CAESAR: Coherence-Aided Elective and Seamless Alternative Routing via on-chip FPGA

Shahin Roozkhosh*§, Denis Hoornaert†§ and, Renato Mancuso*
*Boston University †Technical University of Munich
*{shahin, rmancuso}@bu.edu, †denis.hoornaert@tum.de

Abstract—**Prompted by the ever-growing demand for high-performance System-on-Chip (SoC) and the plateauing of CPU frequencies, the SoC design landscape is shifting. In a quest to offer programmable specialization, the adoption of tightly-coupled FPGAs co-located with traditional compute clusters has been embraced by major vendors. This CPU+FPGA architectural paradigm opens the door to novel hardware/software co-design opportunities. The key principle is that CPU-originated memory traffic can be re-routed through the FPGA for analysis and management purposes. Albeit promising, the side-effect of this approach is that time-critical operations—such as cache-line refills—are fulfilled by moving data over slower interconnects meant for I/O traffic.**

In this article, we introduce a novel principle named *Cache Coherence Backstabbing* to precisely tackle these shortcomings. The technique leverages the ability to include the FGPA in the same coherence domain as the core processing elements. Importantly, this enables Coherence-Aided Elective and Seamless Alternative Routing (CAESAR), i.e., seamless inspection and routing of memory transactions, especially cache-line refills, through the FPGA. CAESAR allows the definition of new *memory programming paradigms*. We discuss the intrinsic potentials of the approach and evaluate it with a full-stack prototype implementation on a commercial platform. Our experiments show an improvement of up to 29% in read bandwidth, 23% in latency, and 13% in pragmatic workloads over the state of the art. Furthermore, we showcase the first in-coherence-domain runtime profiler design as a use-case of the CAESAR approach.

Index Terms—**Coherence Domain, FPGA, Memory Inspection**

I. Introduction

On June 1, 2015 Intel® announced the definitive intention to acquire Altera® with a transaction valued at about $16.7 billion[1]. Altera was at the time one of the two leading companies in the design of Field Programmable Gate Array (FPGA) chips. The other being Xilinx®. The acquisition of Altera by one of the largest players in the general purpose computing world happened only 4 years after Xilinx announced a shift from FPGA-only chips to "all things programmable" Systems-on-Chip (SoC). Meanwhile, the Xilinx UltraScale+ family of CPU+FPGA SoCs has set a new standard for the collaborative co-location of hard and re-programmable logic [1]. With an estimated $50 billion move[2], AMD® announced on February 14, 2022 the acquisition of Xilinx. At the time of writing this article, the two major players in general purpose computing systems have respectively acquired the largest companies involved in the research and development of FPGA technologies.

Programmable specialization. If the recent news is of any indication, CPU+FPGA SoCs are poised to become a standard computing model not only within the embedded market where significant penetration is already happening. But also in the general purpose and high-performance computing segments. One of the driving factors of this transformation is specialization. Following a plateau in CPU speeds, well-established accelerator paradigms such as GPUs and TPUs have filled the gap for *some* data-intensive workloads. FPGAs are just a generalization of the same concept, i.e., the response to a need to accelerate custom computational pipelines.

While the addition of onboard FPGA technology is a natural next-step, we argue that the full extent of its implications in the way we design, engineer, program, and analyze our systems is yet to be determined. In other words, strong implications arise from the co-location of hard and re-programmable logic that goes well beyond the ability to design custom accelerators [2].

Revisiting memory semantics. One such implication is the ability to redefine the semantics of memory operations initiated by any of the processing elements (PE). This capability is of particular interest for the real-time community since the inability to explicitly program and arbitrates the behavior of (main) memory components leads to performance non-determinism and pessimistic worst-case execution time (WCET) estimations. Roozkhosh and Mancuso pioneered this idea in [3], where they demonstrated that FPGA logic can be interposed between CPUs and main memory, following what they refer to as the *Programmable Logic In-the-Middle* (PLIM) approach. Under PLIM, memory transactions originating at the CPUs and targeting main memory are *re-routed* through the FPGA. Custom transformations can then be applied. The original work in [3] demonstrates a use-case where page coloring is hidden from the underlying memory to prevent fragmentation. Since then, other works have surfaced that leverage PLIM to manage the timing and ordering of main memory transactions [4], and perform on-the-fly data re-organization to speed-up access to relational data stores [5].

Bringing fine-grained control over memory operations into the hands of system designers is a major milestone. But existing approaches route memory traffic through the FPGA via memory-mapping semantics. Not only this incurs a fixed and non-negligible performance overhead, but it also limits the management strategies that can be enacted on the FPGA.

Our focus. This paper focuses on hardware platforms in which the FPGA can be cache-coherent with the CPU cluster

[§] These authors contributed equally.
[1] See https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/.
[2] See https://www.amd.com/en/corporate/xilinx-acquisition.

and proposes a basic principle, namely *Coherence Backstabbing*. Coherence backstabbing enables a novel approach for through-FPGA memory traffic re-routing, which we call Coherence-Aided Elective and Seamless Alternative Routing, a.k.a. CAESAR. As the name suggests, CAESAR allows selective re-routing through the FPGA of memory transactions (or their metadata) generated by the CPUs. CAESAR relies on the ability of the FPGA to have a primary role in a cache-coherent interconnect and to intercept/interject coherence traffic. By doing so, the FPGA is logically elevated to sit right after the last-level cache (LLC). Crucially, this unlocks a host of opportunities that significantly complement and enhance the vision of the PLIM approach. We showcase how such a capability can be leveraged to achieve a new degree of FPGA-aided management for CPU-originated memory traffic.

Contribution. This paper makes the following contributions.

(1) Theorizes and demonstrates for the first time previously untapped opportunities arising from the ability of FPGA logic to maintain two-way cache coherence with a CPU cluster. Two-way coherence enables the FPGA to be seamlessly interposed between CPUs and memory to manage CPU-originated traffic, which is not possible with one-way coherence [6]

(2) Proposes a fundamental technique called *Coherence Backstabbing* to elevate the FPGA to become a first-class citizen in the memory hierarchy of CPU+FPGA SoCs;

(3) Proposes proof-of-concept designs that leverage coherence backstabbing to implement the CAESAR approach, i.e. Coherence-Aided Elective and Seamless Alternative Routing;

(4) Showcases a full-stack system implementation that directly compares CAESAR to the work in [3] to discuss the performance and paradigm enhancements unlocked by the proposed CAESAR approach.

The rest of the article is divided in seven sections. We outline the big picture in Section II and Section III introduces key background principles. Section IV defines the concept of coherence backstabbing and outlines the CAESAR approach. In Section V, we provide in-depth technical details on our prototype implementation. We evaluate the potential of CAESAR for real-time system in Section VI. A discussion of how CAESAR unlocks additional programmability of memory semantics is discussed in Section VII. Closely related research is surveyed in Section VIII. The paper concludes in Section IX

II. Motivation and Vision

The introduction of multiple CPUs, accelerators, and high-bandwidth I/O devices that can initiate memory transactions has had a transformational impact on the design and analysis of real-time systems. In contrast, the performance improvements offered by off-chip memory technology have not kept the same pace. This has made the memory subsystem the main performance bottleneck. In a bid to hide latency, hardware designs have increased in complexity: multiple layers of on-chip caching mechanisms, complex interconnect fabrics, and sophisticated off-chip memory controllers are some of the design paradigms that have become commonplace in embedded and general-purpose systems.

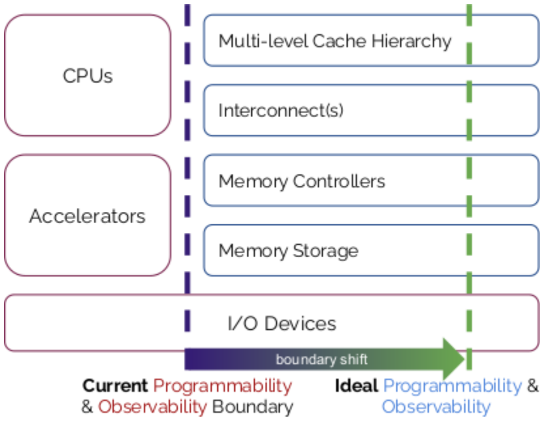

CPUs
Accelerators
Multi-level Cache Hierarchy
Interconnect(s)
Memory Controllers
Memory Storage
I/O Devices
boundary shift
Current Programmability & Observability Boundary
Ideal Programmability & Observability

Fig. 1: CAESAR vision: leveraging onboard FPGA to shift the programmability boundary to include hardware resources in the memory hierarchy.

Unfortunately, the explosion in complexity has had a detrimental effect on our ability to understand and manage the temporal behavior of data movements through the memory hierarchy. The performance unpredictability that emerges in modern platforms can be traced back to the ubiquitous compute-oriented paradigm that has driven the jump from single-core to multi-core and (more recently) to accelerator-enabled platforms. As exemplified in Fig. 1, this has defined a *programmability boundary* where CPUs, accelerators, and certain I/O devices (e.g., smart network interface cards) offer explicitly observable and programmable operational semantics. Conversely, the memory hierarchy components offer limited (or non-existent) programmability that often materializes only as a set of configuration knobs. Observability is also limited to performance counters designed to be accessed by software on the CPUs. As such, strategies that attempt to cope with the unpredictability of modern memory hierarchies—be it WCET analysis tools or OS-level resource management strategies—must overcome three main hurdles.

