2 cache miss rate behaviors in Figure 8. MI6 with OPTIMUS advocates in reducing the L2 cache miss rates by up to 2× compared to the baseline MI6. The <TC, GRAPH> application acts as an outlier, where the MI6 static allocation of equally distributing L2 cache slices results in a slightly better L2 miss rates compared to OPTIMUS. The TC process does not show much L2 cache locality as it traverses the graph once to compute the number of triangles passing through each vertex. Thus, it allocates only two L2 cache slices under OPTIMUS, as shown in Figure 6:(b). The remaining L2 cache slices are allocated to the GRAPH process, which brings insignificant improvements in miss rates due to its small working set. Even with slightly worse L2 cache miss rates, the <TC, GRAPH> application ends up with similar completion times for both MI6 and OPTIMUS, as seen in Figure 6:(a).

Why utilize the MPKI trends as a metric to capture the performance trends of an application? Figure 9 shows the pearson's correlation coefficient (ρ) [40] between the normalized shared cache MPKI and performance scaling trends of various application processes. These performance scaling trends are obtained as a function of core counts; whereas, the MPKI trends are obtained as a function of shared L2 cache slices for MI6. A geometric mean correlation value of $\rho = 0.901$ is observed across all considered secure and insecure processes. This high correlation shows that MPKI is indeed a strong metric for computing load-balanced shared cache allocations under OPTIMUS. Certain processes either have a small working set (i.e., VISION, GRAPH, and QUERY), or do not exhibit significant shared L2 cache locality (i.e., TC, LIGHTTPD, and NIGINX). For these six processes, the inverse shared cache MPKI trends are observed to decrease as more L2 cache slices are allocated to them. However, their performance continues to scale due to high core-level parallelism. The performance scaling exceptions are LIGHTTPD and NGINX that do not scale beyond 16 and 24 threads respectively. For these reasons, the pearson's correlation coefficient is relatively lower for these six processes compared to other processes that exhibit near perfect correlations.

5.2 OPTIMUS and the IRONHIDE Architecture

Figure 10:(a) shows the completion time comparison of IRONHIDE against OPTIMUS. The baseline IRONHIDE is configured without the support for *dynamic hardware isolation*, where core-level resources (core pipeline, caches, TLBs and network routers) are statically distributed proportionally among the secure and insecure clusters. However, the *offline* OPTIMUS implements the default IRONHIDE scheme [14] that enables dynamic reconfiguration. Here, OPTIMUS avoids the *sampling* phase by pre-computing the MPKI trends of all application processes. The *estimation* phase heuristic computations incur runtime overheads to determine the allocation of core-level cluster resources. The *online* OPTIMUS is evaluated using the *64-point* MPKI trends generated in the *sampling* phase. The *64-point* MPKI trend represents a fine grain configuration that enables highest accuracy, while incurring the largest sampling overhead. It improves the geometric mean completion time by ∼24% over the baseline IRONHIDE that is constrained by its static allocation of cluster resources. On one hand, it incurs an average performance overhead of <2% for performing the dynamic core-level resource configuration using the *sampling* and *estimation* phases. On the other hand, dynamic core-level

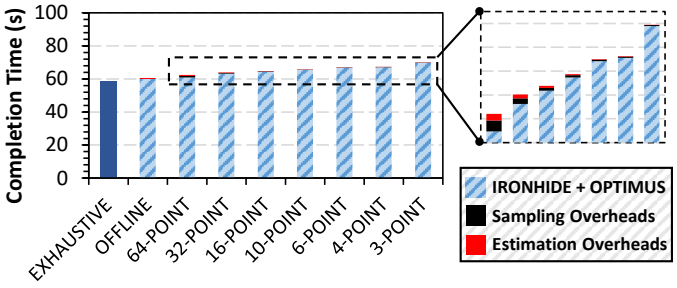

Fig. 11. Completion time analysis of various online and offline configurations for IRONHIDE with OPTIMUS.

