# Coherence-Aided Memory Bandwidth Regulation

Ivan Izhbirdeev*, Denis Hoornaert†, Weifan Chen*, Alexander Zuepke†, Youssef Hammad†,
Marco Caccamo† and Renato Mancuso*
*Boston University, USA
†Technical University of Munich, Germany
Email: {ivani, wfchen, rmancuso}@bu.edu, {denis.hoornaert, alex.zuepke, youssef.hammad, mcaccamo}@tum.de

***Abstract*—With the increasing adoption of PS-PL (Processor System-Programmable Logic) platforms, also known as CPU+FPGA systems, there arises a need for efficient resource management strategies. This work explores memory bandwidth regulation in such systems, leveraging the capabilities of tightly coupled FPGAs to offer elegant, low-overhead solutions with highly flexible regulation policies. We introduce *MemCoRe*, a novel approach that exploits the FPGA's interaction with cache coherence interfaces and cross-trigger signals to achieve fine-grained spatiotemporal awareness of processor activity and software-free control. By comparing *MemCoRe* with state-of-the-art software-based approaches, namely *MemGuard* and *MemPol*, we demonstrate significant improvements in regulation precision and overhead reduction. Key contributions include nanosecond-scale memory bandwidth regulation, off-core memory bandwidth accounting, address-aware regulation, low-overhead token-bucket regulation, and asymmetric on-off core throttling. Our evaluation on a Xilinx Zynq UltraScale+ ZCU102 CPU+FPGA platform showcases *MemCoRe*'s capability to regulate memory bandwidth with nanosecond-scale precision. Overall, *MemCoRe* presents a promising avenue for efficient memory bandwidth regulation in PS-PL platforms, with strong applicability to real-time systems.**

***Index Terms*—bandwidth regulation, coherence, cache**

## I. Introduction

The proliferation of PS-PL (Processor System-Programmable Logic) platforms, also known as CPU+FPGA systems, is rapidly expanding across both the embedded and general-purpose markets. Notably, the landscape has witnessed the emergence of new contenders, such as the AMD Embedded+ platforms and the recently announced AMD Versal 2 platforms. These advancements mark a significant stride, pushing beyond conventional boundaries into the realm of many-core systems coupled with FPGA integration.

The rise in popularity of PS-PL platforms offers a unique opportunity to rethink traditional approaches to system resource management. Following the state of the art, memory bandwidth regulation is a topic that has been extensively explored with the proposal of software-based techniques [1], [2] and dedicated hardware units [3], [4]. Acknowledging the importance of configurable bandwidth distribution in multicore heterogeneous system-on-a-chip (SoC), vendors have also proposed architectural solutions such as Intel RDT [5], [6], Arm QoS [7], [8], and Arm MPAM [9], which are still making their way into commercially available platforms. Nonetheless, these approaches come with various shortcomings, from the need to modify key layers in the system software to the need for custom hardware redesign/integration. Even solutions like RDT, QoS, and MPAM have limited programmability because they cannot enact different regulation policies depending on the exact downstream resource from which bandwidth is being consumed.

In this paper, we demonstrate that if a tightly coupled FPGA is available in an SoC, memory bandwidth regulation can be done elegantly, with minimal overheads, while offering the ability to produce highly flexible regulation policies. In particular, this paper showcases the use of two key enabling features of tightly coupled FPGAs, namely the ability of the FPGA to interact with (1) cache coherence interfaces and (2) cross-trigger signals. Importantly, the combination of these mechanisms allows fine-grained spatiotemporal awareness of the activity of the processors under regulation and software-free control of said processors.

We call the presented approach *MemCoRe* to stress its ability to perform *Memory management via Coherence-aided Regulation*. *MemCoRe* improves on the two state-of-the-art software-based memory bandwidth regulation approaches, *MemGuard* [1] and *MemPol* [10] as follows. First, by enacting regulation from outside the cores, *MemCoRe* overcomes the intrinsic implementation overheads of *MemGuard* for fine-grained regulation. Second, by optimizing the critical path in bandwidth regulation, *MemCoRe* improves performance by an order of magnitude and overcomes the problems of setpoint overshooting observed in *MemPol*. Compared to solutions like Intel RDT, Arm QoS, and Arm MPAM, *MemCoRe* sets itself apart for its ability to enact regulation policies on a per-memory-region basis while being immediately applicable in SoCs with cache-coherent programmable-logic.