(1) Opacity. Unlike the extensive documentation available to describe the operational semantics of CPUs and accelerators, significantly less information is released by vendors regarding the exact behavior of memory components and their interplay. Often, the such interplay is not well understood, let alone documented, and left to be uncovered via reverse engineering.

(2) Lack of Control. Even if the behavior of a given component is well understood, affecting it in a meaningful way via CPU-centric management remains a challenge. For instance, it is known that predictability benefits can be achieved by mapping a disjoint set of DRAM banks to CPUs [7], [8]. Nonetheless, it is non-trivial to rework physical memory allocation to implement DRAM bank partitioning. And, not unlike page coloring-based cache partitioning [9], the side-effects of resource partitioning can outweigh its benefit [10].

(3) Lack of Programmability. Even when some degree of control can be exerted, programmability hardly extends beyond parameter tuning. Conversely, being able to program exact management policies directly into main memory components is currently only possible via custom hardware re-design.

The main research question tackled in this work is: *What are the implications of FPGA-CPU interaction via cache coherence on the ability to shift the programmability boundary?* With reference to Fig. 1, we investigate what new programming paradigms are available to observe and/or act upon the logical and temporal semantics of memory operations.

III. System Model and Background

This section describes the system model and assumptions. We review key background concepts necessary to understand the presented research, implementation, and results.

A. CPU+FPGA System Model

We consider CPU+FPGA platforms where a computing cluster (CPU) is instantiated in silicon together with a block of programmable logic (FPGA). While we primarily refer to the presence of CPUs within the computing cluster, we make no assumption on the exact composition of the cluster. In other words, the cluster might contain a single or multiple CPUs and/or accelerators. Other works refer to the same architectural paradigm as heterogeneous partially reprogrammable platforms [3], [11], [12], or PS+PL platforms, where PS and PL refer to the Processor Subsystem and Programmable Logic.

For simplicity, we assume all the processors within the computing cluster share a single LLC. The downstream memory hierarchy components must resolve LLC misses. We consider a physically-indexed, physically-tagged (PIPT) LLC that implements a write-back write-allocate (WBWA) policy. By WBWA, dirty lines are written back upon eviction, and a write miss allocates a new line in the LLC.

Memory-mapped FPGA. We assume that the FPGA is assigned a static aperture within the physical addressing space. When CPU-originated transactions have a physical address that falls within the FPGA aperture, a miss in LLC initiates a read memory transaction (and potentially a write-back) towards the FPGA. We refer to this approach to initiate CPU-to-FPGA communication as *memory-mapped semantics.*

High-performance CPU-to-FPGA communication. When transactions are initiated towards the FPGA, they are carried over (a set of) dedicated high-performance CPU-to-FPGA bus segments. We further assume that the onboard FPGA is capable of initiating transactions towards the main memory (DDR). Transactions initiated by the FPGA towards main memory travel on bus segments of comparable performance.

B. Advanced Extensible Interface (AXI)

On ARM-based CPU+FPGA systems, which are the immediate target of this work, it is reasonable to assume that on-chip memory streams are carried out using the Advanced Extensible Interface (AXI) protocol [13] or extensions of thereof. The protocol exist in three variants: *Lite*, *Stream*, and *Full*. The latter is the object of this section.

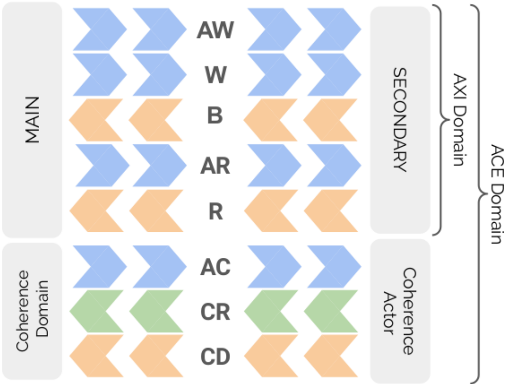


Fig. 2: Overview of AXI and ACE channels.

The AXI protocol relies on two key concepts: (1) the *Main-Secondary*[3] duality and (2) the *handshake* mechanism in order to carry out read and write operations. An AXI bus segment is a point-to-point connection directed from the main port to a secondary port. The main port can initiate a request, while the secondary port serves requests initiated by the main.

The handshake between main and secondary allows (1) the former to indicate the validity of the current data on the bus lines; and (2) the latter to indicate whether it can accept the data. This mechanism is crucial to enable the asynchronous nature of the protocol and decouple the main design from the secondary response time.

AXI channels. To carry out read transactions, two channels are defined, as illustrated in Fig. 2. First, the main initiates an address phase on channel **AR**; the secondary responds on the **R** channel with the requested data payload. Write transactions, as depicted in Fig. 2, proceed in three steps. The main initiates an address phase (**AW** channel) before initiating an arbitrarily long data phase on the **W** channel. It is only once the data phase is over that the secondary responds via channel **B** to acknowledge the completion of the transfer.

C. Programmable Logic In-the-Middle (PLIM)

The work in [3] describes the possibility of logically interposing the FPGA between then CPUs and main memory by (1) using memory-mapped semantics to assign physical memory within the address range corresponding to the FPGA aperture; and (2) by fulfilling CPU-originated memory requests by using the FPGA to access main memory and forward the data. During step (2), data, address, and timing manipulations can be applied before returning the final data items.

Example. Consider 1 GB of main memory and an FPGA mapped at the aperture [0x0000_0000, 0x3fff_ffff] and [0xc000_0000, 0xffff_ffff], respectively. The CPU wants to access address 0x1234_5000. Its page tables are modified to use the physical address 0xd234_5000 with an added constant offset 0xc000_0000. The request reaches the FPGA, which subtracts the offset, retrieves the data from main memory and returns the result.

[3] The authors do not endorse the *master* and *slave* terminology used in the official referenced manuals. The keywords *main* and *secondary* are used instead throughout the article.

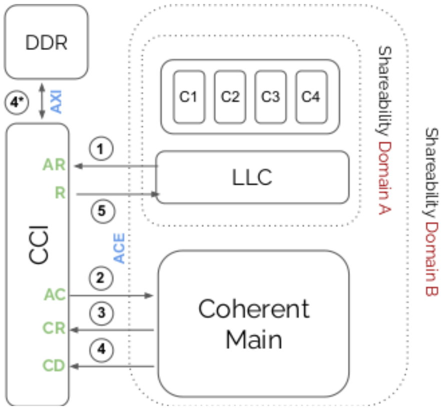


Fig. 3: Snoop-based coherence via the ACE protocol.

Doing so effectively creates a *secondary route* to the main memory. Memory traffic is routed via a cacheable I/O-like path through the on-chip FGPA. This is different from the customary use of hardware accelerators that operates following a load-unload fashion [2]. The concept of PLIM applies to a ever-broadening class of FPGA+CPU commercial systems from vendors such as Intel [14], Xilinx [11], and Microsemi with their PolarFire SoC [15], while large-scale research-grade prototypes are being studied [16], [17].

D. Cache Coherence Model

In complex systems, multiple components might cache data. Coherence protocols allow caching to be distributed across the SoC, while a consistent view of data items is offered to all the parties participating in the protocol. In this work, we assume a snoop-based cache coherence protocol facilitated by a central cache-coherent interconnect (CCI), as depicted in Fig. 3. If caches in two or more subsystems (e.g., CPU cluster and FPGA) are kept coherent, they are said to belong to the same *coherence domain*. In ARM-based platforms, this is referred to with the term *shareability domain*.

We assume that the LLC of the compute cluster is a cache-coherent main attached to the CCI. Via the CCI, the content of the LLC can be kept coherent with other caches defined outside of the compute cluster. Note that we make no assumption about the protocol used by CPUs and accelerators within the compute cluster (Shareability Domain A in Fig. 3). Beyond assuming a snoop-based coherence protocol used by the mains attached to the CCI, no specific protocol is assumed. However, to ensure broader applicability, we assume that the protocol supports write-back caches and data passing between caches (e.g., the Illinois [18] protocol, a.k.a. the Modified-Exclusive-Shared-Invalid or MESI). More specifically, we require that (1) upon a read/write miss in the LLC, a snoop request is broadcasted by the CCI; and (2) when responding to a snoop request, other mains on the CCI can directly pass the most recent copy of the requested cache line.

Cache-coherent FPGA. Mains within the same domain must expose appropriate interfaces to follow the same cache coherence protocol. Importantly, we assume that the FPGA defines one such interface and that it can be included within the same coherence domain as the LLC of the compute cluster. This is the case for the "Coherent Main" block in Fig. 3, since the LLC and said block are both parts of Shareability Domain B.

E. AXI Coherency Extensions (ACE)

In ARM-based platforms, the CCI generally implements the AXI Coherence Extension (ACE) protocol, as depicted in Fig. 3. ACE extends the AXI protocol by adding three channels to support hardware-assisted cache coherence, as illustrated in Fig. 2. In the ACE protocol, upon a load/store cache-miss at the LLC ①[4], the CCI broadcasts the event to all the coherent mains ②. It uses the snoop address (**AC**) channel as a medium to provide the snoop transactions' address and associated control information. Every snoop transaction has a single response associated with it, which (when appropriate) can be used by a snooped main to provide the data for the requested cache line. The snoop response (**CR**) channel is used to provide one such response ③. In case the snooped request can be served by another coherent main, the snoop data (**CD**) channel ④ is used to transfer the data payload. If the LLC cache miss is not resolved via coherence, the CCI satisfies the cache line fill from DDR ④*.

