Interposer-Based Root of Trust

Tapojyoti Mandal
Texas A&M University
tapojyoti.mandal@tamu.edu

Gino Chacon
Texas A&M University
ginochacon@tamu.edu

Johann Knechtel
New York University, Abu Dhabi
johann@nyu.edu

Ozgur Sinanoglu
New York University, Abu Dhabi
ozgursin@nyu.edu

Paul V. Gratz
Texas A&M University
pgratz@gratz1.com

Vassos Soteriou
Cyprus University of Technology
NA

ABSTRACT

Industry is moving towards large-scale system-on-chip (SoC) designs where heterogeneous components such as processor cores, DSPs, memory controllers, and accelerator units are bundled via 2.5D integration. That is, these components are fabricated separately onto chiplets and then integrated using an interconnect carrier, a so-called interposer. Independently, however, general-purpose SoC architectures have raised significant security concerns. Therefore, with many IP modules and hardware components coming from various third-party vendors and manufacturers, ensuring security and integrity of chiplets-based system is a grand challenge. Further, malicious software running within a chiplet can pose significant risks as well. In this work, we propose to leverage an active interposer as secure-by-construction, generic root of trust platform for such modern systems. Our work presents a new architectural framework where untrusted processing elements, running untrusted code, are integrated on top of such an interposer-based root of trust, allowing us to detect and prevent any form of malicious messages exchanged between the heterogeneous components. Our technique has limited design overhead that is furthermore restricted to the active interposer, allowing the heterogeneous components within chiplets to remain untouched. We show that our scheme correctly handles attempted security violations with little impact on system performance, around 4%.

1. INTRODUCTION

Given a sundry of attacks targeting the hardware and/or software of modern computing systems, security has become a significant focus for processor and system design. The possible attack surface is very large in modern computing systems, e.g., ranging from cache side-channel attacks that can leak information between running processes [1], over Spectre and Meltdown [2, 3] and similar attacks that leverage security vulnerabilities in microarchitectures for speculation [4, 5, 6, 7, 8, 9], to hardware Trojans that can outright violate the system security model [10, 11, 12]. Given the complexity of the underlying hardware as well as the firmware and software applications running on these systems, it can be extremely complicated to verify every possible functional and behavioral aspect of such system—this challenge is a main reason leading to various security vulnerabilities.

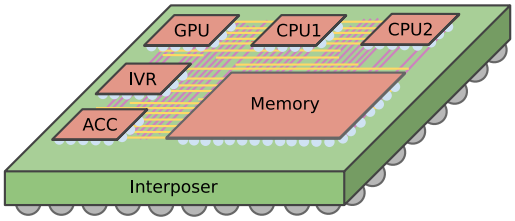


Figure 1: Integration of chiplets, i.e., smaller chips providing optimized hard IP components, on an interposer.

In modern computing systems, where various design IP blocks like processor cores and accelerators, developed by different third-party vendors, are integrated into one system, and where many different clients' software may be running at the same time, establishing some *root of trust* is critical to ensure the security and integrity of the system and the data being handled on it. Commercial solutions such as ARM's TrustZone [13] and Intel's SGX [14], as well as various academic proposals [15, 16, 17, 18], typically rely on dedicated microarchitectural support and other measures, e.g., memory encryption. These approaches often incur a high overhead, are prone to dedicated attacks [19, 20], and nevertheless are still susceptible to hardware Trojans that might be inserted by various third-party actors involved throughout outsourced supply chains [11, 12].

Another notable trend for modern computing systems is the adoption of heterogeneous hardware organization based on chiplets and interposers [21, 22, 23]. Instead of implementing a large system as one monolithic system-on-chip (SoC), this approach disaggregates the functional components across multiple smaller chips, referred to as *chiplets*, which can be designed and manufactured separately with optimized yield and other benefits. These chiplets are then leveraged as hard IP, possibly sourced from a variety of vendors, and consolidated on an integration and interconnects carrier, i.e., the interposer [21, 22, 23, 24, 25]. Figure 1 illustrates such a system where a variety of hard IP chiplets are integrated on an interposer.

The adoption of chiplet and interposer integration raises the important notion of design reuse to the level of the physical system, which provides significant cost benefits. In fact,

chiplet and interposer integration has already been widely adopted by industry with products such as the AMD Epyc processors [23] and the Intel Embedded Multi-Die Interconnect Bridge technology [26] available in the market. Still, for naive chiplet and interposer integration, concerns remain regarding system security vulnerabilities being introduced through the outsourced chiplets, e.g., via untrusted fabs [27] or malicious or simply buggy third-party IPs [28].

The coherence protocol of a system is a reasonable target for a maliciously acting hardware Trojan. This is because the coherence protocol acts as the mediator between chiplets' access to the memory system; a hardware Trojan targeting at the coherence of a system could allow for attackers to gain, or deny [29], control of the memory system or enable other attacks.

In our work, we leverage the notion of chiplets- and interposer-based design to establish a secure-by-construction root of trust in such multi-core, multi-chip systems. To this end, we introduce a security-centric interconnect fabric within an active interposer, which monitors and controls all system-level communication with low performance overheads. The contributions of our work are as follows:

1. We examine how untrusted chiplets, integrated together into an interposer-based larger system, can be attacked directly via unprivileged memory references as well as cache coherence side-channels.
2. We propose the use of an active interposer as a physical backbone for a secure-by-construction root of trust in such multi-core, multi-chip systems.
3. We introduce a novel microarchitecture to securely handle all communication passing from untrusted chiplets onto the interposer (and thus into the system), in terms of unprivileged memory references and cache coherence side-channels, based upon per-packet validation at the interposer ingress links.
4. We implement our proposed technique and examine the workings and implications of our security features. We also empirically characterize the performance impact, under full-scale system emulation, and show the overheads to be low, namely $\sim$4% on average.

2. BACKGROUND & MOTIVATION

Here we review the key concepts of IC and interposer technology, hardware security, and cache coherence protocols. Based on respective challenges for the state-of-art, we are motivating our proposed interposer-based root of trust. We also outline the background of the baseline system architecture.

2.1 IC Manufacturing and Security

Industry has widely adopted a work mode where IC design and verification is carried out by a design house and partners, but fabrication and testing is outsourced to off-shore facilities typically providing access to advanced technologies. While this practice reduces the cost of production and streamlines the time to market [30], it raises concerns regarding the trustworthiness of the outsourced fabrication facilities, which may seek to insert security vulnerabilities in general or hardware Trojans in particular [27].

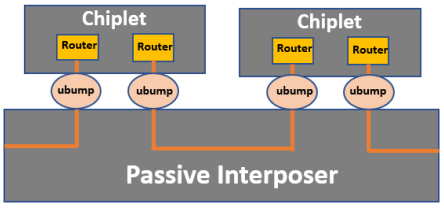


Figure 2: Passive interposer.

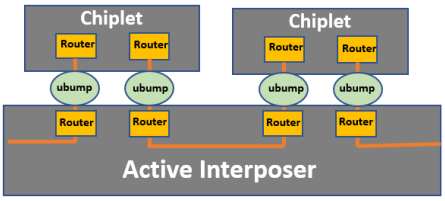


Figure 3: Active interposer.

The threat vector posed by untrusted fabrication facilities prevents the ICs manufactured there from being viewed as fully trustworthy. This can cause a larger security challenge for modern systems in multiple ways. First, any hardware security feature embedded in such outsourced IC may no longer offer the desired protection, which represents a profound challenge. Second, a modern system may be composed of chiplets, possibly fabricated in multiple facilities, with various levels of trustworthiness. Any malicious chiplet behavior may compromise the whole system due to its integrated and interconnected nature.

The interposer technology, if applied carefully, can help to avoid such complications. This is because an interposer can be fabricated separately in a trusted facility and may also embed security features. Accordingly, an interposer can be designed to constitute a hardware-enforced root of trust, which can be built upon to ensure the overall system's trustworthiness [31].

2.2 Interposer Technology

The interposer technology, also known as 2.5D integration, generally describes the process of manufacturing two or more chips or chiplets separately and subsequently stacking, bonding, and interconnecting them using an integration carrier made of silicon or other materials [21, 22, 23, 24, 25].