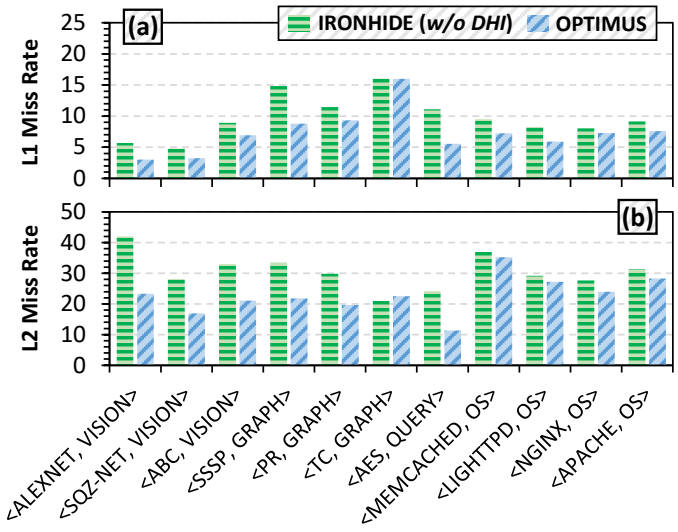

Fig. 12. Per-application (a) L1 cache and (b) L2 cache miss rates for IRONHIDE (*w/o dynamic hardware isolation*) and *64-point* OPTIMUS.

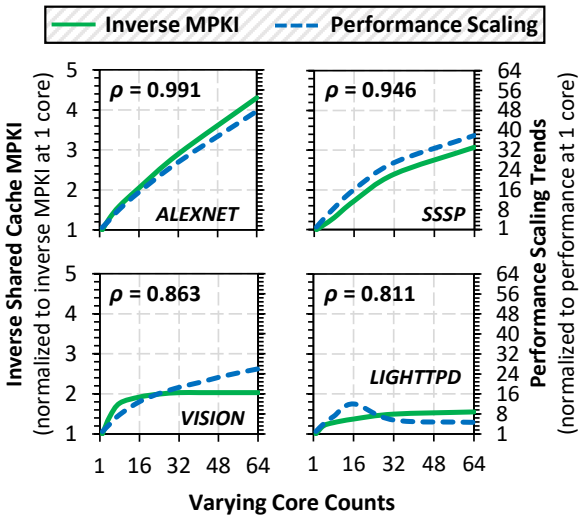

Fig. 13. Normalized shared cache MPKI and performance scaling trends as a function of varying core counts.

allocations match the demands of the underlying application processes and result in improved parallelism. Overall, these performance gains from dynamic resource allocations outweigh the overheads of OPTIMUS. The *offline* OPTIMUS scheme burdens the security kernel to pre-compute and store the MPKI trends of all processes. However, it benefits from avoiding the overheads of the *sampling* phase. It is shown to improve performance by ∼26% over the baseline IRONHIDE, which is 2% better compared to the *64-point* OPTIMUS. This shows that the contributions of the *sampling* phase using fine grain 64-point MPKI trends do not overwhelm OPTIMUS. Overall, from Figure 10:(a), it is clear that static partitioning of the core-level resources across the IRONHIDE clusters of cores adversely impacts core-level parallelism across application processes. To visualize the distribution of core-level resources, Figure 10:(b) shows the number of cores given up (below 32), or gained (above 32) by the security monitor. OPTIMUS improves performance by load-balancing the allocation of core-level resources to the two IRONHIDE clusters at the application level granularity.

To further investigate the online OPTIMUS capabilities and tradeoffs, Figure 11 shows the performance comparisons of different estimation phase heuristic variants, i.e., *exhaustive search*, and OPTIMUS with *offline* and *online*. Moreover, the *sampling* phase tradeoffs are evaluated by comparing online MPKI trends from *64-point* to a coarse-grain *3-point* sampling granularity. The *exhaustive search* reports the best core-level cluster resource allocations at the application granularity, and also excludes the computation overheads of the search. Thus, it serves as a baseline to understand the efficacy of various overheads from the *sampling* and *estimation* phases of OPTIMUS. When compared to the *exhaustive* scheme, the *offline* and *64-point online* OPTIMUS configurations perform ∼4% and ∼6% worse, respectively. The *offline* OPTIMUS configuration performs ∼4% worse in terms of completion time when compared to the *exhaustive* scheme. Relatively, the default *64-point online* OPTIMUS scheme is shown to degrade performance by ∼6%, compared to the *offline* scheme. However, the *64-point* scheme does not require any pre-computations, and computes the load-balanced core-level resource distributions with negligible runtime overheads.