We compare *MemCoRe* with *MemPol* using its original software implementation (*MemPol-SW*). For fairer comparison, we also implemented a custom *MemPol* implementation in FPGA (*MemPol-HW*). This allows us to discuss and evaluate design trade-offs and optimizations to achieve the presented nanosecond-scale bandwidth regulation. All the systems we contrast, *i.e.*, *MemCoRe*, *MemPol-SW*, and *MemPol-HW*, were fully implemented and evaluated on a Xilinx Zynq UltraScale+ ZCU102 [11] CPU+FPGA platform where they are used to regulate the memory bandwidth of the platform's four Arm Cortex-A53 cores.

We make the following key contributions in this work:

- *Nanosecond-scale memory bandwidth regulation* of applications cores from the FPGA component of a commercially available PS-PL platform.

- Off-core *memory bandwidth accounting* based on monitoring the core's coherency traffic instead of using performance counters (PMCs).
- *Address aware* accounting of memory traffic that enables fine-grained tuning of the regulation for different memory regions.
- Memory bandwidth *regulation* based on token-bucket regulation with low implementation overhead in hardware.
- An asymmetric *on-off core throttling* approach that optimizes for fast halting of cores to overcome *setpoint overshooting* of the regulation setpoint.
- A substantial *design space exploration* of alternative approaches for implementing key modules in a PL-side memory bandwidth regulator.

The rest of this paper is structured as follows. Sec. II presents the background on memory bandwidth regulation. Sec. III reviews challenges and discusses opportunities for improvements in PS-PL platforms. Sec. IV details *MemCoRe*'s design and Sec. V its implementation. We evaluate the proposed approach in Sec. VI, with a discussion of key limitations and possible avenues for future work in Sec. VII. Sec. VIII discusses closely related work, and Sec. IX concludes the paper.

## II. Background

### A. Memory & Cache Model

Modern computer systems employ a hierarchical memory architecture to bridge the performance gap between fast on-chip processing elements (PEs) and the slower off-chip main memory. The hierarchy consists of multiple levels of caches and the main memory. The cache levels closer to the PE, such as the L1 cache, are typically private to each PE. The last-level cache (LLC) is generally shared among multiple PEs and directly interfaces with the memory controller. The memory controller is responsible for the data movement between the on-chip logic and the off-chip main memory (*e.g.*, DRAM).

*Memory bandwidth* is the rate at which cache lines are transferred between the LLC and the main memory. Two architectural events contribute to the memory activity: cache *refills* and *write-backs*. Upon a refill, a cache line is fetched from lower to higher levels of the hierarchy. During a write-back, a *dirty* line being evicted from higher levels is written back to lower levels, eventually reaching the main memory. The combined rate of refills and write-backs between the LLC and the memory controller constitutes the overall memory bandwidth.

In hard real-time systems, memory budgets are dimensioned using the maximum *sustainable bandwidth*—the highest bandwidth a memory controller can sustain under worst-case workloads (*e.g.*, row misses in the same bank). As documented in [8], this can be much lower than the peak achievable bandwidth. This conservative metric serves as the baseline for memory regulation. Determining the sustainable bandwidth requires platform-specific knowledge and experimentation [12], [13].

### B. PMC-based Memory Bandwidth Regulation

Modern platforms typically contain performance monitoring counters (PMCs) to count the occurrence of various architectural events. Techniques such as *MemGuard* and *MemPol* use PMCs to monitor the number of LLC refills and write-backs in a given time window to estimate the main memory bandwidth usage at runtime. If the estimated bandwidth exceeds the user-set threshold, a regulation action is taken to throttle bandwidth usage. Regulation actions could be scheduling a CPU-intensive high-priority task or stalling the core exceeding the threshold.