Compared to the traditional and monolithic SoC design approach, the interposer technology drastically reduces time to market for modern and large-scale computing systems. A system designer can procure IP as commodity chiplets and directly integrate them at the system level, with efforts only required for designing the interposer.

Figures 2 and 3 contrast, in simplified abstractions, a passive and an active interposer. In both cases, the chiplets are flipped and mounted onto the interposer using microbumps (ubumps). The main difference is that an active interposer also contains active devices, namely network-on-chip (NoC) routers in the example of Fig. 3. In contrast, passive interposers solely act as an integration carrier and wiring medium,

but do not interact with signals that traverse it.[1] An active interposer with an embedded NoC fabric allows an advanced interconnect fabric to be built, serving well for large-scale chiplet integration and system communication. Furthermore, the chiplet interconnect fabric is encapsulated away from the interposer NoC architecture beyond the edge router on the interposer to which it is attached. Such heterogeneous fabric allows for the cross-optimization of topologies and technologies across chiplets and interposer, opening up considerable opportunities for system design [21, 32, 33, 34, 35, 36].

Further to these general benefits of an active interposer, we note that active interposers are typically manufactured in older nodes [21, 31]. Therefore, it is realistic that a trusted facility is available for manufacturing of such active interposer. Accordingly, in this work, we propose an interposer-based root of trust with security features directly embedded within the NoC routers placed in an active interposer.

2.3 Hardware Security

2.3.1 *Hardware Trojans*

Hardware-centric attacks such as malicious insertion or modifications of circuitry, also known as *hardware Trojans*, can lead to catastrophic security failures within a system. For example, Bidmeshki et al. [37] provide an attack scenario wherein a hardware Trojan renders the cryptography subsystem vulnerable. Khan et al. [38] have demonstrated Trojans that can leak data from cache memory of processors. Kim et al. [29] introduce Trojans which inject malicious coherence messages to create a denial-of-service attack.

We note that our work is orthogonal to and compatible with prior art on Trojan detection and mitigation, e.g., [39, 40, 41]. That is, we do not seek to detect or prevent hardware Trojans, but to prevent related attacks from affecting the system-level security. More specifically, we seek to prevent any hardware-centric attacks that are executed through the system-level cache-coherent memory system of modern computing systems. This notion of system-level security is enforced by a clear physical separation of untrusted commodity chiplets and security features residing in the trusted active interposer; prior art on Trojan detection and mitigation cannot offer such secure-by-construction hardware organization.

2.3.2 *Secure Interconnect Fabrics*

Selected prior art focuses on securing the system through the encryption and decryption of packets or messages exchanged between cores through NoC fabrics [42, 43]. Similarly, Kinsy et al. [15] propose an architecture that organizes secure and non-secure software/hardware entities as tenants and configures the NoC routers for the secure exchange of network messages. While enabling a secure NoC fabric, the amount of key exchanges required to isolate selected nodes/tenants incurs high latencies and is not easily scalable. Fiorin et al. [44] propose security features for policy-based message checking in NoC fabrics. Nabeel et al. [31] propose an interposer-based architecture where security modules monitor the interconnect fabric at the low level of bus addressing, to block malicious transactions that violate memory access policies. They also leverage an interposer for security, however their system does not consider cache coherent memory traffic as the primary mode of communication between cores and IPs as in typical modern computing systems. Valamehr et al. [45] focus on monitoring signals routed through their notion of a control chip that is 3D-stacked to the host computation chip. Dedicated monitor circuitry is used to force all sensitive communications to traverse the control chip such that, once a malicious signal is detected, the control chip can override that signal.

Most of this prior art implicitly assumes trusted manufacturing of the whole system. As discussed above, such assumptions are challenged by outsourced IC supply chains. In contrast, our work does not make such overarching assumptions—we enforce system-level security for untrusted commodity hardware modules by integrating them on an interposer-based root of trust, the only component requiring trusted fabrication.

2.3.3 *Root of Trust*

Concerning hardware support for root of trust schemes, Intel's SGX provides an extension to create trusted execution environments, called enclaves [14, 46]. Enclaves prevent unprivileged access to secure data during security-sensitive execution. More specifically, SGX maps protected memory pages to reserved memory regions in which the pages are encrypted by a hardware encryption module. Recent work, however, has shown vulnerabilities in SGX arising from programming errors and reliance on untrusted software [47, 48] as well as from speculative execution [4, 8, 49, 50], though many of these vulnerabilities have since been corrected.

ARM's hardware-enforced Trusted Execution Environment (TEE) isolates secure execution from untrusted software [13]. TEE leverages a normal operating system (OS) running in tandem with a secure OS. The latter has access to the full range of a device's peripherals and memory, whereas the normal OS has only access to a subset of peripherals and memory regions, to prevent unauthorized access of sensitive resources. However, recent work has shown TEE implementations are prone to vulnerabilities due to architectural, implementation, and hardware issues [51].

In general, schemes like the above tend to incur high overheads, are prone to dedicated attacks, and are susceptible to hardware Trojans. In contrast, our approach has little impact on system performance and its key components are secure-by-construction.

2.4 Cache Coherence and Security

Coherence protocols ensure updates to cached copies of data are visible to all cores so they can receive the latest version of the data. Broadly, coherence schemes can be categorized as broadcast (or snooping) protocols [52, 53, 54] and directory protocols [55, 56, 57]. Broadcast protocols, while simple to implement, suffer from high traffic due to the amount of messages multi-core systems require to maintain coherence. Directory protocols allow for fine-grained state

[1] Although passive interposers are simple and cheap to manufacture and fully support the key idea of system-level integration of multiple chiplets, the physical design of such systems can become very challenging [22, 32]. For example, the interposer wires are of considerable length as they must follow all chiplet outlines to connect with microbumps; this can incur significant power and delay overheads on those system-level interconnects.

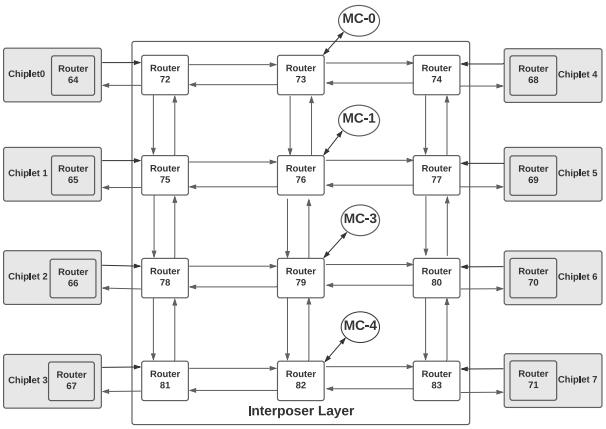


Figure 4: Overall system architecture. Routers with IDs 64–71 are the central routers within chiplets, connecting to the interface routers of the interposer NoC. Note that routers with IDs 0–63 connect the CPU cores within their respective chiplet NoC fabric (not illustrated).

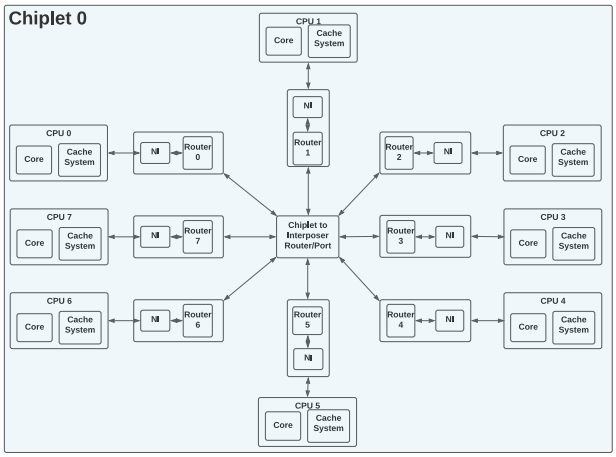


Figure 5: Chiplet architecture. NI is short for network interface.

tracking and unicast messages, making them highly scalable but difficult to implement and have higher access latencies.

A system's coherence protocol is generally oblivious to software attacks and may thus permit malicious accesses. Prior work demonstrates that this obliviousness can indeed be exploited to leak sensitive information [58, 59]. Existing countermeasures address conflict-based and transient execution side channels, but do not consider threats from maliciously manipulated/malformed coherence message packets [60, 61, 62, 63]. Given that the coherence protocol only acts as a set of rules for how memory is updated across multiple actors, attackers may exploit the coherence protocol's low-level behavior. For example, hardware Trojans injected into a chiplet could be designed to maliciously modify coherence messages to mount attacks on other chiplets, as outlined in our threat model in Sec. 3.