Figure 11 also compares the completion time breakdowns for various MPKI sampling points. As the *sampling* phase captures lesser number of MPKI data points, its overheads reduce. However, the overall completion time increases as sampling granularity is varied from *64-point* to *3-point*. In fact, the performance degrades from ∼6% to ∼17.2% when the *3-point* variant is considered rather than deploying the *64-point* variant compared to the *exhaustive* scheme. As the accuracy of capturing the MPKI trends reduces, the estimation heuristic results in making sub-optimal allocation of cores per cluster. OPTIMUS favors accurate sampling of MPKI trends by utilizing the *64-point* as its default scheme to optimize the performance and accuracy tradeoff.

The core-level locality improvements from OPTIMUS are evaluated by investigating the private L1 and shared L2 cache miss rate behaviors in Figure 12. IRONHIDE with OPTIMUS dynamically adjusts the core-level resources to load-balance the cache resource utilization at the application level granularity. These benefits are greatest when the cache locality variations are imbalanced across the two clusters, as seen with <ALEXNET, VISION> application. However, when application processes have low data locality and/or working set size (e.g., <TC, GRAPH>), the cache behaviors do not improve over static allocation of resources.

Figure 13 shows the normalized inverse shared cache MPKI trends (left y-axis) and normalized performance scaling variations (right y-axis) as a function of core counts

(x-axis) for a set of representative processes. The individual pearson's correlation coefficients (ρ) [40] are also reported. Similar to MI6, high correlation is observed between the inverse MPKI trends and performance scaling variations. However, VISION and LIGHTTPD processes show relatively lower pearson correlation. VISION has a small working set that easily fits in eight shared L2 cache slices. However, it exhibits high core-level parallelism and keeps scaling beyond 8 cores. LIGHTTPD exhibits low shared cache locality due to its random request generation. Moreover, it does not show performance scaling beyond a small number of cores. Overall, the geometric mean correlation coefficient value of 0.9 is observed for all processes, which shows that MPKI trends are a good metric to predict the allocation of core-level resources in IRONHIDE.

6 Related Work

Academic works, Bastion [41] and Iso-X [42] reduce the trusted computing base (TCB) to a secure processor chip. Bastion relies on a trusted hypervisor to ensure confidentiality and integrity for security-critical software modules, whereas Iso-X provides fine-grain isolation at the memory-page level and enables flexible allocation of memory to trusted and untrusted software modules. Industry has developed secure processor architectures to secure arbitrary computations, such as Intel's SGX [10], AMD's Secure Encrypted Virtualization (SEV) [11], and ARM's Trustzone [12]. AMD's SEV represents a virtualization security paradigm, where it integrates main memory encryption capabilities with the existing virtualization technologies to support encrypted virtual machines. The trusted-execution environment (TEE) of ARM's TrustZone, called the *secure world*, provides protection for trusted hardware and software resources. Theses resources are enforced to be inaccessible to the untrusted OS, or *normal world*, via hardware-based mechanisms. Intel's SGX enables on-chip enclaves that isolate processes from the untrusted OS via key management and memory address partitioning. However, SGX has been shown to be vulnerable against cache-timing and control flow speculation attacks [15], [16]. Sanctum [43] combines minimal hardware modifications with a trusted software component to offer an isolated and trusted execution environment similar to Intel's SGX.