F. Software Layer

We assume the platform software uses cacheable and shareable memory. Beyond that, we make no assumptions regarding the software layers. The hardware mechanisms we discuss function independently from the software layers. As such, we assume unmodified user-space applications, OS, and hypervisor layers.

IV. CAESAR Design and Paradigms

This section discusses the proposed cache coherence backstabbing principle and outlines how it can be leveraged to enable the CAESAR approach.

A. Coherence Backstabbing

We define *coherence backstabbing* as the principle of leveraging the cache-coherent interface(s) exposed to the FPGA for purposes other than maintaining cache coherence within the shareability domain. We use the word *backstabbing* to refer to the idea that the FPGA inserts itself on the back of a protocol where (1) useful information about the behavior of the CPUs is transferred (**observability**); (2) the state of the upstream caches, and the semantics and timing of the downstream data accesses can be impacted following user-defined logic (**programmability**). When performing coherence backstabbing, it is vital that the correctness of the protocol is maintained.

Backstabber designs. A *backstabber design* (or *backstabber IP*) is an FPGA design that performs coherence backstabbing. I.e., it maintains protocol correctness while implementing custom functionalities with different degrees of intrusiveness.

[4] Note that both a load and a store cache-miss will appear as a read transaction on the **R** channel of the ACE interface due to the WBWA policy. In ARM Aarch64, only non-temporal loads/stores (`LDNP`/`STNP`), which are seldomly used instructions, can cause a different behavior.

Backstabbed interface. On the other hand, a memory interface (e.g. a memory-mapped aperture) is said to be *backstabbed* if a backstabber design is actively or passively interacting with the traffic directed to the interface employing coherence backstabbing.

B. The CAESAR Approach

We propose to use coherence backstabbing as a building block for what we define as the Coherence-Aided Elective and Seamless Alternative Routing (CAESAR) approach. The key features of CAESAR are the following. First, it enables memory traffic to follow routes that are alternative to those that would be followed with traditional memory-mapped semantics (see Section III-A). Because such a route bypasses slower SoC interconnects meant for I/O traffic, it also enables lower re-routing overheads. Second, it allows re-routing of data and metadata with different trade-offs between seamlessness and resulting overheads. Third, it allows dynamic activation of the alternative route (for cache-line refill traffic) without the need to modify a page-table translation. From the properties above, we hereby describe a set of immediate memory programming paradigms enabled by the CAESAR approach.

Silent Observer. Once a CPU undergoes an LLC miss, the CCI broadcasts a snoop to a CAESAR IP implemented on the FPGA. The most immediate programming paradigm consists of single-ended listening without active participation. We refer to this paradigm as the *silent observer*. This paradigm supports designs that only need to rely on collecting meta-data of the traffic produced by the CPUs. In this case, only transfer metadata are being re-routed, but not actual data. For instance, the silent observer paradigm can be used to acquire a precise trace of such traffic, as we investigate in Section V-C. This is arguably the approach that incurs the minimum overhead.

Read-Only Re-routing. Intuitively, this paradigm follows the *"I do not have the data, but I am going to lie about it"* principle. The CAESAR IP behaves as an active component in the coherency domain. Upon receiving a snooped LLC miss, it can take charge of providing the corresponding data. In other words, the IP responds by *simulating* a cache-hit on its side.

To maintain correctness w.r.t. the coherence protocol, it is now the burden of the IP to pass the data. From here, the IP has two broad choices. First, it can directly interface with a memory storage device (e.g., main memory) to fetch the data (not necessarily at the same address!), or it can make up the data by constructing it on the fly. A combination of both approaches is also possible.

Consequently, this approach allows re-routing read traffic to the FPGA without requiring the modification of the underlying physical addresses. The CAESAR IP discussed in Section V-B precisely explores the read-only re-routing potential.

Read-Write Re-routing. Under the WBWA semantics, if the LLC follows a MESI [18] coherence protocol, only two cases cause snoops to be broadcasted by the CCI. (1) Transition to Modified state upon write-hits on a cache line in Shared state; (2) cache refills caused by a read/write LLC miss (i.e., transitions to Shared, Modified, or Exclusive state from Invalid state). Conversely, write-backs (resulting in write memory transactions) are only triggered upon the eviction (or clean+invalidation) of dirty LLC cache lines. Therefore, write-backs are not snooped and the CCI forwards the transaction to the downstream component depending on the physical address of the cache line to be written back.

As such, re-routing of write traffic requires modifying the virtual-to-physical translation of the data to be re-routed via the FPGA. In this case, a CAESAR IP will handle snoop requests for cache refills like in the previous paradigm. Write traffic (and only write traffic!) will reach the FPGA following traditional memory-mapped semantics instead. This paradigm allows the FPGA to act upon the entirety of memory traffic, while still handling read requests faster than using memory-mapped semantics for both read and write traffic. Because reads are located on the critical path from the point of view of CPUs' performance, doing so still achieves noticeable performance improvements, as we evaluate in Section VI-C.

If, instead of re-routing write-backs, one wanted to expose to the FPGA which cache-lines are modified by the CPUs, for instance, to analyze read/write patterns and false-sharing, another approach can be used. Specifically, in cases where the LLC controller follows a MOSI [19] or MOESI [20] protocol, a write-hit on a cache-line in Owned state causes a snoop request with the most recent data payload to be broadcasted by the CCI. The FPGA could theoretically force the state of the cache lines in the LLC for which read/write traffic needs to be re-routed to the Owned state. Exploring this direction is currently out of scope.

Correctness and Generalization. The system model described in Section III-A is aligned with the architectural organization of existing CPU+FPGA systems with a single CPU cluster and a CCI connecting its LLC to the FPGA. In this case, the occurrence of a miss in the LLC is sufficient to guarantee that the most updated version of the requested data item is in main memory. Thus, a CAESAR IP can safely reroute data fetches by replying to the corresponding snoop transactions without breaking cache coherence. Nevertheless, it is foreseeable that future CPU-FPGA architectures might feature multiple CPU clusters and FPGA modules kept coherent through the same CCI. Performing coherence backstabbing in such architectures will require a generalization of our approach, the details of which currently lie outside of the scope of this paper. The intuition, however, is that in multi-cluster SoCs, a CAESAR IP (or equivalent) will need to implement a full coherence protocol. With that, through-FPGA handling of memory requests can be initiated only when the state of a cache line is safely determined to be in the "invalid" state in all the CPU-side caches.

V. Practical Instantiation

This section describes a practical full-stack implementation carried out on a commercially available CPU+FPGA platform that follows the model and assumptions outlined in Section III.

A. Target Platform

The implementation of a set of CAESAR prototype designs and their evaluation has been conducted on Xilinx UltraScale+ MPSoC, specifically the ZCU102 development board. The ZCU102 is a CPU+FPGA platform equipped with a compute cluster grouping four ARM Cortex-A53 cores running at 1.5GHz and a shared LLC of 1MB. The sizeable onboard FPGA (600k+ LUTs) exports two high-performance (HPM) CPU-to-FPGA ports with memory-mapped semantics. The FPGA can initiate direct transactions to the main memory.

Cache-coherent FPGA. Importantly, the FPGA is also connected to the system's CCI via an ACE port, which places the FPGA in the shareability domain of the LLC (see Section III). The CCI used in this platform is an instance of a standard ARM CCI-400 component [21]. Usually, the CCI-400 is used to connect two clusters of heterogeneous cores following the ARM big.LITTLE [22], [23] architectural paradigm. Instead, on the ZCU102, the second cluster is replaced by an FPGA. We selected Xilinx's ZCU102 as the target platform for evaluation to be immediately applicable and comparable to existing works [3], [4], [12]. The overhead for involving the FPGA in the coherence domain is evaluated in VI-A.

In this platform, the CCI supports not only coherent data caching but also Distributed Virtual Memory (DVM) [13]. DVM is used to keep translation look-aside (TLB) entries coherent across the SoC if multiple entities implement memory management units (MMU) with cached virtual-to-physical address translations. DVM management causes specific snoop requests on the ACE port that any CAESAR IP must correctly handle. There are two classes of DVM transactions that need to be handled by the IP. (1) DVM operations: these transactions convey particular operations, such as TLB invalidation requests. (2) DVM sync: a synchronization transaction that component issues to check that all previous coherence-related requests have been completed. DVM Sync requests are served to enforce barrier-like semantics for TLB operations beyond the boundaries of the CPU cluster.

The CCI is configured at the power-up by the ARM Trusted Firmware (ATF). We modified the BL31 stage (resident bootloader part of the ATF) to activate the FPGA-facing ACE port. The BL31 bootloader stage executes right after the FPGA bitstream is loaded and before booting any OS or hypervisor. This is safe to do as long as a valid backstabber IP is contained in the programmed bitstream. From the moment when the coherence domain is extended to include the FPGA, the CCI expects the FPGA to correctly handle all the snoop requests, including the DVM snoops. As such, a litmus test for the correctness of the design is the ability of the system to complete the boot sequence that includes a full Linux kernel boot and (potentially) that of a hypervisor. Indeed, a burst of cache and TLB maintenance operations are issued during the boot sequence. Hiccups in the way the instantiated IP participates in the coherence protocol cause a fail-stop failure of the system.