*MemGuard* relies on the Performance Monitoring Unit's (PMU) ability to interrupt a PE when its PMC exceeds a configured threshold. Thus, *MemGuard* periodically resets the PMC to replenish a PE's budget. Due to the interrupt overhead for periodic replenishment, *MemGuard* regulates at millisecond granularity [1], [14]. Conversely, *MemPol* periodically polls the PMCs of all the monitored PEs from an auxiliary PE and regulates the bandwidth by halting/resuming a specific PE via on-chip debug signals. This allows *MemPol* to regulate at microsecond granularity [10].

### C. PS-PL Platforms

PS-PL platforms represent highly heterogeneous System-on-Chip architectures characterized by the combination of *"traditional"* PEs (the *Processing System* or PS) and a re-programmable fabric (the *Programmable Logic* or PL). In high-performance embedded PS-PL platforms, the PS typically features several CPU cores clustered together with an LLC and connected to a DRAM controller through a Cache Coherent Interconnect (CCI). Such platforms may also include multiple real-time PEs to handle low-latency tasks. The PL-side comprises a *Field Programmable Gate Arrays* (FPGA) and is tightly connected to the PS-side via many memory and I/O interfaces to enable a high degree of PS-PL cooperation.

**Unidirectional high-speed bus interfaces.** Several unidirectional ports allow communication (1) from PS to PL and (2) from PL to PS. Each PS-to-PL port is associated with a unique SoC-wide physical address range so that any PE-originated transaction can be non-ambiguously routed to the PL. Likewise, PL-to-PS ports allow the PL to access any memory target using their SoC-wide addresses. This includes access to on-chip memories and the main memory. In most embedded platforms, these ports utilize the AXI-Full protocol [15].

**Two-way coherent memory accesses.** The PL is not confined to host peripheral modules and non-coherent accelerators and can be elevated to become a member of the SoC cache coherence domain. This enables the PL to receive *snoops* from the CCI, effectively exporting information about the memory-related activity occurring in the PS to the PL. Previous research [16] demonstrates the level of detail that can be extracted and exploited this way.

**Interrupts and cross-triggers.** In addition, the PL can both send/receive interrupts to/from the PS. In Arm-based platforms, this can be done either via a FIQ (Fast Interrupt Request) or IRQ (Interrupt Request). Moreover, PS and PL are inter-connected via direct cross-trigger interface (CTI) lines

to/from the CoreSight infrastructure, which, when adequately controlled, can command the PEs to halt/resume their execution in an invisible way to the software layers.

### D. Cache Coherence

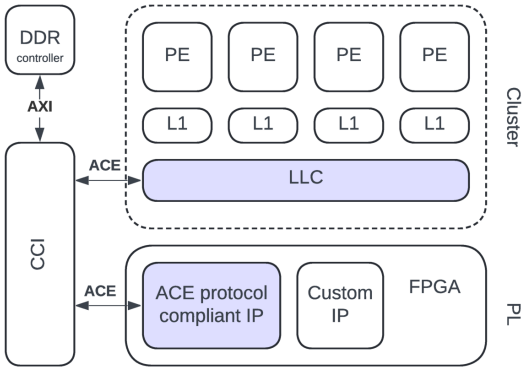


Fig. 1: SoC cache coherency architecture. The colored components are in the same coherence domain. When an LLC miss occurs, CCI will broadcast a snoop to the protocol compliant IP.

A cache coherence protocol facilitates the distribution of cached data across the SoC while providing a consistent view of the data items to all PEs adhering to the protocol. If the caches of several PEs are kept coherent, they are said to be in the same *coherence domain*. For example, on PS-PL platforms, the PL can be configured to be in the same domain as the PS using the ACE [15] protocol as depicted in Figure 1.