Recent secure processor works [13], [14], [44] extend the idea of enclaves to alleviate microarchitecture state attacks. For instance, DAWG [44] utilizes protection (or security) domains to isolate secure data from malicious insecure applications. MI6 [13] extends Sanctum by introducing the concept of *strong isolation*, which requires purging of the microarchitecture state of time-shared resources at every secure enclave entry/exit, and static partitioning of the last-level cache and DRAM regions across processes. IRONHIDE architecture [14] spatially partitions system resources to form strongly isolated secure and insecure core clusters, where processes temporally execute within their respective clusters. It pins the secure process(es) to the secure cluster, and incurs no secure enclave entry/exit purging overheads. The main contribution of the proposed OPTIMUS scheme is to mitigate the performance overheads of the MI6 and IRONHIDE architectures by introducing a *security-centric* dynamic partitioning of shared hardware resources. OPTIMUS bounds information leakage by limiting the resource reconfiguration events to a deterministic controllable factor for each application invocation.

7 Conclusion

State-of-the-art MI6 secure processor architecture employs the idea of strong isolation to prevent microarchitecture state vulnerabilities. However, it suffers from performance degradation due to static partitioning of shared last-level cache across the secure enclave and insecure processes, and microarchitecture state purging of private resources on every secure enclave entry and exit. The IRONHIDE secure processor addresses the performance shortcomings of MI6 by forming two spatially isolated clusters of cores, where these clusters are load-balanced for performance via a core reallocation heuristic. However, the heuristic requires *a priori* knowledge (pre-computation) of the underlying processes to proportionally distribute resources across clusters, which essentially burdens the security kernel of IRONHIDE. This paper proposes OPTIMUS, a novel *security-centric* resource partitioning scheme that reconfigures hardware resources dynamically (without requiring pre-computations), while keeping strong isolation guarantees of MI6 and IRONHIDE architectures intact. OPTIMUS is prototyped on a real 72-core multicore processor. For a set of user and OS-level interactive applications, it is shown to improve the performance for both MI6 and IRONHIDE architectures. Certainly, the performance implications of assuring strong isolation using OPTIMUS are expected to vary with different core types, cache-hierarchy setups, and on-chip network capabilities. Evaluating OPTIMUS for different processor architectures targeting specific market segments is left as part of the future work.

References

[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*, pp. 605–622, 2015.

[2] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*, CHES06, (Berlin, Heidelberg), p. 201215, Springer-Verlag, 2006.

[3] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *meltdownattack.com*, 2018.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *USENIX Security Symposium)*, pp. 973–990, Aug. 2018.

[5] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA 19, (New York, NY, USA), p. 360371, Association for Computing Machinery, 2019.

[6] F. Liu, H. Wu, K. Mai, and R. B. Lee, "Newcache: Secure cache architecture thwarting cache side-channel attacks," *IEEE Micro*, vol. 36, p. 816, Sept. 2016.

[7] G. Saileshwar and M. K. Qureshi, "Cleanupspec: An undo approach to safe speculation," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 52, (New York, NY, USA), p. 7386, Association for Computing Machinery, 2019.

[8] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," in *IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, p. 428441, IEEE Press, 2018.

[9] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative taint tracking (stt): A comprehensive protection for speculatively accessed data," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 52, (New York, NY, USA), p. 954968, Association for Computing Machinery, 2019.

[10] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution.," in *HASP@ ISCA*, p. 10, 2013.

[11] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption.," AMD white paper, 2016.

[12] "Arm security technology building a secure system using trustzone technology." ARM white paper, 2009.

[13] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "Mi6: Secure enclaves in a speculative out-of-order processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 52, (New York, NY, USA), p. 4256, Association for Computing Machinery, 2019.

[14] 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.

[15] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, EuroSec17, (New York, NY, USA), Association for Computing Machinery, 2017.

[16] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 142–157, 2019.

[17] A. Ferraiuolo, M. Zhao, A. C. Myers, and G. E. Suh, "Hyperflow: A processor architecture for nonmalleable, timing-safe information flow security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), pp. 1583–1600, ACM, 2018.

[18] C. W. Fletchery, L. Ren, X. Yu, M. Van Dijk, O. Khan, and S. Devadas, "Suppressing the oblivious ram timing channel while making information leakage and program efficiency trade-offs," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 213–224, 2014.