Since modifying existing protocols requires extensive verification and can result in dangerous side-effects for the system behavior, our proposed solution does not affect the coherence protocol. Rather, we carefully ensure messages' integrity and prevent untrustworthy actors (chiplets) from exploiting the coherence protocol's behavior.

2.5 Baseline System Architecture

Here we outline the baseline architecture for our interposer-based system. The system is loosely based on the architecture of the Rocket-64 design proposed by Kim et al. [22].

2.5.1 System Architecture

Figure 4 shows the baseline interposer-based, multi-chip, multi-core system architecture used for evaluation in our work.[2] In this system, we leverage eight chiplets, each containing eight CPU cores, for 64 cores in total. The chiplets are integrated via an active interposer as discussed in Sec. 2.2. Chiplets are interconnected to each other and to the four memory controllers (MC) via an NoC of 2D mesh topology residing in the active interposer. The interface routers, depicted along the east and west edges of the system, serve as the ingress links for the chiplets into the interposer NoC.

2.5.2 Chiplet Architecture

The top-level architecture of a chiplet containing multiple cores and respective private cache memories is shown in Figure 5. For the purpose of this study, we examine chiplets containing multiple processor cores, similar to AMD's present-day processors [23]. That said, our proposed architecture places no restrictions on the chiplets' contents, be they cores, accelerators, GPUs or others, as long as they are cache-coherent and obey shared-memory semantics.

Each chiplet contains eight CPU cores in our baseline model. Each of these cores have a L1 instruction and data cache, and a unified L2 cache, all of them private to each of the cores. The cache controllers generate cache coherence messages which are converted into network packets by the network interfaces (NI) in each chiplet, prior to being injected into the interposer NoC via its interface routers.

2.5.3 Cache Coherence

We focus on the MOESI Hammer cache coherence protocol [64] as the basis of our implementation; however, our design is extendable to any desired coherence scheme. The MOESI Hammer protocol was initially implemented in the AMD Opteron design as a scalable protocol for multi-core systems [64]. The MOESI Hammer protocol is a light-weight hybrid protocol. It encapsulates the scalability of directory-protocols without the high implementation complexity, and it achieves the low-latency of broadcast protocols without the increased coherence message traffic typical of broadcast schemes. To that end, the MOESI Hammer protocol maintains a sparse directory between multiple home nodes to track cache lines' states and owners. Requests access a cache line's

[2] Many other architectures and configurations are practical for interposer-based systems. Assuming cache coherent shared memory and an active interposer, our proposed security features can be ported to those as well.

home-node directory and DRAM in parallel, to reduce the cost of a directory miss, cancelling the DRAM response if a directory entry is found. Network traffic is reduced by only broadcasting to all coress for specific state transitions.

We note that by choosing a protocol that leverages broadcast messages, there is an increased exposure to potential side-channels, e.g., by a hardware Trojan snooping broadcast messages to memory regions which the given chiplet cannot access. A key feature of our proposed technique addresses these side-channels directly.

3. THREAT MODEL

Creating an architecture robust enough to handle all facets of security is a grand challenge. Thus, here we specify the particular threat vectors our architecture is designed to counter, along with a discussion of the limitations of our work.

3.1 Hardware-Based Threats

The focus of this work is a system wherein multiple chiplets have been fabricated in various facilities and then have been connected together using interposer technology. The assumption is that the fabrication of the chiplets, either designed in-house or obtained as third-party IPs, cannot be trusted. We also assume that attacks are targeted at memory traffic and the cache coherence messages of the system, which is the only type of traffic passing through the interposer.

Specific hardware-focused attacks create unique security challenges in interconnected systems, although these can be broadly categorized as outlined in [65]. Accordingly, our threat model addresses the related four types of hardware-focused threat vectors on system-level communication:

- **Passive reading**: Also known as snooping, this threat occurs when a malicious chiplet can read data meant for other chiplets. For example, a malicious chiplet may monitor another chiplet's cache accesses, by monitoring invalidation broadcasts caused by the victim chiplet.
- **Masquerading**: Also known as spoofing, this threat occurs when a malicious chiplet disguises itself as another chiplet to gain access to critical data or control of resources. Malicious chiplets can modify the requestor IDs and memory addresses embedded in cache coherence messages, tricking the coherence directories or other unsuspecting cores into passing critical data to those malicious chiplets.
- **Modifying**: Such threat modifies cache coherence messages, potentially modifying otherwise secure data. For example, a chiplet can try to disguise itself as having write access to a memory region it had read-only access to.
- **Diverting**: In shared-memory applications, a malicious chiplet may directly divert data meant for one chiplet to another untrusted chiplet, bypassing existing memory permissions, or may divert cache coherence messages, undermining the coherence protocol.

3.2 Software-Based Threats

In general, we also aim to prevent software-driven unauthorized access to memory regions at the chiplet granularity, whether by software privilege escalation, transient execution attacks, cache side-channels, or by any other means. Our proposed work will prevent attacks running on some chiplet from violating the security of the overall system, as we physically enforce protection against any unauthorized access to shared-memory regions and conduct continuous checking of the integrity and validity of cache coherence messages.

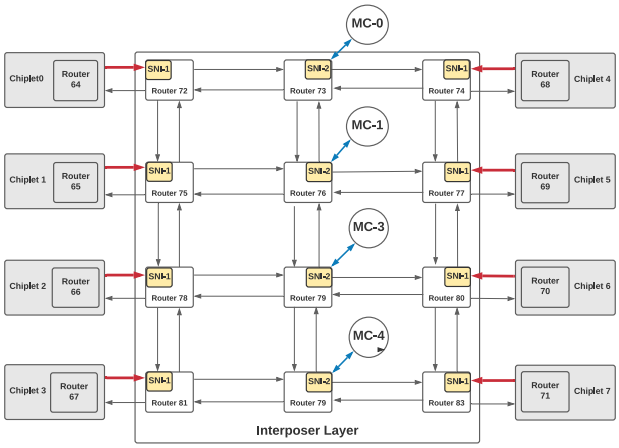


Figure 6: Proposed secure system architecture. SNIs are embedded within routers of the interposer NoC, along the ports/links connected to chiplets (SNI-1, red arrows) and memory controllers (SNI-2, blue arrows).

3.3 Limitations

Intra-chiplet attacks are out of scope for this work. Hence, side-channel attacks across cores within the same chiplet, e.g., [66, 67], are out of scope. Similarly, software-based attacks wherein code running on a core within a given chiplet attempts to violate the security of other processes also running on that same core, or another core in the same chiplet, are outside the scope for this work. Further, attacks wherein one chiplet can leverage memory transactions to its assigned memory region to modify DRAM rows that are not assigned to it, e.g., Rowhammer [68], are out of scope for this work.

4. DESIGN

First, we give a broad overview of our design and approach. We then outline the proposed system architecture, microarchitecture, and finally the implementation in detail.

4.1 Design Overview

Our objective is the integration of untrustworthy chiplets into a secure system. To this end, we propose that the interposer be the root of trust in the system. The key attributes that would enable such secure system are the following:

- The interposer is to be manufactured separately, in a trusted facility, independent from the untrusted chiplets.
- The interposer serves as the integration and communication backbone for the multi-chiplet, multi-core system. Any system-level communication across chiplets has to pass through the interposer.

Accordingly, all memory system traffic must traverse through the interposer as network packets. If a CPU core in a chiplet

wants to read/write data from/to memory, the corresponding cache coherence messages, embedded in these packets, will have to traverse the interposer. Similarly, if a core within a chiplet wants to communicate with another core in another chiplet, the messages have to traverse the interposer as well.[3]

These considerations lead to the straightforward decision to embed related security features directly within the interposer, because then all messages must inevitably traverse through and are checked by the trusted, active interposer. Therefore, we add *Security Network Interfaces (SNIs)* to the physical ingress links. These SNIs serve to validate all coherence messages coming from the chiplets into the active interposer. We also add SNIs to the physical links connecting with the memory controllers; the related details are discussed in Sec. 4.3.1. Note that the SNIs are implemented exclusively within the trusted active interposer, thereby rendering the SNI hardware trustworthy and free from the influence of hardware Trojans by construction. Figure 6 shows our secure system so defined.