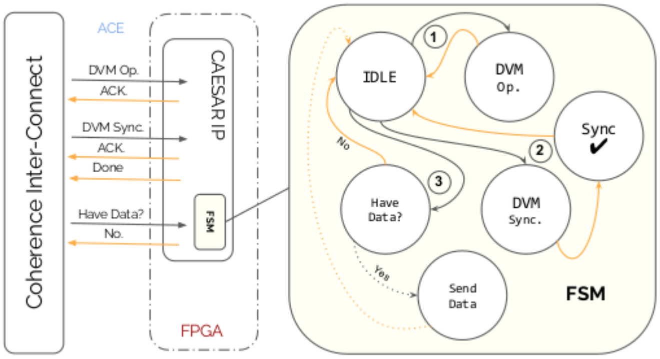


Fig. 4: Minimalist design of a coherence-enabled CAESAR IP: the Silent IP. Interaction with the CCI is depicted on the left; finite state machine (FSM) is reported on the right.

B. Template CAESAR IPs.

CAESAR Silent IP. To the best of our knowledge, the only publicly available ACE-enabled IP that can be instantiated on the ZCU102 platform is the Xilinx System Cache [24], which implements the semantics of an accelerator-facing cache. However, the sources of the System Cache are not available. Thus, we implemented *from scratch* a first template design for an IP that has the bare-minimum logic to handle the coherence protocol correctly. The internal organization of this component, called the *Silent IP*, is depicted in Fig. 4

Looking at the left-hand side of Fig. 4, the IP must handle three prominent cases, as reflected in the finite state machine (FSM) states/transitions on the right-hand side. ① If a DVM operation is received, the IP must acknowledge it, but no active response is necessary. ② If a DVM synch is received, the IP must acknowledge it, actively provide a reply message (DVM complete), and finally exchange a round of acknowledgments with the CCI. ③ If a regular cache snoop is received, the IP must simply acknowledge the transaction and reply that it does not own a more recent version of the requested cache line.

CAESAR Read-Only IP. Next, we implemented a CAESAR IP capable of performing read traffic re-routing over a configurable range of physical addresses. Whenever the address of a data snoop request falls within the configured range, the IP commits to provide the data. A birds-eye view of the implementation is provided in Fig. 5. The orange data path refers to the scenario in which the IP accepts a request (lying scenario, see Section IV).

All snoop transactions are strictly ordered. I.e., responses on the snoop response channel (**CR**) must be issued in the same order they arrived on the snoop address channel (**AC**). Via three dedicated signals in the **CR** channel, the IP notifies the interconnect about the state of the cache line of data to be sent. These are `PassDirty`, `IsShared` or `WasUnique` [13]. If the IP asserts the `PassDirty` signal, it passes to the LLC the cache line data, but the line will be written back by the LLC upon eviction. This is not ideal because the IP could force more write-backs than necessary. If the `IsShared` is asserted, additional snoop traffic will be generated if a store is issued on the same line. Finally, asserting the `WasUnique`

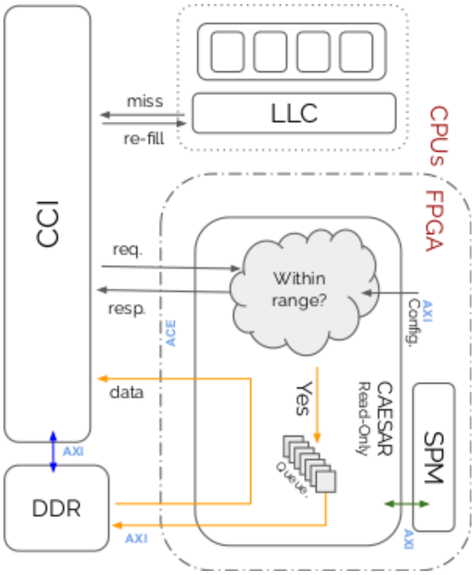


Fig. 5: CAESAR Read-Only architecture. AXI to ACE route (orange) compared to direct AXI route (blue). The read data can be sent to the SPM (green route)

signal means that no other cache holds a copy of the cache line and allows the snooping process to complete. Thus, our IP uses the combination `PassDirty` = 0, `IsShared` = 0, and `WasUnique` = 1 for any cache line fetch being re-routed. If the IP commits to providing a cache-line, it then issues an AXI request to the main memory and forwards the AXI response data on the ACE data channel (orange route in Fig. 5).

Alternatively, the implemented IP can fetch the data from any other AXI secondary, such as an on-chip scratchpad (BRAM) memory (green route in Fig. 5). We performed latency and bandwidth analysis of going through the highlighted routes compared to the direct classic route (blue route), where the CCI issues its own AXI request towards the main memory. We present these experiments in Section VI-B.

When the CCI handles a cache-line refill, by default, it also performs a speculative prefetch to main memory. This makes sense as it is likely that a copy of the accessed cache-line is not present in any other cache. Speculative fetching causes the CCI to issue a downstream fetch in parallel (via the blue route in Fig. 5) with the issuance of a coherent snoop transaction. The speculative reads are discarded if a response on the snoop channel is received. Hence, we disabled speculative fetches through the CCI Control Override Register to prevent these parallel memory accesses and let the backstabber have full control over the DRAM. The effect of disabling speculative fetches on the system's performance and latency depends on the benchmark access pattern. It reduces latency when the probability of a snoop-miss is high but introduces extra access to DDR if the snoop hits the cache. On the other hand, disabling speculative fetching may relieve the load on the DRAM if the prefetched data is discarded often. We investigate the impact of disabling CCI prefetches in Section VI-A.

CAESAR Read-Write IP. Finally, we implemented a CAESAR IP capable of performing both read and write traffic re-routing over a configurable range of physical addresses. Write-backs must be re-routed by modifying the page tables used by the CPUs. In particular, the physical address must fall within the range of the exposed CPU-to-FPGA port aperture (HPM). To do so, it suffices to perform a linear translation by adding a constant offset that is later removed by the IP when it interacts with the DDR interface. Conversely, cache refills are handled similarly to the read-only case.

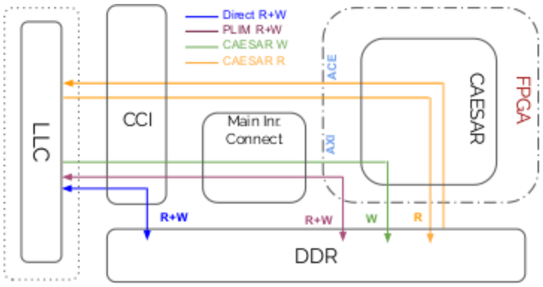


Fig. 6: CAESAR routes (orange, green) for memory traffic compared to existing routes (blue, purple).

Fig. 6 summarizes all the routes described so far. The blue arrow represents the *normal* route to the main memory. The purple arrow refers to the case where re-routing is performed only using memory-mapped semantics (like in the PLIM case [3]). The orange route refers to the route followed by read traffic in a read-only or read+write CAESAR design. In this case, it can be noted that the –slow– main interconnect is bypassed. Lastly, the green route is followed by writes (LLC write-backs) traffic in a read+write CAESAR design.

C. *CAESAR use-case: Silent Profiler*

Beyond improving well-known applications of through-FPGA traffic re-routing, such as those mentioned in Section I, we hereby showcase a novel memory traffic inspection technique. Specifically, we use the privileged position of the FPGA in the SoC to capture and construct a trace of LLC cache misses with minimal impact on their timing. For this purpose, we implemented the Silent Profiler IP.

Essentially, the silent profiler is an altered instantiation of a silent observer as defined in Section IV. The IP pushes key meta-data extracted from the snooped transactions (i.e., timestamp and physical address) in an internal queue. The IP can be activated on-demand from user space and configured to redirect the queued meta-data to any write-able memory, including another FPGA IP, a scratchpad memory, or main memory. Because this IP follows the *silent observer* paradigm (see Section IV), the timestamped trace of memory accesses is acquired with minimum overhead, as we evaluate in Section VI-A. A closer look at the type of introspection enabled by the silent profiler is discussed in Section VI-D.

VI. Evaluation

This section evaluates coherency backstabbing via multiple CAESAR prototypes implementing the designs described in Section V. The section is divided in four parts. First, in Section VI-A, we assess the impact of including the FPGA within the cache coherence domain. Then, in Section VI-B, we investigate the intrinsic raw performance gains offered by the coherence backstabbing approach for various memory targets. Finally, in Section VI-C, we identify the performance positioning of CAESAR designs compared to their state-of-the-art counterparts. In Section VI-D, we provide a first demonstration of the in-coherence-domain profiling capabilities of the silent profiler on a set of widely used pragmatic workloads.

Throughout the evaluation, we use the platform described in Section V, i.e., the Xilinx UltraScale+ ZCU102 development board. Unless stated otherwise, the FPGA always operates at 300MHz, the maximal frequency on the target platform. As for the software layers, we use an unmodified Linux kernel 4.14 to operate the system and host the workloads of interest. Exceptionally, for the experiments conducted in Section VI-C, we leverage the virtualization capabilities offered by the target platform. In such cases, we use the Jailhouse hypervisor [25]. Note that the combined use of a full-fledged OS (Linux) and the hypervisor-layer (Jailhouse) is not strictly necessary to use the CAESAR IP. Conversely, any software layer capable of explicitly allocating physical memory address ranges for cases such as the CAESAR Read-Write IP can be used.

The designs are evaluated using synthetic and pragmatic workloads. The synthetic workloads are used to characterize the performance of the considered memory targets. We selected the IsolBench [26] suite to extract the latency and bandwidth measurements. On the other hand, to assess the real-world impact of the coherency backstabbing approach, we consider the real-time adaptation of the San-Diego Vision Benchmarks Suite [27] (SD-VBS) that is part of the RT-Bench suite [28]. We focus only on the `VGA` input size for the `disparity`, `mser`, `texture-synthesis`, `stitch`, `tracking`, and `sift` benchmarks. The larger input size (`fullhd`) induces runtimes that makes collecting multiple samples of the same experiments impractical. This is the same reason why the `multi-ncut` benchmark was excluded; the smaller input sizes do not trigger enough memory operations to yield interesting results. The `svm` benchmark does not function reliably in our unmodified system, while `localization` never yields interesting results. Thus, the two benchmarks were also excluded.