[19] S. K. Haider, O. Khan, and M. van Dijk, "Revisiting definitional foundations of oblivious RAM for secure processor implementations," *CoRR*, vol. abs/1706.03852, 2017.

[20] T. M. John, S. K. Haider, H. Omar, and M. Van Dijk, "Connecting the dots: Privacy leakage via write-access patterns to the main memory," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018.

[21] 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.

[22] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pp. 295–306, 2003.

[23] H. Omar, S. K. Haider, L. Ren, M. van Dijk, and O. Khan, "Breaking the oblivious-ram bandwidth wall," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 115–122, 2018.

[24] V. Costan and S. Devadas, "Intel sgx explained." Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.

[25] H. Omar, H. Dogan, B. Kahne, and O. Khan, "Multicore resource isolation for deterministic, resilient and secure concurrent execution of safety-critical applications," *IEEE Computer Architecture Letters*, vol. 17, pp. 230–234, July 2018.

[26] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA 19, (New York, NY, USA), p. 318331, Association for Computing Machinery, 2019.

[27] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, p. 1531, Sept. 2007.

[28] U. Demiryurek, B. Pan, F. Banaei-Kashani, and C. Shahabi, "Towards modeling the traffic data on road networks," in *Proceedings of the Second International Workshop on Computational Transportation Science*, IWCTS 09, (New York, NY, USA), p. 1318, Association for Computing Machinery, 2009.

[29] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, eds., *The Shortest Path Problem*, DIMACS workshop, 2009.

[30] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *2015 IEEE International Symposium on Workload Characterization*, pp. 44–55, 2015.

[31] M. Buckler, S. Jayasuriya, and A. Sampson, "Reconfiguring the imaging pipeline for computer vision," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 975–984, 2017.

[32] Y. Xue, J. Jiang, B. Zhao, and T. Ma, "A self-adaptive artificial bee colony algorithm for global optimization," *Soft Computing*, 2018.

[33] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[34] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC 10, (New York, NY, USA), p. 143154, Association for Computing Machinery, 2010.

[35] O. Weisse, V. Bertacco, and T. Austin, "Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves," vol. 45, p. 8193, Association for Computing Machinery, June 2017.

[36] Jan, "Lighttpd: An open-source web server optimized for speed-critical environments," https://www.lighttpd.net/.

[37] "Nginx web-server application," https://www.nginx.com/.

[38] "Apache http server project," https://httpd.apache.org/.

[39] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, pp. 53–64, June 2012.

[40] W. Kirch, ed., *Pearson's Correlation Coefficient*, pp. 1090–1091. Dordrecht: Springer Netherlands, 2008.

[41] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in *IEEE International Symposium on High-Performance Computer Architecture*, pp. 1–12, 2010.

[42] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Iso-x: A flexible architecture for hardware-managed isolated execution," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 190–202, 2014.

[43] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX*, 2016.

[44] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 974–987, 2018.

Hamza Omar is a Ph.D. student in the department of Electrical & Computer Engineering at the University of Connecticut (UConn). His research interests broadly cover computer architecture and architectural support for security and resiliency in multicore processors. Hamza interned at NXP Semiconductors during Summer 2018. Prior to joining UConn in 2015, he did his Bachelors in Electrical Engineering from University of Engineering & Technology, Lahore, Pakistan.

Brandon D'Agostino is a senior undergraduate student in the department of Electrical & Computer Engineering at the University of Connecticut. He worked on this project as a National Science Foundation Research Experiences for Undergraduates student. His research interests include computer security, reconfigurable hardware, and artificial intelligence. He develops high-end audio products in his father's company, and offers contract-based embedded system services.

Omer Khan is the Castleman Associate Professor in the Department of Electrical & Computer Engineering at the University of Connecticut. Prior to joining UConn, he was a Postdoctoral Research Scientist at the Massachusetts Institute of Technology. His research interests include developing cross-layer methods to improve the performance scalability and security of multicore processor architectures. Khan received a PhD in Electrical and Computer Engineering from the University of Massachusetts Amherst. He is a senior member of IEEE and a member of ACM.