As is the case for ACE, cache coherence is often implemented using a *snoop-based cache coherence protocol* [17]. In this case, whenever a coherent PE requests access to a cache line (*e.g.*, upon a refill), the coherency fabric broadcasts a *snoop request* to all PEs of the coherency domain. These PEs must reply to indicate whether they have a local copy of the requested cache line. If so, they must perform adequate actions to maintain cache coherence, *e.g.*, updating the coherence state of the cache line or providing the most up-to-date content of the line. Likewise, PEs can use snoop requests to announce state changes of cache lines, *e.g.* when a cache line needs to be modified. Importantly, snoop requests carry meta-data identifying the requested line, such as its physical address.

## III. Exploring Regulation in PS-PL Platforms

The feature-rich nature of PS-PL platforms opens a range of opportunities to rethink memory bandwidth regulation to overcome traditional challenges of state-of-the-art PMC-based techniques such as *MemGuard* and *MemPol*. Without loss of generality, all memory bandwidth regulators can be decomposed into three individual *modules*:

1) An *accounting module* whose objective is to obtain and/or estimate the PEs' memory activity metrics (*e.g.*, refills, write-backs).
2) A *decision module* whose objective is to decide on what actions to undertake (*i.e.*, halt, resume, no-action), based on the current and past memory activity.
3) An *enacting module* whose objective is to conduct the action dictated by decision module.

Realizing each of these modules on an off-the-shelf platform at the software level comes with technical challenges that hinder the regulation's quality. Let us closely examine the models and limitations of both regulators.

### A. MemGuard Model

In a *MemGuard*-like regulator, as mentioned in Sec. II-B, the PMC is used as a countdown counter. When its value reaches zero, an interrupt is issued[1] immediately to execute the decision module. Hence, the delay between the moment throttling is required and its enacting is short, meaning that *MemGuard* does not suffer from budget overshooting. A second periodic interrupt is required to replenish the budget and enforce a given memory bandwidth value over time. Unfortunately, when higher enforcement granularity is requested, the reliance on two interrupt handlers becomes problematic. Achieving finer granularity with the same target bandwidth implies that both the per-period budget and the replenishment period must be smaller. Consequently, a ten-fold granularity increase can lead to a twenty-fold increase in the delivered interrupt rate. Due to this limitation, *MemGuard* typically operates at millisecond-granularity. At microsecond-granularity, it incurs prohibitive overhead [10]. In short, *MemGuard* is good at preventing overshooting, but realistically, it can only operate at millisecond-scale granularity.

### B. MemPol Model

Recognizing the overhead problem, *MemPol* proposes to use other auxiliary on-chip PE(s) to execute the regulation logic. Now that the PMC(s) can be sampled from an external PE and since the regulator also resides *outside* the core(s) under regulation, the bandwidth estimation must be done following a new approach. *MemPol* periodically polls the value of the relevant PMCs associated with each regulated core. It does so by accessing the PMCs via memory-mapped (AXI) transactions to the PE's CoreSight registers and using the difference between two consecutively sampled values to estimate the current bandwidth. To halt/resume the activity of the cores, *MemPol* also accesses debug control registers via memory-mapped transactions. The immediate benefit of using said approaches for the accounting and enacting modules is that *MemPol* no longer injects interrupts in the control flow of the regulated PEs. With less overhead, *MemPol* can operate in microsecond-scale granularity. However, the periodic sampling nature makes it prone to overshooting problems. Indeed, regulation can only react as fast as the maximum achievable polling frequency. If a burst of memory transactions occurs, *MemPol* cannot respond until the newest PMC reading is polled. In short, *MemPol* can operate on microsecond-granularity due to low overhead but is prone to overshooting problems.

[1]More precisely, the interrupt is issued when the counter overflows, so the budget is encoded as the value representing the maximum precision of the PMC minus the budget value to be tracked.

### C. New Opportunities on PS-PL Platforms

Despite the improvement brought by *MemPol* at the cost of dedicating a (low-power) PE, overcoming the challenges described above requires shortening the accounting and enacting stages, which is rendered difficult by the unsuitability of a typical system's architecture. For instance, in the accounting stage, the impossibility for *MemGuard* and *MemPol* to swiftly access or act on the PMCs' content constitutes a major hurdle against regulation granularity. Both architectures also suffer from the inability to distinguish accesses to different memory regions. They must, for instance, assume that all the accesses target the same DRAM bank. Software-driven PMC-based regulation on off-the-shelf platforms has hit a limit.