4.2 Shared-Memory System Architecture and Operating System Support

In a typical shared-memory system, per-process memory permissions are defined on the basis of physical pages by the OS at time of memory allocation. While designing a system capable of enforcing these per-page permissions in some form of SNI is possible, doing so would pose several challenges. In particular, page-level tracking would require implementing a TLB-like structure for caching translations [69]. The required hardware and operating support for maintaining the structure in coherence with the full system's page table may significantly increase hardware complexity and performance overhead.

Given that the system is composed of relatively few chiplets of coarse granularity, we argue that such page-level implementation at the interposer would be excessive. Instead, we consider partitioning the available physical main memory into a coarse-grained set of memory regions, similar to some prior proposals [70, 71, 72, 73]. Each memory region can be designated as read- or write-able independently to any given chiplet; a given region's per-chiplet permissions can also be changed as needed over time. In our scheme, then, data private to a single given chiplet would be placed in a region (or set of regions if more space is needed) to which read/write permissions are given only to that chiplet and no others. This initial memory partitioning and permission setting occurs during the initial soft page fault on a given allocated virtual page, with some modifications to the physical-page allocation routine in the OS. When a page must be shared across multiple chiplets, it is assigned to a memory region wherein both chiplets are allowed access. In the rare event a page's sharing permission must be changed after allocation, it may be moved into memory region with the appropriate sharing permissions at some overhead.

In designing this system, we aim for a sweet spot between too coarse grained, where only a handful of memory regions are available and capacity may well be wasted to fragmentation, versus too fine grained, where the hardware table could not hold the overly large number of regions, at least not without incurring high access latency or requiring entries to be placed in a backstore. We find that a total number of memory regions between 4x–8x the number of chiplets is more than sufficient for most use-cases, allowing diverse private and shared memory regions without too much fragmentation.

Designing a secure OS to take advantage of our active interposer-based root of trust is beyond the scope of this work. That said, various prior works have explored security-enabled OS environments [71,73,74,75,76] as well as trusted execution environments (see Sec. 2.3.3), which could be extended to utilize the proposed interposer-based root of trust. Further, the interposer could support the inclusion of a secure co-processor that would provide support for a trusted OS, for secure boot-up and execution environments [13,72,77]. In our scheme, particular tasks like updating the access permissions at runtime could be delegated to such a secure environment.

4.3 Microarchitecture

Figure 6 shows our proposed interposer-based system with SNIs inserted at the links between the interposer and the chiplets or memory controllers. SNIs are implemented exclusively within the interposer. Next, we discuss the microarchitecture of the SNIs in more detail.

We adopt the idea of monitoring and validating incoming packets to the interposer, in order to detect attacks wherein malicious and/or modified packets would be utilized, as outlined in Sec. 3. Figure 7 depicts the SNI embedded in a router of the interposer NoC. In general, the SNI monitors the messages traversing the physical links prior to entering the corresponding virtual channel buffers within the routers.

Each SNI has two components described as follows:

1. **Packet Checker/Modifier (PCM)**: The PCM monitors and/or modifies cache coherence messages as needed. Because the proposed system follows standard shared-memory semantics, all legal communication between cores, other IPs, as well as memory occurs through memory accesses, and these accesses create cache coherence messages. Thus, the PCM works directly on cache coherent packets for checking addresses, permissions, etc., and for modifying messages as needed. More details are discussed in Sec. 4.4.

2. **Address Protection Unit (APU) Table**: This is a direct-mapped, SRAM-based look-up table with entries for each memory region and their associated per-chiplet permissions. As outlined in Sec. 4.2, the main physical memory is partitioned into multiple fixed-size regions. An entry is mapped within the APU table for each of these sections of memory. Hence, the number of entries within the APU table is determined by the number of regions in the main memory.

4.3.1 SNI Types and Placement

Recall that Fig. 6 depicts SNIs embedded in the secure interposer-based system. The SNIs connected to the physical links from chiplets to interposer are denoted as "SNI-1" and those connected to the physical links from memory controllers are denoted "SNI-2". The difference between these

[3] All direct messages between chiplets/cores must conform to the cache coherence protocol. Thus, direct messages are limited to legal cache coherence messages such as invalidation replies.

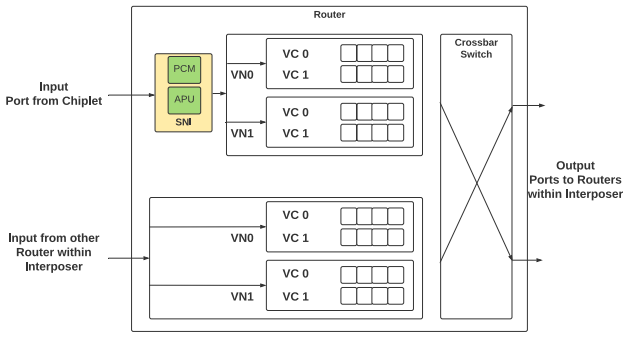


Figure 7: An SNI, embedded within an interface router of the interposer NoC, monitoring the packets traversing through the physical links.

	Chiplet 7	Chiplet 6	Chiplet 5	Chiplet 4	Chiplet 3	Chiplet 2	Chiplet 1	Chiplet 0
Entry 0	0	0	0	0	0	0	0	3
Entry 1	0	0	0	0	0	0	3	0
Entry 2	0	0	0	0	0	3	0	0
Entry 3	0	0	0	0	3	0	0	0
Entry 4	0	0	0	3	0	0	0	0
Entry 5	0	0	3	0	0	0	0	0
Entry 6	0	3	0	0	0	0	0	0
Entry 7	3	0	0	0	0	0	0	0
Entry 8	0	0	0	0	0	0	3	3
Entry 9	0	0	0	0	0	1	1	3
•								
•								
•								
Entry 61	3	0	0	0	0	0	1	1
Entry 62	0	1	3	1	0	0	0	0
Entry 63	1	1	0	0	0	1	1	3

Figure 8: An APU table, configured for 64 memory regions, with each region covering 1/64 of the available physical memory. Each entry describes access permissions for each chiplet individually. Some memory regions are private, e.g., Region 1 to Chiplet 1, while others are shared, e.g., Region 8 is read-write shared between Chiplets 0 and 1.

two types of SNIs is that SNI-1 only monitors and verifies cache coherence messages entering from the chiplet to the interposer, whereas SNI-2 performs modification of certain coherence messages at the directories, to counter passive reading threats on broadcast messages, as described in detail further below. Not all physical links need to be monitored by SNIs. In particular, router-to-router connections running exclusively within the interposer need no SNI as the interposer is the root of trust itself.

SNI-1: Its purpose is to prevent the attached chiplet from injecting malicious coherence messages into the system which would violate the provisions of the shared-memory architecture, as outlined in Sec. 4.2. To this end, the PCM within SNI-1 monitors all injected traffic from the attached chiplet on the basis of the physical address to which the packet refers. This physical address is compared against the per-region permissions stored in the APU table (described further below). In the event that the message is of an allowed type, to an allowed memory region for the given chiplet (e.g., a GETX to a read-only memory region it owns), the packet will be permitted to proceed into the interposer's NoC. Otherwise, should the packet have to be rejected, a dedicated security signal realized as machine-check exception is thrown and execution of the whole system stops.[4]

SNI-2: Its purpose is to prevent the broadcast of coherence messages to chiplets which are not permitted access to the related memory regions. As described in Sec. 2.5.3, the MOESI Hammer coherence protocol used in our system employs certain broadcast messages as follows. The directories do not maintain per-core sharing information in this protocol, hence certain message requests cause the directory to broadcast the corresponding request to all cores. The cores then respond based on whether the cache block is shared by that core. In turn, this raises a concern of passive reading/snooping (see Sec. 3).

[4]This is a secure and protocol-conform approach. For the sake of system-level throughput, one may want to only stop and isolate the chiplet(s) triggering such a security violation. Doing so safely, however, is not trivial, as it would require significant modifications of the coherence protocol itself to prevent deadlocks. Such considerations are left for future work.