A. PL-Snooping Overhead

The extension of the coherence domain to include the FPGA is always required to perform backstabbing. Conversely, disabling the prefetcher at the CCI is optional albeit recommended when actively re-routing traffic through a CAESAR IP. However, doing so incurs some performance degradation. This section seeks to highlight the impact of these mechanisms using synthetic and pragmatic workloads.

For both pragmatic and synthetic workloads evaluations, four scenarios combining the inclusion of the FPGA in the coherence domain and the activation of the prefetching are considered. To focus our study on the delays introduced by the inclusion of the FPGA, a silent observer is implemented.

The characterization of the CAESAR overhead is performed by running tests using IsolBench [26] for each of the configurations presented earlier. The Fig. 8 shows the extent of the overhead w.r.t. the bandwidth (left inset) and the latency (right inset). In this figure, one can observe that, regardless of whether prefetching is enabled, both a bandwidth of 2.6 GBps and a latency of 152 ns can be sustained as long as the FPGA is non-coherent. However, performance degradation appears when the FPGA is added to the coherence domain. In the case where the prefetching is enabled, the bandwidth drop remains limited to 100 MBps and the increase in latency is contained to 5 ns. When the prefetcher is disabled, the bandwidth drops to 1.64 GBps and the latency increases to 215 ns.

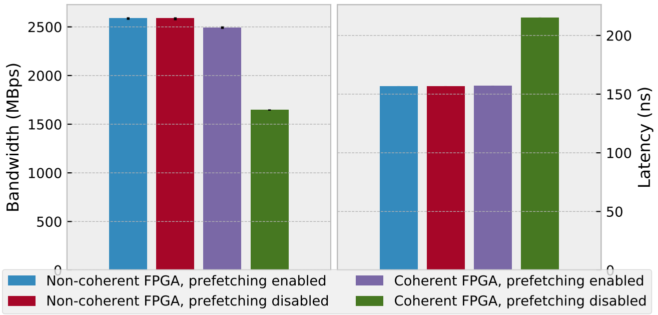


Fig. 7: Bandwidth and latency for designs featuring a combination of prefetching (on/off) and (non-)coherent FPGA.

In addition to the aforementioned four scenarios, the experiment on pragmatic workloads presented in Fig. 8 extends the coherent FPGA scenarios by varying its frequency from 300MHz to 50MHz. Fig. 8 is divided in two insets: one for the execution times obtained with the prefetcher enabled (left) and disabled (right). In both insets, each workload is associated with a bar cluster grouping the recorded execution times for each scenario. Within each cluster, the execution times are normalized over the non-coherent FPGA scenario.

Similarly to the observations made just before, Fig. 8 confirms that pragmatic workloads also suffer from little-to-no interference when the FPGA is not coherent or when the prefetching is enabled. In fact, except for variations smaller than 1%, all scenarios remain reasonably close to the baseline. However, in scenarios with coherent-FPGA, a non-negligible increase in execution can be observed. For instance, memory intensive workloads such as `mser` and `disparity` see their execution time increased by 17% and 6%. Other workloads such as `tracking` and `sift` manage to remain below the symbolic threshold of 5% while non-memory-intensive workloads like `texture_synthesis` and `stictch` remains unaffected. Interestingly, the operating frequency of the coherent FPGA does not seem to impact the raw system performance despite the prefetching mechanism being disabled.

B. Latency and Bandwidth Comparison

We evaluate the improvements enabled by the inclusion of various FPGA-located memory targets in the coherence domain under several levels of contention. We consider four targets with diverging profiles that have been used in closely related works. In addition to the CPU-side DRAM controller, this includes the (1) FPGA memory block primitive (or

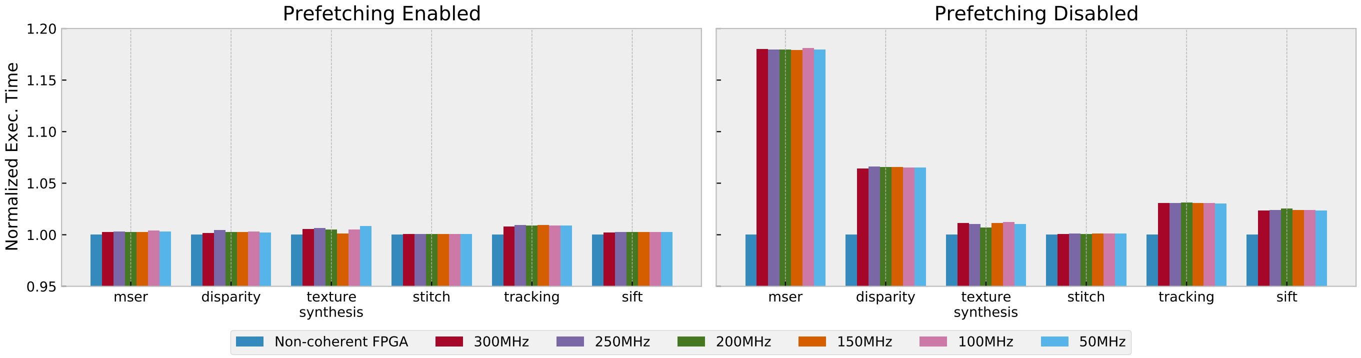


Fig. 8: SD-VBS normalized exec. times with prefetching enabled (left) and disabled (right) using a silent observer IP synthesized at various FPGA frequencies. The normalization baseline is the case with non-coherent FPGA.

BRAM, or SPM [12]) with low-latency and low memory-level parallelism (MLP), (2) high-latency, high-MLP through-FPGA memory loopback [3], and (3) low-latency, high bandwidth AXI-Sink. The latter is a custom-made IP accepting and responding with the best timings allowed by the protocol but without providing any meaningful data. Except for *DRAM*, all targets are evaluated in coherent (i.e., the read-only CAESAR IP is used) and non-coherent (i.e., reads are served via memory-mapped semantics like in [3]) mode. For each of these seven configurations, we measure the bandwidth (Fig. 9) and latency (Fig. 10) experienced by the core under analysis (core 3) under four levels of contention. The level of contention is the number of other cores creating read memory activity.

To illustrate all seven scenarios under the four levels of contention, Fig. 9 is organized around seven clusters of four violin plots. The violin plots are composed of three horizontal bars informing on the minimum, average, and maximum bandwidth while the envelope shows their distribution. As expected, the bandwidth decreases according to the contention level. However, the bandwidth drops and the measurement distributions vary widely depending on the scenarios. Starting from the left, one can quickly observe that direct DRAM access provides the highest bandwidth despite a sharp drop under maximal contention. For lower contention levels, a high average bandwidth can be sustained but at the cost of large variations. For the *BRAM* configuration, the figure highlights the improvements brought by coherence backstabbing. In fact, under no contention, the sustained bandwidth reaches 1.49 GBps, a 29.5% improvement over the non-coherent equivalent. Despite providing higher bandwidths, the gain sharply fades away as the contention increases. Similarly, the *loopback* route (CPU→FPGA→DRAM) consistently displays a higher bandwidth when made coherent with a 949 MBps top bandwidth, improving by 29.5% over the non-coherent version. This observation correlates with the MLP of these memory targets (8, 16, and 32, respectively). The FPGA can use up to 3 direct DDR ports, while we currently use only one. The design of a CAESAR IP capable of multiplexing between them is left as future work. Overall, coherent routes enabled by CAESAR tend to outperform their non-coherent counterparts for lower levels of contention. However, the results are more nuanced under contention, with the bandwidth distributions for comparable memory targets overlapping.

While in slight contrast with the bandwidth results, Fig. 10 mainly mirrors previous observations in the sense that, across the board, the coherent routes always provide lower latencies than their non-coherent counterparts. Not only this observation is true for on-chip targets such as *BRAM* and *AXI-Sink*, but it is also the case for *loopback* routes. Unfortunately, the performance gap decreases as contention increases. Interestingly, the *loopback* route seems to be the most resilient to contention. In fact, *"Coherent loopback"* displays a distinguishably lower latency when running alongside up to two contenders in comparison to *"Non-coherent loopback"*. In contrast, both *BRAM* and *AXI-Sink* offer a marginal improvement in latency for the same level of contention. On higher contention levels, only smaller improvements are observed. Unlike the previous experiment, the *DRAM* route is not providing the lowest latency, an additional hint that the DRAM's higher MLP contributes to the high bandwidth observed in Fig. 9.

In the case of partial read-only rerouting, Fig. 9 clearly shows for each type of memory target that access through CAESAR is beneficial for low contention scenarios. Conversely, under high contention, CAESAR only provides improved best and average bandwidth. This correlates with the latency experiments displayed in Fig. 10. Together, these experiments highlight the ability of CAESAR to provide high-bandwidth, low-latency access to local memory targets such as *BRAM*. This could prove valuable in hybrid scenarios where tasks need to access coherent *BRAM* for performance-critical read-only cache lines (e.g., instruction pages), while the rest of the traffic targets main memory directly.