To overcome these architectural limitations, we propose *Memory management via Coherence-aided Regulation* (*MemCoRe*). *MemCoRe* is a hardware module designed for off-the-shelf PS-PL platforms. It addresses the aforementioned challenges by elevating the role of the PL to achieve fine-grained bandwidth regulation of high-performance PEs. *MemCoRe* is designed from its inception with three core concepts:

**Accounting-regulation locality.** We identify that one of the primary sources of regulation latency—*i.e.*, the time between budget exhaustion and halting of the target PE—originates from inadequate interfacing between the PMCs and the PE driving the regulation. We argue that co-locating the accounting and the decision units is fundamental to reducing the regulation latency and taming the overshooting.

**On-time throttling.** Another important intuition to tame budget overshooting is allowing the decision unit to be informed of budget overruns as soon as they occur. This way, it is possible to immediately activate the enacting unit.

**Discerned regulation.** One common drawback of existing PMC-based regulation mechanisms is their *one-size-fits-all* approach to throttling. Traditionally, any memory access generated by the PE under analysis counts against the assigned quota of sustainable bandwidth. Modern systems, however, often feature heterogeneous memory sub-systems—*e.g.*, comprised of multiple DRAM controllers, scratchpad memories, and non-volatile memory. Thus, the budget consumed from different memory targets should be appropriately differentiated.

Instead of fixing a single bandwidth target per PE, with all tasks running on that PE sharing the same quota, we postulate that bandwidth targets should be associated with memory regions to allow for greater flexibility and reduced pessimism when assigning bandwidth targets.

To do so, we propose elevating the PL's role to that of a (passive) coherent actor in the coherence domain. From this position, the PL-located *MemCoRe* can collect *virtually all* coherent bus activity information. This capability to observe the SoC memory activity is leveraged to implement PMC-like counters within *MemCoRe*, effectively enabling the accounting-regulation co-location we seek. It also enables on-time throttling as the PMC-like registers can inform the decision logic of bandwidth budget overruns with a delay of a single clock cycle. Finally, thanks to the dedicated PL-to-PS cross-trigger lines, halt and resume commands can be swiftly sent to the PS-side PEs.

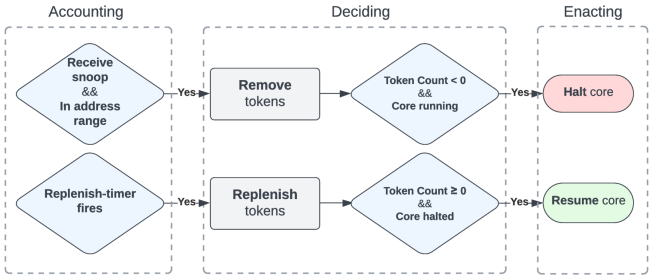


Fig. 2: *MemCoRe* block diagram. At each clock cycle, both threads of hardware logic will be performed.

## IV. Design

This section describes the key design choices of *MemCoRe*. To better leverage the flexibility offered by PS-PL platforms, *MemCoRe* is designed with modularity in mind. Thus, *MemCoRe* offers multiple alternative approaches to implement the accounting, decision, and enacting modules. Figure 2 shows the logic organization of the most optimized version of *MemCoRe*. The remainder of this section will explain each component in detail.

### A. System Requirements

In this work, we assume a PS-PL platform complying with the description in Sec. II-C. In particular, we assume a *cache coherent interconnect* linking one or more *computing clusters* comprising one or more PEs (PS) and the programmable logic (PL), with the ability to logically include the PL in the same *coherency domain* as the cluster(s) to monitor cache coherence messages. We further assume that a snoop-based cache coherence protocol is used. For regulation, the PL must efficiently halt and resume specific PEs. We assume that this can be done in two ways: (1) by sending per-core interrupts or (2) by using the debug infrastructure of the platform.