To address this snooping threat, we revise the PCM module to recognize broadcasted messages and reformat them as follows. Based on the information contained in the APU table (described further below), the PCM determines whether a given broadcast message is being directed towards a chiplet that is allowed access to the referred memory region. In case the chiplet has no access to the referred memory region, the broadcast message is converted into an appropriate response message which is directed only to the original requester core/chiplet. This is legal within the coherence scheme of the system because if the chiplet in question is not allowed to access memory within the given memory region then it cannot have cache lines associated with that region in its caches. This allows the SNI-2 to divert broadcast messages from the directory and thus prevent snooping.

4.3.2 APU Table

As the cores and IP blocks in the system execute given workloads, memory is accessed in the form of instruction or data. The OS is responsible for the memory mapping of the virtual to physical memory for these applications. The APU table holds entries that describe the access permissions for applications running within chiplets. Within the APU table, each entry corresponds to a pre-determined physical memory region. Note that these permission levels are determined by the OS and programmed into the APU table during runtime, as outlined in Sec. 4.2.

Figure 8 depicts the structure of our APU table. As shown, each entry represents one and only one memory region. Each entry allocates two bits for each chiplet, capturing the access permissions of an application that is running within those chiplets. There are three permission levels for each chiplet, encoded as follows:[5]

- **No Access ("00" or 0)**: The chiplet has no access to the concerned memory region.
- **Read-Only Access ("01" or 1)**: The chiplet has read-only access to the concerned memory region.

[5]Encoding "10" or 2 is unused.

- **Read-Write Access ("11" or 3)**: The chiplet has read and write access to the concerned memory region.

As indicated, the APU table is an SRAM array, serving as lookup table, with entries containing per-chiplet permissions for each memory region in the system. When the PCM intercepts a packet, the upper bits of its physical address[6] are extracted and used to index into the APU table. The related entry is read and handed back to the PCM so that the request type, requester ID and destination ID are compared against the permission levels in the APU table entry. See also Sec. 4.4 for more details.

4.4 Implementation Details

Here we discuss implementation details for a baseline implementation of our interposer-based root of trust, with particular focus on those aspects relevant for the SNI design.

4.4.1 NoC Configuration

The network topology (as well as the routing algorithm) of the interposer NoC do not impact the SNI design. This is because SNIs are emplaced at the interface between chiplets or memory controllers and the active interposer, whereas a similar placement will be followed in any topology. However, the width of the physical link has an impact on the SNI design and its logic. In our design and evaluation, the link width is 128 bits within chiplets and 64/128 bits within the interposer.

Recall that we adopt the broadcast-based MOESI Hammer protocol [64], described in Sec. 2.5.3, and also discussed in more detail for its message field further below. Every MOESI Hammer control message fits within a 128-bit flit. Thus, when such flit enter the interposer, it is broken down into two/one flits which are analyzed in the SNI logic over two/one clock cycles, depending on the 64/128 width of the interposer link. In the case of 64-bit links, also depicted in Fig. 9, we dedicate the first cycle to extract the control parameters from the head and the second cycle to extract the address for the cache block being accessed by the coherence message. Note that the SNI logic is similar for request and response messages, as both cases require the first two flits to be analyzed.

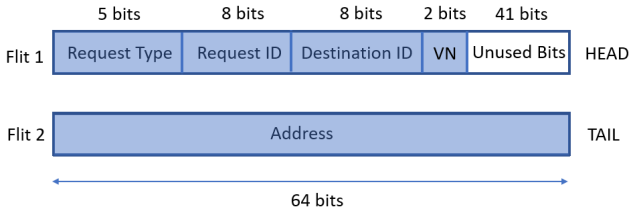
For other system configurations, with different link widths and/or packet size, we may need to expand the SNI design and its logic, to retrieve the fields across flits over multiple pipeline stages and subsequently perform security checks on the accumulated data. Hence, the physical-link width and the flit size determine the latency observed at SNIs.

4.4.2 Cache Coherence Protocol

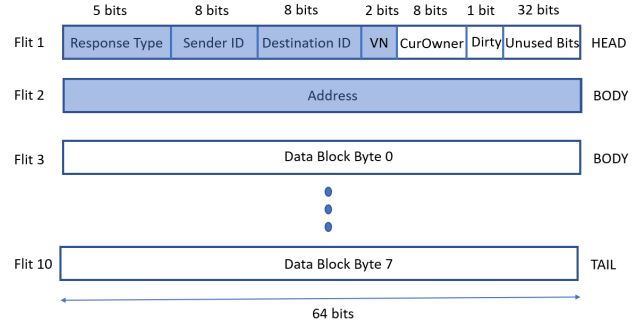
The system's cache coherence protocol, MOESI Hammer in our work, directly impacts the SNI design and logic as the coherence message fields need to be analyzed by the SNI.

Here, we elaborate on the message fields (Fig. 9). Note that MOESI Hammer response messages are either a control or data message; a control response follows the same flit structure as that of a request message, whereas a data response carries additional flits with 64 bytes each of cache line data.

- **Request/Response Type**: This specifies the type of cache coherence request and related response. We have ten types of requests and twelve replies. Hence, five bits are allocated for this field.
- **Requester/Sender ID**: This specifies the requester/sender identity. For our baseline system, we have 64 cores and four memory controllers, i.e., 68 nodes in total. We allocate eight bits for this field, to allow for larger system configurations as well.
- **Destination ID**: This specifies the destination. As with requester/sender ID, eight bits are allocated.
- **VN**: This specifies the virtual network (VN) that this message belongs to. We have four different VNs, hence two bits are required.
- **Address**: This is the 64-bit physical memory address of the cache line under consideration.
- **CurOwner**: This response field specifies the current owner of the cache block. This is required for handling *unblockS* responses for the MOESI Hammer protocol.
- **Dirty**: This response field specifies if the cache block is dirty or not, as in whether a cache block has been modified or not.
- **DataBlock**: This response field contains the data. If the response is of type *Control*, there is no corresponding DataBlock.



(a) Request Message structure



(b) Response Message structure

Figure 9: Structure of messages. Fields highlighted are to be checked by SNIs

Based on a rigorous analysis of the message types and the possible threats in our model (Sec. 3), we observe that certain key parameters must be analyzed by the SNI; these parameters have been highlighted in Fig. 9. These parameters are extracted by the PCM and then compared with the permission information present in the APU table, to determine if a coherence message was tampered with or fabricated by some maliciously acting chiplet. We note that it is important to analyze both request and response messages, since an attacker may seek to exploit response messages as well, not only requests for data.

[6] Six bits in our baseline system for its 64 memory regions.

4.4.3 Malformed Packets

Cache coherence messages are converted into network packets by the chiplets' NIs, but it cannot be assumed that these packets adhere to the rules of the network and coherence protocol. For example, a Trojan within a chiplet may fabricate an invalid message type, yielding undefined and possibly vulnerable behavior.

We note that messages corresponding to particular VNs must follow a specific, limited set of requester/destination IDs and message types. Thus, the PCM checks the possible field values within the SNI, to verify the legality of messages. Since these checks are orthogonal from the memory region permission checking, they go on in parallel and incur no extra pipeline stages of delay.

4.4.4 Design Cost

Here we discuss the design cost for a baseline implementation of the proposed interposer-based root of trust, incorporating SNIs as described. We design the PCM module with three pipeline stages, one for lookup, a second for checking, and a third for packet modification. Note that this third stage maybe be bypassed if not needed. Accordingly, SNI-1 instances incur a two-cycle latency, as they only monitor packets on ingress to the interposer, whereas SNI-2 instances incur a three-cycle latency, also accounting for packet modification. An APU table requires two bits for identifying each of the eight chiplets, and there are 64 table entries. Accordingly, 1024 bits are required for an APU table. There are twelve APU tables in each of the router in the interposer layer, imposing a total memory footprint of 12Kbits over the baseline unsecured system.

5. EVALUATION

Here we first discuss our evaluation methodology. Then we discus the threat model coverage our scheme provides. Finally we examine the performance overheads caused by our system.