C. Complete Memory Traffic Rerouting

The previous experiments highlight the raw performance benefits of backstabbing FPGA-located memory targets for read transactions, which is the critical path in systems with WBWA LLCs. However, the inability to directly handle write transactions can hinder the benefits previously shown as the write bandwidth and latency remain unchanged from numbers reported in [4], [12]. To assess the exact extent of the impacts, we emulate previous works [3], [4] in rerouting a whole OS through the FPGA and comparing its workload execution times for various path configurations. While not strictly necessary

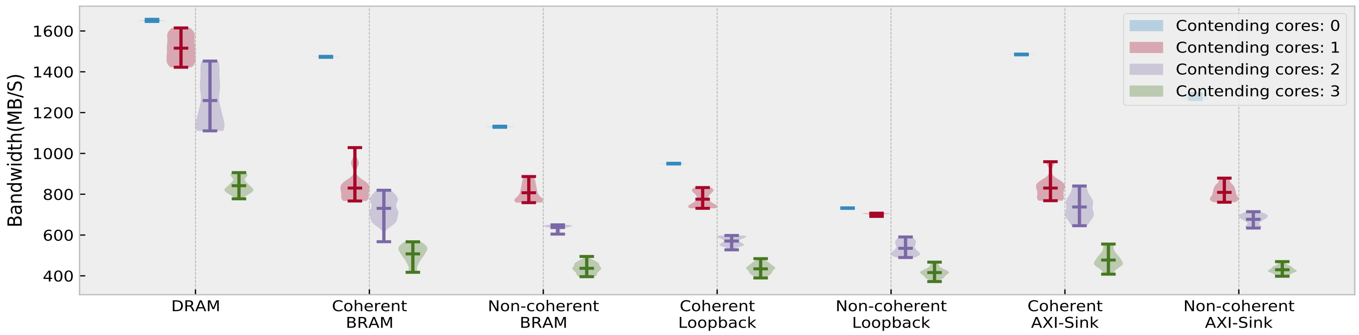


Fig. 9: Bandwidth experienced by a core under multiple levels of contention for various memory targets and routes.

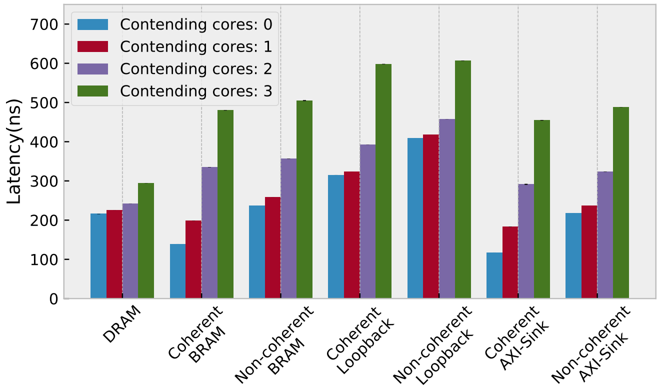


Fig. 10: Latencies experienced by a core under multiple levels of contention for various memory targets and routes.

for the purpose of this experiment, we utilize Jailhouse, a hypervisor, to redirect a complete Linux virtual machine (VM) through the FPGA by allocating it the exposed FPGA aperture. The workloads under analysis run on top of this re-routed VM, guaranteeing the re-routing of both read and write transactions.

Four different path configurations are considered. In the *DRAM* case, the VM is confined to a dedicated part of the DRAM memory. In this configuration, used as a baseline, neither read nor write transactions cross into the FPGA. In contrast, with the *HPM* path, all transactions enter the FPGA through one of the interfacing ports before being relayed to the DRAM, like in [3]. In addition, we consider the *backstabbed* version of these two paths. In these configurations, nothing changes compared to their *non-backstabbed* counterparts except that LLC refills are intercepted and routed by our CAESAR IP through the FPGA. More accurately, *backstabbed DRAM* means that write-backs and uncached transactions remain on the CPU side and directly reach the DRAM controller. Instead, all LLC linefills are handled through the FPGA before arriving to the DRAM controller. The latter configuration is used as a secondary baseline to highlight the impact imputable to the limited write MLP of the *HPM* paths. Similarly, for the *backstabbed HPM* configuration, both writes and uncached read transactions traverse the FPGA via the loopback, whereas linefills are intercepted. This configuration, alongside *HPM*, corresponds to the full memory traffic rerouting case, the main object of this experiment.

Fig. 11 shows how these path configurations impact the execution time of the selected SD-VBS workloads. Each workload (x-axis) is associated with a bar cluster, one bar for each of the above-mentioned path configurations. The height of the bars (y-axis) reports the normalized execution time of the workloads w.r.t. the baseline *DRAM* path configuration.

As expected from the previous experiments, none of the configurations involving the FPGA perform as well as the *DRAM* for memory intensive workloads such as `mser`, `disparity`, `tracking`, and `sift`. The secondary baseline, *backstabbed DRAM*, highlights the impact that rerouting cached reads has, with all workloads seeing their execution time increase. Globally, the *HPM* path configuration always induces the highest execution times recorded, with increases reaching up to 45%, 25%, and 8%. As proven by the numbers for the *backstabbed HPM*, the simple addition of a read-only CAESAR IP for the address range of interest manages substantially reduce the gap with the *backstabbed DRAM* path configuration. In comparison, increases in execution time are reduced to 29%, 15%, and 6%.

Despite the noticeable improvements when compared to PLIM routing (*HPM*), for all benchmarks in Fig. 11, *backstabbed HPM* still introduces large overheads. These can be explained by three limitations in our implementation: (1) the read MLP, (2) the write MLP, and (3) CPU-to-FPGA clock domain-crossing overhead. In the current implementation, read MLP is limited to about 50% of what is theoretically available. This is due to the ACE port being capable of handling up to 32 outstanding reads, while the FPGA-PS AXI port only allows up to 16 outstanding reads. Comparing *DRAM* and *backstabbed DRAM* helps to distill the impact of (1) and (3). In the platform used, write MLP to the FPGA (via the HPM port) is limited to 8, a third of the issuing capability of the core cluster. The overhead imputable to (2) can be appreciated by comparing the *backstabbed* version of *HPM* and *DRAM*.

D. Silent Profiler

This experiment showcases the silent profiler feature by which we are able to capture the bus activity in both memory space and time through its privileged position in the MPSoC. For that, a *Silent Spy* is attached to the coherence domain in *passive collaboration* and with prefetching enabled. Thus, all tasks running on top of Linux reach the DRAM directly. As discussed in SectionVI-A, this introduces negligible interference compared to the system's default configuration.

The traces for `tracking`, `mser`, and `disparity` have been collected and are presented in Fig. 12a, 12b, and 12c.

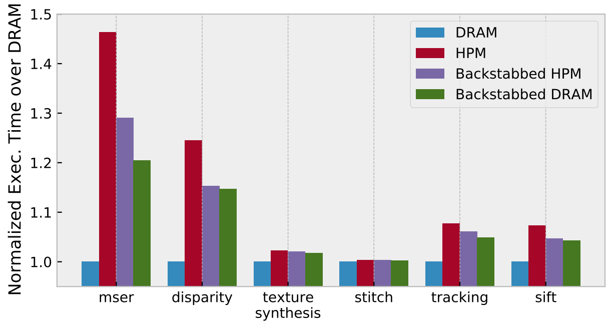


Fig. 11: Workload exec. times for various path configurations.

Each of these figures is composed of two insets. The upper inset shows the cumulative number of memory accesses captured with a high resolution (i.e., transactions are accounted using their exact timestamp) or with a coarser time-bin resolution of $1e6$ clock cycles. The lower inset is a heat map depicting the frequency at which the most popular 100 pages are requested during the workload execution. The time on the x-axis is divided in numerous time bins of $1e6$ clock cycles, while the y-axis represents the overall 100 most popular pages of the workload. Popularity is defined by the number of accesses to a page throughout workload execution. Each cell of the heat map denotes the popularity of these pages during the current time-bin. The color coding of each bin represents the per-bin number of accesses to a given page.

We can observe on Fig. 12a, upper inset, that `tracking` features a "four-steps" curve denoting an alternation of memory and computation intensive segments. The shape resonates with the workload logic consisting in the identification and tracking of an object on four images. Interestingly, while we clearly see in the lower inset the interleaving of memory intense and quiet segments, five intensive segments are observed instead of the four expected. This suggests that the first memory-intensive segment touches a larger range of pages, including the less popular ones not shown in the inset.

As shown by Fig. 12b, the `mser` workload has a more linear progression. It starts with a spike of accesses touching three specific pages out of the most popular ones. Thereafter follows a period of relative inactivity w.r.t. the memory traffic even though the three previously mentioned pages still seem to be particularly hit. After around $1.8e7$ clock cycles, a point of inflection is reached, marking the start of a memory-intensive segment where the most popular pages are hit.

Finally, the trace obtained for the *disparity* workload (Fig. 12c) highlights its memory-intensive nature. Indeed, the upper inset depicts a sustained memory access pattern. The heat map shows a high-frequency repetitive pattern of high and low memory intensity, symptomatic of a workload sweeping through its allocated memory. Despite not being shown in Fig. 12c, we observed that all the top-accessed pages are equally hit. The top-100 selection shown in the figure is likely due to the randomness in the random policy of the LLC.

For the sake of clarity and space, the analysis above is discrete and solely focuses on the 100 most popular pages. However, traces obtained via the *Silent Spy* are substantially more precise as all coherent requests are captured.