### B. Accounting of Cache-Coherent Memory Transactions

In Sec. III, we identified that reading PMCs via their memory-mapped CoreSight interface is a limiting factor as the access delay scales linearly with the number of cores. Moreover, read requests in the form of AXI transactions may traverse bus segments shared with the route taken by the PE-originated memory requests.

While *MemCoRe* can be configured to sample the PMCs in this way, the full potential of *MemCoRe* is unlocked by allowing the PL to observe the *coherency traffic* (cache snooping) generated by the PEs, similar to the approach in [16]. For each cache line refill operation that cannot be satisfied by the LLC, the coherency fabric queries the other cache-coherent agents in the system. This is done to access the latest version of the cache line from other coherent caches in the system. Said *snoop requests* identify the cache lines by physical address. This allows the PL (1) to identify the source PE based on the

cluster ID and the cache line address, (2) to count all cache coherent refill transactions for bandwidth regulation, and (3) to apply different *cost factors* for accesses to different memory regions associated to corresponding physical address ranges.

Unfortunately, the PL cannot immediately distinguish the source PE in these transactions. Therefore, the PL must also use the line address to differentiate between PEs. This is usually not a problem when using partitioning hypervisors, which typically allocate and assign memory to VMs statically (memory range based) or when cache coloring is used (color based), *e.g.*, as described in [18], [19]. Also, the PL cannot observe the PEs' write-back transactions, as voluntary write-back transactions—write-backs of dirty cache lines evicted from the LLC—do not need to be broadcasted on the coherence interconnect.

Relying on coherence traffic scales better than using PMCs, and it frees PMCs for software use. It also removes the dependency on the CoreSight infrastructure that is otherwise needed to access the per-core PMCs from outside the PEs. Furthermore, the accounting stage no longer requires strict serialization between the cores for reading the PMCs as in *MemPol*. This allows a decision stage to run *independently* for each core and in lock-step with the accounting module. The new limiting factor becomes the sustainable throughput of the cache coherent interconnect between the agents [16].

### C. Per-Core Token Bucket Regulation

MemPol uses a *sliding window* method for regulation [10]. The sliding window effectively guarantees a minimum bandwidth over time and allows PEs to spare bandwidth during the window for short memory burst phases. We observe a similarity between the sliding window method and the *token bucket* regulation often used in networking. In a token bucket regulation, a token dispenser constantly refills a proverbial bucket with fresh tokens (replenishment) until the bucket is full (available budget). A consumer can take tokens as long as the bucket has tokens (burst). In our case, the tokens are the cache lines contributing to memory bandwidth, and we use the condition that the bucket is empty to halt a core. Unlike in networking, where token bucket regulations are often used as *admission protocols*, *e.g.*, before sending data to a network, *MemCoRe* cannot delay an ongoing memory transaction at the source, and therefore must cope with overshooting. Hence, our bucket level can become negative. In this case, we keep a core halted until its available budget reaches the zero level again.

With this, a regulation of PE $i$ at each replenishment period of length $\Delta_t$ comprises: (1) a configurable *replenishment rate* $R_i$—number of cache lines per replenishment period $\Delta_t$, (2) a configurable *budget limit* $B_i$—*bucket size* expressed in number of cache lines, and (3) the currently *available budget* $A_i(t)$—*bucket level* expressed in number of cache lines, with $A_i(t) \leq B_i$. We denote $C_i(t)$ the number of cache lines *consumed* by PE $i$, defined as the number of observed cache refills during the time interval $[(t - \Delta_t), t]$ when using a regulation period of length $\Delta_t$.

Under *synchronous* available budget tracking, we adjust the available budget every $\Delta_t$ (period) time units so that

$$A_i(t) = min(A_i(t - \Delta_t) + R_i - C_i(t), B_i), \tag{1}$$

*i.e.*, we both replenish and consume in one step and do not let the bucket overflow. *MemCoRe* throttles a core if $A_i(t) < 0$, effectively halting it on the first instant at which the condition

$$A_i(t - \Delta_t) \geq 0 \wedge A_i(t) < 0 \tag{2}$$

holds; *MemCoRe* releases the core again as soon as

$$A_i(t - \Delta_t) < 0 \wedge A_i(t) \geq 0 \tag{3}$$

holds. In the model described above, the halt decision is synchronous or "periodic" in the sense that the halt decisions can only be made at the end of each replenishment period. We will refer to this variant as *MemCoRe-periodic*.