5.1 Methodology

We implemented and evaluated our proposed system in simulation in gem5 [78]. Table 1 depicts the configurations we evaluated. The system design configuration was chosen to approximate the recent Rocket-64 design [22]. As such, we simulate an 8-chiplet, 64-core system with each core operating at 1GHz. The interposer operates at a quarter of the frequency of the chiplets as the interposer is fabricated with a lower process node. Performance impact is measured in terms of the IPC speedup for the enabled SNI configuration over the baseline configuration without SNIs. In the SNI configuration, the SNIs incur latencies as discussed in Sec. 4.4.4. We disable these latencies in the non-secure baseline configuration.

Due to the long simulation times induced for this large system, we evaluate the performance impact using a subset of SPEC 2006 benchmarks. We perform single-threaded and multi-programmed benchmark simulations to understand the impact of the SNIs.

Component	Variable
Chiplet Architecture	
Core	8 RISC V cores
Private L1 I-Cache	32KB
Private L1 D-Cache	64KB
L2 Cache	2MB
NoC	Eight-port, 128-bit Crossbar
vc_per_vnet	4, 6, 8, or 10
Chiplet frequency	1GHz
System Architecture	
Chiplets	8
Memory Controllers	4
Main Memory	4GB
Memory Regions	64, 64MB each
NoC	3x4 2D-Mesh, 64-bit or 128-bit
vc_per_vnet	4, 6, 8, or 10
Interposer frequency	250MHz
Cache Coherence	
Model	AMD MOESI Hammer
Simulation Configuration	
Processor model	TimingSimpleCPU
Simulation model	System Emulation

Table 1: System Architecture Configuration

5.2 Threat Model Coverage

Our scheme addresses each of the threats discussed in Sec. 3 as follows:

- **Passive reading**: Snooping of broadcast messages is prevented by the SNI-2 re-routing the broadcast messages. Broadcast messages from the directories are converted into negative acknowledgments back to the requester, for chiplets where permission to the memory region of the given address is denied. This is done as the messages enter the SNI-2 at the interposer/memory controller boundary.
- **Masquerading**: SNI-1 instances are placed at each chiplet and interposer boundary. Every SNI-1 is programmed with the range of requestor ID's expected in each cache coherence message's requester ID field. For example, in Fig. 6, the SNI-1 in router 72 can expect requestor IDs in the range of 0 to 7 and will reject any message with an ID outside of this range, as discussed in Sec. 4.4.3. In this event, the SNI will throw a security machine check exception and halt execution.
- **Modification**: These threats are detected by comparing a message type, such as GETX/GETS, with the access permission levels in the APU table. If the message accesses memory addresses outside of its allowed address space, a security machine check exception is thrown.
- **Diversion**: Diversion threats are detected by checking the destination ID and the message type. Only specific message types can have other cores as the destination ID. This, along with the memory region permissions in APU table, allows us to detect any malicious diversion of coherence messages. Again, a security machine check exception is thrown if a threat is detected.

Software Threats: Our design generally prevents unauthorized accesses to memory regions due to privilege escalation or exploitation using the same mechanics described

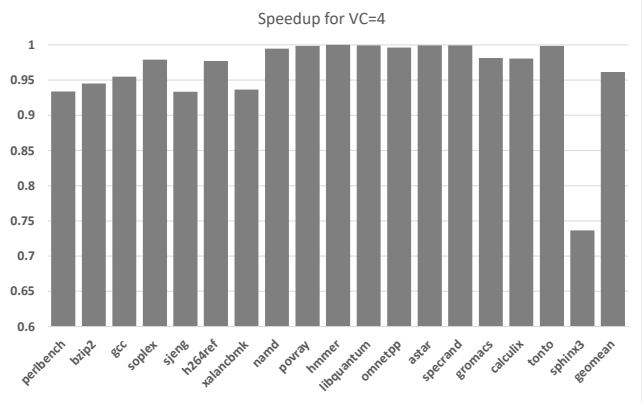


Figure 10: Speedup factor for the SNI-enabled system, compared to non-secure baseline architecture, for vc_per_vnet configuration of 4.

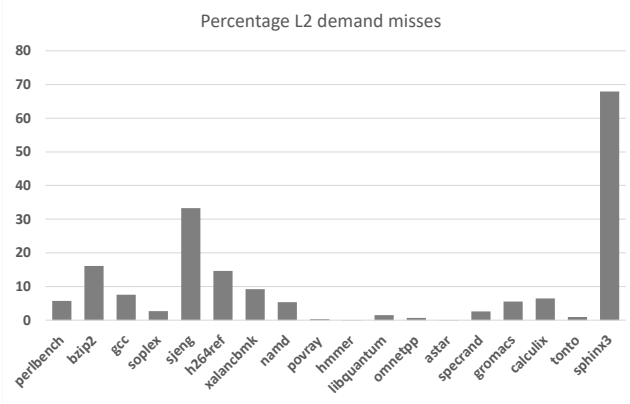


Figure 11: L2 cache misses.

above for hardware threats. Since coherence messages are generated by hardware, a solely software-driven attack cannot engage in masquerading, modification, or diversion threats through packet manipulation without malicious hardware intervention.

Security Testing: In order to test the system's ability to counter the threats discussed in Sec. 3, we injected malicious cache coherence messages at the network interface of cores to cover each of the above threats. We verified that in each case a machine check exception was thrown and no malicious packets were allowed into the interposer's NoC before the system was halted.

5.3 Single-Threaded Performance Impact

Figure 10 shows the speedup of the system with SNIs enabled compared to the baseline configuration without SNIs. All workloads experience a speedup less than 1, which is expected as the SNIs introduce higher latencies to the network. As the figure shows, on average the SNIs impose a performance loss of ∼4%, with several benchmarks (*povray*, *hmmer*, *libquantum*) showing little to no impact. *sphinx3*, however, is an outlier, showing a significant ∼27% performance loss.

To analyze the results further, we examine the L2 miss rates of each benchmark, as shown in Fig. 11. The figure demonstrates that the variation between each benchmark's result in Fig. 10 is highly correlated to the difference in the cache hit rate between benchmarks. For instance, *sphinx3* shows a much higher L2 cache miss rate than the other benchmarks at ∼68%, while workloads performing closer to a speedup of 1, such as *povray, hmmer,* and *libquantum*, have a lower cache miss rate. The SNI-enabled system's performance is dependent on the number of coherence messages injected into the network due to L2 cache misses. The SNIs must process each packet which results in increased memory access latencies.

To further analyze the source of the performance degradation in some benchmarks, Fig. 12 shows the percentage change for pre-injection queuing latency versus in-network latency and the total latency experienced by packets injected in the network for each benchmark. Interestingly, while the

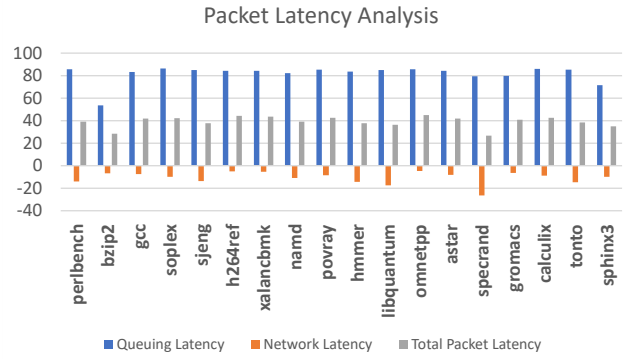


Figure 12: Packet latency breakdown, percent change in packet latency induced by the SNIs, broken out by queuing latency, and in-network latency.

queuing latency increases by ∼80%, the in-network latencies drop by 5-10%. While the increase in queuing latency is expected due to the extra pipeline delays on network insertion the SNIs cause, the decrease in in-network latency is due to the reduction in hop count caused by the SNI-2's rerouting of acknowledgment request messages back to the original requester as negative acknowledgment replies. Thus, the SNI-2 reduces total network load by removing one packet in the transaction.

The total latency of the packets increases by 39% on average. Interestingly, we see that, although *sphinx3* incurs a higher performance impact than the other benchmarks, it does not see a significantly different packet latency. *sphinx3*'s performance loss is due to a higher miss rate and hence more packet injection as discussed above, not a higher per-packet latency. The higher miss rate exposes *sphinx3* to the network latency increase more than other applications which have lower miss rates.