VII. Discussion: CAESAR for Real-time Systems

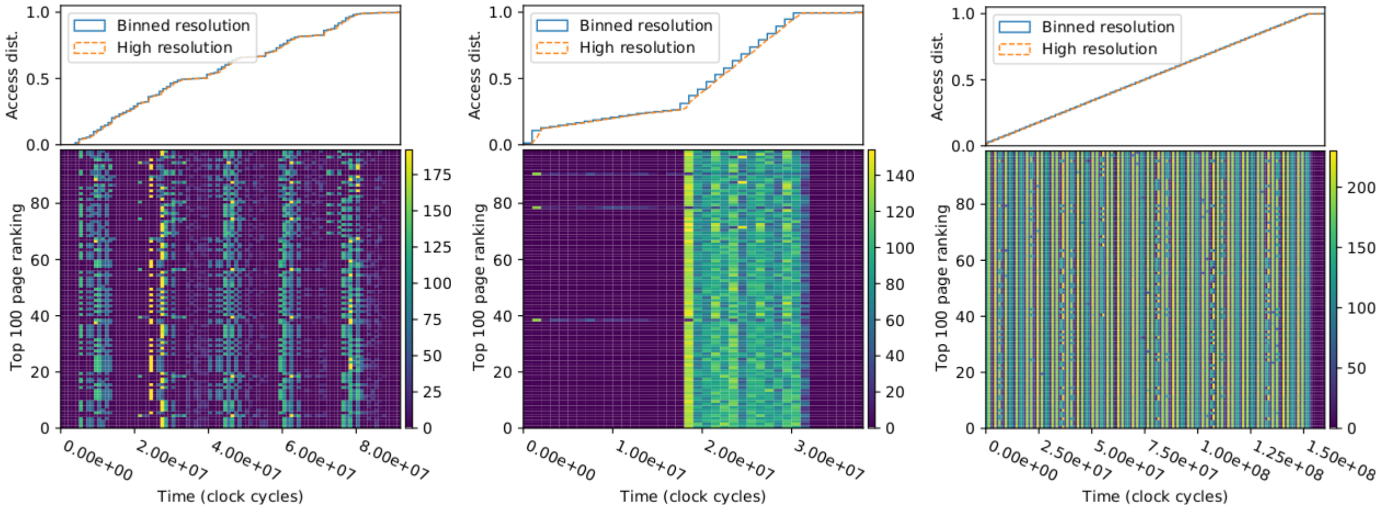
As mentioned in Section II, opacity and lack of control/programmability over memory resources are major roadblocks toward the safe adoption of accelerator-enabled multicore SoCs in real-time systems. The proposed CAESAR demonstrates that a sizable shift in the observability and programmability boundary (Fig. 1) is possible in CPU+FPGA systems, as reviewed in the following.

Observability. The profiling capability presented in Section VI-D sets this work apart from other non-simulation-based profiling tools such as those in [28], [29] that rely on sampling of performance counters. These approaches can approximate the cumulative access pattern (upper insets in Fig. 12) but are unable to capture positional information about which cache lines (or pages) are being accessed. Since the proposed approach is hardware-based, it is also software-independent with negligible overhead as shown in Section VI-A and Fig.7. It opens the doors to FPGA-directed main memory bandwidth management and DRAM bank orchestration.

Compared to using memory-mapped semantics (PLIM) [3], [4], CAESAR IPs provide the same degree of profiling accuracy with a lower performance hit. For instance, the traces in Fig. 4 can also be obtained with *PLIM*. However, as presented in Section VI-A and VI-C, profiling via CAESAR IPs incurs a fraction of PLIM's overhead. Importantly, the magnitude of the improvement depends on the profiling objective. For instance, if profiling cached read requests through a Silent IP is sufficient, a negligible overhead is introduced (see *"Coherent FPGA, prefetching enabled"* in Fig.7). On the other hand, profiling the metadata (e.g., QoS, protection attributes) of cached read transactions implies the adoption of a Read-Only IP. This introduces a larger overhead that is still below what observed with PLIM, as highlighted by Fig. 9 and 10 ("Coherent Loopback") and Fig. 11 ("Backstabbed HPM"). Finally, profiling writes is only attainable by performing a complete rerouting through the FPGA, which introduces the largest (and yet less than PLIM's) cost reported in this article (see Fig. 11, "HPM" vs. "Backstabbed HPM").

Programmability. All the CAESAR IPs described in Section IV hide the complexity of the CPU-side ACE port for ease of FPGA-side programming. The **Read-Only** and **Read-Write** IPs also expose a standard AXI port through which snooped transactions handled by the IP are transmitted. This enables smooth integration with most memory targets and other IPs—e.g., BRAMs [12], Relational Memory Engines [5], cache bleachers [3]. Evaluating CAESAR for such IPs is part of our future work roadmap.

We also envision two new classes of mechanisms leveraging CAESAR capabilities for real-time applications. (1) For read-only traffic, one can selectively and transparently cache in the FPGA certain pages that are deemed important. This is akin to having an explicitly programmable scratchpad that does not need explicit DMA phases. When transactions are not cached, the FPGA re-route is bypassed and the direct route is



(a) Workload: `tracking` (b) Workload: `mser` (c) Workload: `disparity`

Fig. 12: Traces obtained with the *Silent Spy* showing the evolution of memory traffic and physical pages' popularity at run-time.

used instead, which was not possible with traditional PLIM. The logic to decide whether transactions should be cached or not is programmable. (2) For read-write traffic, the FPGA can handle a cache refill by asserting ownership during the CCI-generated snoop. If ownership is asserted (backstabbed access), the route followed by that line refill bypasses the main interconnect and is only queued with other re-routed read requests. If ownership is not asserted (access via memory-mapped semantics), the transaction follows a higher-latency route also followed by write traffic. The logic according to which certain (read) requests should be prioritized in this way over others is programmable.

VIII. Related Work

The CAESAR method presented in this paper fits within a broad vision to define programmable memory subsystems. As computing resources become increasingly bottlenecked by memory resources, the idea of managing the performance and the semantics of memory operations has been embraced by many researchers. For instance, the works on Programming in Memory (PIM) [30] propose the inclusion of key logic operations close to memory cells. In other cases, the ability of the software to augment memory semantics to include ad-hoc operations has been explored in [31]. Rethinking the traditional semantics of memory hierarchies has sparked much interest in disparate sub-fields: from graph acceleration and large tree traversal [32], [33] to garbage collection [34] and file-system redundancy [35]; from (on-the-fly) data transformation [5], [17], [36] to page de-duplication [37] and support to optimize specific access patterns [38], [39].

A large number of works [40] within the last decade have demonstrated that issues with unmanaged contention over memory resources constitute a significant roadblock against predictable consolidation of multi-core, accelerator-enabled safety-critical systems. Many mitigation strategies have been proposed. Software strategies have been investigated to throttle the memory bandwidth used by the CPUs [9], [29], [41] or partition DRAM banks between cores [7], [8]. Available hardware support to regulate accelerators has been studied in [29], [42]. Performing high-level scheduling of computation and clusters of memory accesses was proposed in [43]–[45]. Revisions to traditional hardware components have also been proposed to make them either more configurable, more capable of enforcing QoS, or both [46]–[50]. The work in [6] offers a performance evaluation of one-way coherence ports that are useful for accelerators to remain coherent with CPU caches.

The most closely related works proposed re-routing CPU-originated activity through an FPGA. That is, debug traces [51], [52], or memory traffic [3]–[5]. Unlike the aforementioned related works, CAESAR focuses on the new opportunities offered by two-way coherency between the CPU cluster and the FPGA. Other works have used cache-coherent FPGAs for traditional acceleration purposes such as remote memory accesses [53], arithmetic kernels [54], and machine learning applications [55]. The proposed idea in [53], named PBerry focuses on OS-level awareness of memory accesses page management via FPGA as-a-proxy for networked/remote memory only evaluated in simulation (PBSim). Conversely, we explore the implications of a cache-coherent FPGA for OS-agnostic low-overhead on-chip memory flow programming with a full-stack implementation and real-hardware evaluation.

IX. Conclusion

This paper studies the potential of seamless FPGA interaction with cache-coherence protocols. We presented a novel and previously unexplored set of techniques to manage CPU-originated memory traffic. These techniques are enabled by the ability to logically position the FPGA logic as a top-tier component in the memory hierarchy of modern CPU+FPGA SoCs. We evaluated our proof-of-concept CAESAR approach through a full-stack system implementation considering four designs and corresponding Silent, Read-Only, Read-Write, and Silent Profiler IPs. Our results suggest that remarkable paradigm enhancements and performance improvements are unlocked by the proposed CAESAR method.

ACKNOWLEDGMENT

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CCF-2008799. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF. Denis Hoornaert was supported by the Chair for Cyber-Physical Systems in Production Engineering at TUM and the Alexander von Humboldt Foundation.

REFERENCES

[1] Xilinx. (2016) Zynq UltraScale+ MPSoC - All Programmable Heterogeneous MPSoC. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[2] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 1–12.

[3] S. Roozkhosh and R. Mancuso, "The potential of programmable logic in the middle: Cache bleaching," in *RTAS 2020*, Sydney, Australia, 2020.

[4] D. Hoornaert, S. Roozkhosh, and R. Mancuso, "A Memory Scheduling Infrastructure for Multi-Core Systems with Re-Programmable Logic," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 2:1–2:22. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13933

[5] S. Roozkhosh, D. Hoornaert, J. H. Mun, T. I. Papon, A. Sanaullah, U. Drepper, R. Mancuso, and M. Athanassoulis, "Relational memory: Native in-memory accesses on rows and columns," in *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, J. Stoyanovich, J. Teubner, N. Mamoulis, E. Pitoura, and J. Mühlig, Eds. OpenProceedings.org, 2023, pp. 66–79. [Online]. Available: https://doi.org/10.48786/edbt.2023.06

[6] S. W. Min, S. Huang, M. El-Hadedy, J. Xiong, D. Chen, and W.-m. Hwu, "Analysis and optimization of i/o cache coherency strategies for soc-fpga device," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 301–306.

[7] H. Yun, R. Mancuso, Z. P. Wu, and R. Pellizzoni, "Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 155–166.

[8] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in cots-based multi-core systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 145–154.

[9] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 1651–1657.

[10] N. Kim, B. C. Ward, M. Chisholm, J. H. Anderson, and F. D. Smith, "Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning," *Real-Time Syst.*, vol. 53, no. 5, p. 709–759, sep 2017. [Online]. Available: https://doi.org/10.1007/s11241-017-9272-9

[11] Xilinx, Inc., "Zynq UltraScale+ MPSoC - All Programmable Heterogeneous MPSoC," August 2016, accessed 09.01.2020. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[12] G. Gracioli, R. Tabish, R. Mancuso, R. Mirosanlou, R. Pellizzoni, and M. Caccamo, "Designing mixed criticality applications on modern heterogeneous MPSoC platforms," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[13] ARM, "AMBA AXI and ACE Protocol Specification," Tech. Rep., 2019. [Online]. Available: https://static.docs.arm.com/ihi0022/g/IHI0022G_amba_axi_protocol_spec.pdf

[14] Intel, Corp., "Intel's Stratix 10 FPGA: Supporting the smart and connected revolution," October 2016, accessed 09.01.2020. [Online]. Available: https://newsroom.intel.com/editorials/intels-stratix-10-fpga-supporting-smart-connected-revolution/

[15] Microsemi — Microchip Technology Inc., "PolarFire SoC - Lowest Power, Multi-Core RISC-V SoC FPGA," July 2020, accessed 09.01.2020. [Online]. Available: https://www.microsemi.com/product-directory/soc-fpgas/5498-polarfire-soc-fpga

[16] G. Alonso, T. Roscoe, D. Cock, M. Ewaida, K. Kara, D. Korolija, D. Sidler, and Z. ke Wang, "Tackling hardware/software co-design from a database perspective," in *Conference on Innovative Data Systems Research (CIDR)*, Amsterdam, Netherlands, Jan. 2020.

[17] D. Cock, A. Ramdas, D. Schwyn, M. Giardino, A. Turowski, Z. He, N. Hossle, D. Korolija, M. Licciardello, K. Martsenko, R. Achermann, G. Alonso, and T. Roscoe, "Enzian: An open, general, cpu/fpga platform for systems software research," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 434–451. [Online]. Available: https://doi.org/10.1145/3503222.3507742

[18] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, ser. ISCA '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 348–354. [Online]. Available: https://doi.org/10.1145/800015.808204

[19] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon, "Implementing a cache consistency protocol," in *Proceedings of the 12th Annual International Symposium on Computer Architecture*, ser. ISCA '85. Washington, DC, USA: IEEE Computer Society Press, 1985, p. 276–283.

[20] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[21] Xilinx, "ARM® CoreLink™ CCI-400 Cache Coherent Interconnect," Tech. Rep., 2015. [Online]. Available: https://developer.arm.com/documentation/ddi0470/k/functional-description/snoop-connectivity-and-control

[22] A. L. or its affiliates, "Processing Architecture for Power Efficiency and Performance," Tech. Rep., 2022. [Online]. Available: https://www.arm.com/technologies/big-little

[23] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous multi-processing solution of exynos 5 octa with arm big. little technology," *Samsung White Paper*, 2012.

[24] Xilinx, "System Cache v5.0 LogiCORE IP Product Guide," Tech. Rep., 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/pg118-system-cache

[25] J. Kiszka, V. Sinitsin, H. Schild, and contributors, "Jailhouse Hypervisor," accessed 09.01.2020. [Online]. Available: ttps://github.com/siemens/jailhouse

[26] P. K. Valsan, H. Yun, and F. Farshchi, "Taming non-blocking caches to improve isolation in multicore real-time systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–12.

[27] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "SD-VBS: The san diego vision benchmark suite," in *IISWC 2009*, 2009, pp. 55–64.

[28] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 184–195. [Online]. Available: https://doi.org/10.1145/3534879.3534888

[29] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso, "E-WarP: a system-wide framework for memory bandwidth profiling and management," in *41st IEEE RTSS 2020*, Houston, TX, USA, Dec. 2020.

[30] G. H. Loh, N. Jayasena, M. Oskin, M. Nutter, D. Roberts, M. Meswani, D. P. Zhang, and M. Ignatowski, "A processing in memory taxonomy and a case for studying fixed-function pim," in *Workshop on Near-Data Processing (WoNDP)*, 2013, pp. 1–4.

[31] E. Lockerman, A. Feldmann, M. Bakhshalipour, A. Stanescu, S. Gupta, D. Sanchez, and N. Beckmann, *Livia: Data-Centric Computing Throughout the Memory Hierarchy*. New York, NY, USA: Association

for Computing Machinery, 2020, p. 417–433. [Online]. Available: https://doi.org/10.1145/3373376.3378497

[32] S. Zhou and V. K. Prasanna, "Accelerating graph analytics on cpu-fpga heterogeneous platform," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2017, pp. 137–144.

[33] R. Molina, F. Loor, V. Gil-Costa, F. M. Nardini, R. Perego, and S. Trani, "Efficient traversal of decision tree ensembles with fpgas," *Journal of Parallel and Distributed Computing*, vol. 155, pp. 38–49, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731521000915

[34] M. Maas, K. Asanović, and J. Kubiatowicz, "A hardware accelerator for tracing garbage collection," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 138–151.

[35] R. Kateja, N. Beckmann, and G. R. Ganger, "Tvarak: Software-managed hardware offload for redundancy in direct-access nvm storage," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 624–637.

[36] A. Pourhabibi, S. Gupta, H. Kassir, M. Sutherland, Z. Tian, M. P. Drumond, B. Falsafi, and C. Koch, *Optimus Prime: Accelerating Data Transformation in Servers*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1203–1216. [Online]. Available: https://doi.org/10.1145/3373376.3378501

[37] D. Skarlatos, N. S. Kim, and J. Torrellas, "Pageforge: A near-memory content-aware page-merging architecture," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 302–314. [Online]. Available: https://doi.org/10.1145/3123939.3124540

[38] Z. Wang, J. Weng, J. Lowe-Power, J. Gaur, and T. Nowatzki, "Stream floating: Enabling proactive and decentralized cache optimizations," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 640–653.

[39] G. Weisz, J. Melber, Y. Wang, K. Fleming, E. Nurvitadhi, and J. C. Hoe, "A study of pointer-chasing performance on shared-memory processor-fpga systems," ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 264–273. [Online]. Available: https://doi.org/10.1145/2847263.2847269

[40] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems," *ACM Comput. Surv.*, vol. 52, no. 3, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3323212

[41] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 55–64.

[42] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms," *Software: Practice and Experience*, vol. 52, no. 5, pp. 1095–1113, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3053

[43] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 269–279.

[44] C. Maia, L. M. Nogueira, L. M. Pinho, and D. G. Pérez, "A closer look into the AER model," in *2016 IEEE International Conference on Emerging Technology and Factory Automation, (ETFA 2016)*, September 2016.

[45] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric OS for multi-core embedded systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2016, pp. 1–11.

[46] Y. Zhou and D. Wentzlaff, "MITTS: Memory inter-arrival time traffic shaping," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 532–544, 2016.

[47] F. Farshchi, Q. Huang, and H. Yun, "BRU: Bandwidth regulation unit for real-time multicore processors," *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 364–375, 2020.

[48] M. Hassan, "Reduced latency DRAM for multi-core safety-critical real-time systems," *Real-Time Systems*, vol. 56, pp. 171–206, 2019.

[49] R. Mirosanlou, M. Hassan, and R. Pellizzoni, "DRAMbulism: balancing performance and predictability through dynamic pipelining," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 82–94.

[50] P. K. Valsan and H. Yun, "MEDUSA: A predictable and high-performance DRAM controller for multicore based embedded systems," in *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*. IEEE, 2015, pp. 86–93.

[51] J. Freitag and S. Uhrig, "Closed Loop Controller for Multicore Real-Time Systems," in *Architecture of Computing Systems – ARCS 2018*, M. Berekovic, R. Buchty, H. Hamann, D. Koch, and T. Pionteck, Eds. Cham: Springer International Publishing, 2018, p. 45–56.

[52] L. Feng, J. Huang, J. Hu, and A. Reddy, "Fastcfi: Real-time control-flow integrity using fpga without code instrumentation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 5, jun 2021. [Online]. Available: https://doi.org/10.1145/3458471

[53] I. Calciu, I. Puddu, A. Kolli, A. Nowatzyk, J. Gandhi, O. Mutlu, and P. Subrahmanyam, "Project PBerry: FPGA acceleration for remote memory," ser. HotOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 127–135. [Online]. Available: https://doi.org/10.1145/3317550.3321424

[54] H. Giefers, R. Polig, and C. Hagleitner, "Accelerating arithmetic kernels with coherent attached fpga coprocessors," in *2015 Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2015, pp. 1072–1077.

[55] Z. Wang, J. Sim, E. Lim, and J. Zhao, "Enabling efficient large-scale deep learning training with cache coherent disaggregated memory systems," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 126–140.