Figure 13 depicts the speedup of the benchmarks with three different configurations of vc_per_vnet. We observe that with an increase in the number of virtual channels, the geometric mean speedup approaches 0.98. In the case of *sphinx3*, we see a significant improvement in speedup. This occurs

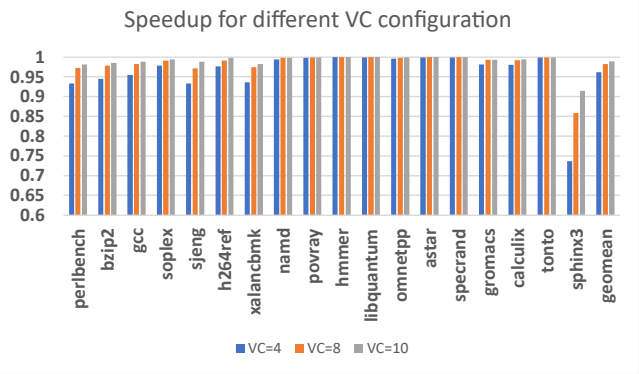


Figure 13: Speedup factor for different vc_per_vnet configurations.

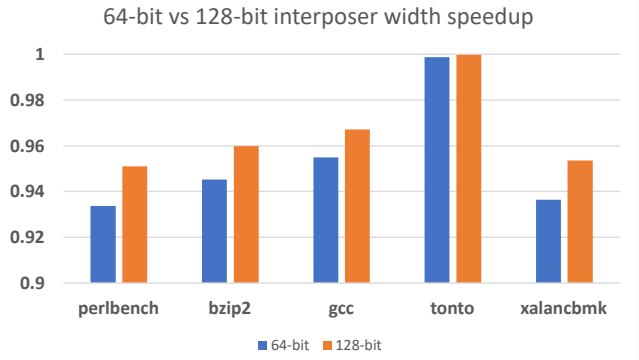


Figure 14: Speedup for 128-bit link widths within interposer.

primarily because of the improvement in queuing latencies at the network interfaces. These significant gains imply that increasing VC count is a good way to improve performance should the applications cause many cache misses.

In Figure 14, we analyze the impact of increasing the interposer link widths to 128 bits versus the baseline of 64 bits.[7] This improved bandwidth provides a slightly better speedup as compared to the previous configuration of 64-bit links within the interposer. These modest gains imply that increasing bit-width in the interposer is likely not worthwhile, although this depends on the designer's trade-off for costs/overheads and scalability of the system.

5.4 Multi-Programmed Performance Impact

We evaluate the impact of the SNIs multi-programmed workloads using random mixes of two benchmarks each, executed in two cores in separate chiplets. Here we simulate until all applications complete at least five billion instructions and we report the weighted speedup of the combination using a methodology from Kadjo *et al.* [79].

Figure 15 shows the speedup for these multi-programmed workloads. In general, speedups range between 0.95 and 1.06. In some cases, namely *bzip2-namd* and *bwaves-gcc*, the speedup with the SNIs enabled was more than baseline. Further, the mixes which included *sphinx3* showed reduced performance loss versus the stand alone *sphinx3*. With the increased interconnect bandwidth pressure that multiple applications induce, the reduction in interconnect bandwidth that SNI-2's packet modification causes has an increasing effect relative to the static pipeline latency increases that the SNIs incur. Thus, the overheads of the SNI approach appear to reduce with greater workload counts, though more benchmark runs would be useful to confirm the behavior.

[7] Due to runtime constraints for such large-scale simulations running on our shared high-performance computing cluster, we focused on a representative subset of benchmark runs.

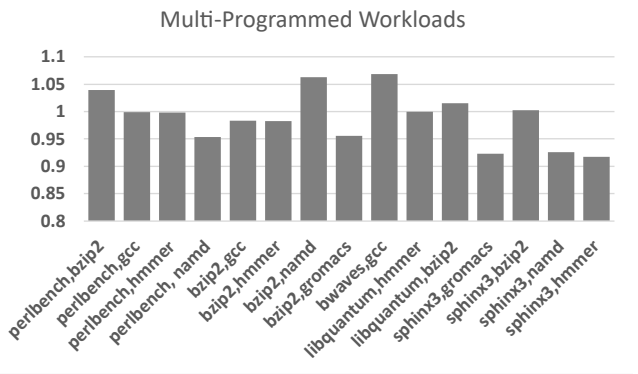


Figure 15: Speedup for multi-programmed workloads.

6. CONCLUSION

In this work, we propose the use of an active interposer as root of trust for modern chiplets-based systems, by implementing hardware security features directly within the interposer. More specifically, we devise and demonstrate a security network interface (SNI), which we propose to include at the boundary between the interposer and the chiplets/memory controllers. We show how such a scheme addresses various hardware and software attacks arising from malicious chiplets, with relatively low performance impact, ∼4% on average, compared to a non-secure baseline system.

In future work, we will tackle OS support and dynamic memory management for more complex applications being run on our secure system. We will also explore mechanisms to allow the controlled shut-down of a specific chiplet that violates security protocols, whereas the rest of the system would be allowed to proceed in a safe if degraded manner, rather than completely shutting down the system in prudence.

REFERENCES

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *IACR*, 2005.

[2] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[3] 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 *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[4] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre returns! speculation attacks using the return stack buffer," in

12th USENIX Workshop on Offensive Technologies (WOOT 18), (Baltimore, MD), USENIX Association, Aug. 2018.

[5] S. Islam, A. Moghimi, I. Bruhns, M. Krebbel, B. Gulmezoglu, T. Eisenbarth, and B. Sunar, "Spoiler: Speculative load hazards boost rowhammer and cache attacks," 2019.

[6] G. Maisuradze and C. Rossow, "Ret2spec: Speculative execution using return stack buffers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), p. 2109–2122, Association for Computing Machinery, 2018.

[7] V. Kiriansky and C. Waldspurger, "Speculative buffer overflows: Attacks and defenses," 2018.

[8] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), p. 991–1008, USENIX Association, Aug. 2018.

[9] J. Stecklina and T. Prescher, "Lazyfp: Leaking fpu register state using microarchitectural side-channels," 2018.

[10] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," pp. 18–37, 2016.

[11] S. Bhunia and M. M. Tehranipoor, eds., *The Hardware Trojan War: Attacks, Myths, and Defenses*. Springer, 2018.

[12] D. Mehta, H. Lu, O. P. Paradis, M. A. M. S., M. T. Rahman, Y. Iskander, P. Chawla, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "The big hack explained: Detection and prevention of pcb supply chain implants," vol. 16, no. 4, 2020.

[13] A. Limited, ""security technology building a secure system using trustzone technology," tech. rep., 2009.

[14] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, HASP 2016, (New York, NY, USA), Association for Computing Machinery, 2016.

[15] M. A. Kinsy, S. Khadka, M. Isakov, and A. Farrukh, "Hermes: Secure heterogeneous multicore architecture design," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 14–20, 2017.

[16] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *TC*, vol. 67, no. 3, pp. 361–374, 2018.

[17] H. Zhang, S. Ghosh, J. Fix, S. Apostolakis, S. R. Beard, N. P. Nagendra, T. Oh, and D. I. August, "Architectural support for containment-based security," pp. 361–377, 2019.

[18] I. Lebedev, K. Hogan, J. Drean, D. Kohlbrenner, D. Lee, K. Asanović, D. Song, and S. Devadas, "Sanctorum: A lightweight security monitor for secure enclaves," 2019.

[19] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in darkness: Return-oriented programming against secure enclaves," pp. 523–539, 2017.

[20] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," pp. 195–209, 2019.

[21] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, G. Moritz, I. Miro-Panadès, C. Fuguet, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. Meunier, A. Farcy, A. Arriordaz, S. Cheramy, and F. Clermidy, "A 220GOPS 96-core processor with 6 chiplets 3D-stacked on an active interposer offering 0.6ns/mm latency, 3Tb/s/mm2 inter-chiplet interconnects and 156mW/mm2@ 82%-peak-efficiency DC-DC converters," pp. 46–48, 2020.

[22] J. Kim, G. Murali, H. Park, E. Qin, H. Kwon, V. Chaitanya, K. Chekuri, N. Dasari, A. Singh, M. Lee, H. M. Torun, K. Roy, M. Swaminathan, S. Mukhopadhyay, T. Krishna, and S. K. Lim, "Architecture, chip, and package co-design flow for 2.5D IC design enabling heterogeneous IP reuse," 2019.

[23] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony, "AMD chiplet architecture for high-performance server and desktop products," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 44–45, 2020.

[24] M. Matsuo, N. Hayasaka, K. Okumura, E. Hosomi, and C. Takubo, "Silicon interposer technology for high-density package," in *2000 Proceedings. 50th Electronic Components and Technology Conference (Cat. No.00CH37070)*, pp. 1455–1459, 2000.

[25] S. Takaya, M. Nagata, A. Sakai, T. Kariya, S. Uchiyama, H. Kobayashi, and H. Ikeda, "A 100GB/s wide I/O with 4096b TSVs through an active silicon interposer with in-place waveform capturing," pp. 434–435, 2013.

[26] R. Mahajan, R. Sankman, N. Patel, D. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik, "Embedded multi-die interconnect bridge (emib) – a high density, high bandwidth packaging interconnect," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pp. 557–565, 2016.

[27] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[28] J. J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: A high-level synthesis approach," vol. 24, no. 9, pp. 2946–2959, 2016.

[29] M. Kim, S. Kong, B. Hong, L. Xu, W. Shi, and T. Suh, "Evaluating coherence-exploiting hardware trojan," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 157–162, 2017.

[30] P. Yang and M. Marek-Sadowska, "Making split-fabrication more secure," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[31] M. Nabeel, M. Ashraf, S. Patnaik, V. Soteriou, O. Sinanoglu, and J. Knechtel, "2.5d root of trust: Secure system-level integration of untrusted chiplets," *IEEE Transactions on Computers*, vol. 69, p. 1611–1625, Nov 2020.

[32] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "NoC architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?," pp. 458–470, 2014.

[33] P. Coudrain, J. Charbonnier, A. Garnier, P. Vivet, R. Vélard, A. Vinci, F. Ponthenier, A. Farcy, R. Segaud, P. Chausse, L. Arnaud, D. Lattard, E. Guthmuller, G. Romano, A. Gueugnot, F. Berger, J. Beltritti, T. Mourier, M. Gottardi, S. Minoret, C. Ribière, G. Romero, P. . Philip, Y. Exbrayat, D. Scevola, D. Campos, M. Argoud, N. Allouti, R. Eleouet, C. Fuguet Tortolero, C. Aumont, D. Dutoit, C. Legalland, J. Michailos, S. Chéramy, and G. Simon, "Active interposer technology for chiplet-based advanced 3D system architectures," pp. 569–578, 2019.

[34] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh, "Modular Routing Design for Chiplet-Based Systems," in *ACM/IEEE ISCA*, pp. 726–738, 2018.

[35] A. Coskun, F. Eris, A. Joshi, A. B. Kahng, Y. Ma, A. Narayan, and V. Srinivas, "Cross-layer co-optimization of network design and chiplet placement in 2.5D systems," 2020.

[36] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna, "Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling," 2020.

[37] M. Bidmeshki, G. R. Reddy, L. Zhou, J. Rajendran, and Y. Makris, "Hardware-based attacks to compromise the cryptographic security of an election system," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 153–156, 2016.

[38] M. N. I. Khan, A. De, and S. Ghosh, "Cache-out: Leaking cache memory using hardware trojan," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1461–1470, 2020.

[39] W. Hu, C. H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An overview of hardware security and trust: Threats, countermeasures and design tools," 2020.

[40] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "ICAS: an extensible framework for estimating the susceptibility of IC layouts to additive trojans," pp. 1742–1759, 2020.

[41] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin, "QIF-Verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment," pp. 91–100, 2019.

[42] C. H. Gebotys and R. J. Gebotys, "A framework for security on noc technologies," in *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.*, pp. 113–117, 2003.

[43] S. Evain and J. . Diguet, "From noc security analysis to design solutions," in *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, pp. 166–171, 2005.

[44] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure memory accesses on networks-on-chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1216–1229, 2008.

[45] J. Valamehr, M. Tiwari, T. Sherwood, R. Kastner, T. Huffmire, C. Irvine, and T. Levin, "Hardware assistance for trustworthy systems through 3-d integration," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, (New York, NY, USA), p. 199–210, Association for Computing Machinery, 2010.

[46] D. Costan, V., "S. intel sgx explained.," tech. rep., 2016.

[47] M. R. Khandaker, Y. Cheng, Z. Wang, and T. Wei, "Coin attacks: On insecurity of enclave untrusted interfaces in sgx," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, (New York, NY, USA), p. 971–985, Association for Computing Machinery, 2020.

[48] J. Park, N. Kang, T. Kim, Y. Kwon, and J. Huh, "Nested enclave: Supporting fine-grained hierarchical isolation with sgx," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 776–789, 2020.

[49] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel SGX," *CoRR*, vol. abs/1902.03256, 2019.

[50] 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.

[51] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1416–1432, 2020.

[52] E. E. Bilir, R. M. Dickson, Ying Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood, "Multicast snooping: a new coherence method using a multicast address network," in *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, pp. 294–304, 1999.

[53] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "Power5 system microarchitecture," *IBM Journal of Research and Development*, vol. 49, no. 4.5, pp. 505–521, 2005.

[54] N. Agarwal, L. Peh, and N. K. Jha, "In-network snoop ordering (inso): Snoopy coherence on unordered interconnects," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 67–78, 2009.

[55] J. Archibald and J. L. Baer, "An economical solution to the cache coherence problem," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, ISCA '84, (New York, NY, USA), p. 355–362, Association for Computing Machinery, 1984.

[56] J. Zebchuk, M. K. Qureshi, V. Srinivasan, and A. Moshovos, "A tagless coherence directory," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 423–434, 2009.

[57] J. Laudon and D. Lenoski, "The sgi origin: A ccnuma highly scalable server," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ISCA '97, (New York, NY, USA), p. 241–251, Association for Computing Machinery, 1997.

[58] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 168–179, 2018.

[59] C. Trippel, D. Lustig, and M. Martonosi, "Meltdownprime and spectreprime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols," 2018.

[60] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (sharp): Defending against cache-based side channel attacks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 347–360, 2017.

[61] 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.

[62] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," MICRO-51, p. 428–441, IEEE Press, 2018.

[63] M. Yan, J. Wen, C. W. Fletcher, and J. Torrellas, "Secdir: A secure directory to defeat directory side-channel attacks," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 332–345, 2019.

[64] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *IEEE Micro*, vol. 30, no. 2, pp. 16–29, 2010.

[65] A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security assurance for system-on-chip designs with untrusted ips," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1515–1528, 2017.

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

[67] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *24th USENIX Security Symposium (USENIX Security 15)*, (Washington, D.C.), pp. 865–880, USENIX Association, Aug. 2015.

[68] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[69] E. Witchel, J. Cates, and K. Asanović, "Mondrian memory protection," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, (New York, NY, USA), p. 304–316, Association for Computing Machinery, 2002.

[70] B. W. Lampson, "Protection," *SIGOPS Oper. Syst. Rev.*, vol. 8, p. 18–24, Jan. 1974.

[71] E. Witchel, J. Rhee, and K. Asanović, "Mondrix: Memory isolation for linux using mondriaan memory protection," *SIGOPS Oper. Syst. Rev.*, vol. 39, p. 31–44, Oct. 2005.

[72] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe, "The cheri capability model: Revisiting risc in an age of risk," *SIGARCH Comput. Archit. News*, vol. 42, p. 457–468, June 2014.

[73] K. Koning, X. Chen, H. Bos, C. Giuffrida, and E. Athanasopoulos, "No need to hide: Protecting safe regions on commodity hardware," in *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, (New York, NY, USA), p. 437–452, Association for Computing Machinery, 2017.

[74] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, (New York, NY, USA), p. 2–13, Association for Computing Machinery, 2008.

[75] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, (New York, NY, USA), p. 2–13, Association for Computing Machinery, 2008.

[76] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 857–874, USENIX Association, Aug. 2016.

[77] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "vtz: Virtualizing ARM trustzone," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 541–556, USENIX Association, Aug. 2017.

[78] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020.

[79] D. Kadjo, J. Kim, P. Sharma, R. Panda, P. Gratz, and D. Jimenez, "B-fetch: Branch prediction directed prefetching for chip-multiprocessors," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 623–634, 2014.