Alternatively, the halt decision can be decoupled from the replenishment strategy and be made *asynchronous*. As the accounting module on the PL-side operates at each clock cycle, the token accounting is continuously updated, and a halt decision can be fired whenever the tokens are depleted. The best-performing version of *MemCoRe* employs said "asynchronous-halt" approach. This model is described in Figure 2, and we quantify the performance gain achieved through asynchronous-halt in our evaluation in Sec. VI.

### D. Core Throttling using CTI Triggers

*MemPol* uses the PEs' debug interfaces to halt and resume cores. In particular, *MemPol* triggers specific halt and resume signals of the PEs exposed via memory-mapped *CTI trigger registers* on the CoreSight interface that let PEs enter or leave the *debug halt* state. As noted in Sec. III, and similar to the accounting stage, accesses to the memory-mapped CoreSight registers require serialization to guarantee bounded access times and bounded reaction times. However, promptly halting the PEs is more critical to reduce regulation overshooting than resuming them.

With this insight, a deeper platform analysis has shown that an alternative mechanism exists to trigger CTI signals from the PL. Indeed, the PL can directly trigger up to four debug signals that can either halt or resume PEs, or even listen to CTI signals from the cores. Therefore, we wire the PEs' halt signals to the PL and allow it to halt the PEs directly. This eliminates any dependency on CoreSight transactions to halt a PE and further allows the enacting module to be activated in lock-step with both accounting and decision modules, independently for each PE.

To resume any PE, the enacting module uses serialized AXI transactions as in *MemPol*. This effectively introduces a *release delay* $D$ for a previously halted PE that depends on the number of pending AXI transactions on the CoreSight interface. On the other hand, the delay allows the PEs to accumulate some budget, preventing them from being immediately throttled after release. However, the bucket should not fully fill up during the delay, so $D \leq \frac{B_i}{R_i}\Delta_t$. This effectively puts a lower limit on bandwidth settings.

### E. Core Throttling using Interrupts

As an alternative to CTI-based core throttling, *e.g.*, on platforms where the PL has no access to the PE debug infrastructure, we can also consider an interrupt-based regulation. Here, the PL raises an interrupt to halt a PE, and the PE's interrupt handler keeps the core in a busy waiting state as long as the PE is halted. However, the interrupt handler's code and data footprint contribute to unavoidable overshooting. Therefore, the code path to throttle the PE in the interrupt handler must be as short as possible. But we also have to consider that the regulation interrupts compete with *other* interrupt activity in the system. The Arm Generic Interrupt Controller (GIC) architecture supports interrupt prioritization. This could reduce the competition to, at most, one currently ongoing interrupt. To further reduce interference, the Arm architecture defines two interrupt groups (*FIQ* and *IRQ*) with independent handlers, where FIQs take precedence over IRQs [20]. However, FIQs are often used for firmware purposes and require a handler at the firmware level. Besides testing the general feasibility of interrupt-based regulation, we refrained from extensively modifying the Arm Trusted Firmware (ATF). We, therefore, focus on CTI-based throttling in the rest of this paper. Note that all interrupt-based regulation mechanisms share these problems, including *MemGuard*.

### F. Address Awareness

As mentioned in Sec. IV-B, the address awareness of the proposed regulation approach allows using different *cost factors* for accesses to different memory regions. This becomes handy for several reasons. For instance, (1) when accessing read-only memory, *e.g.*, code segments or video data from an incoming camera stream, the cache lines can never become dirty and thus need never to be written back. This enables reducing the pessimism for access to these memory regions, as mentioned in Sec. IV-C. (2) DRAM-like memories of different types, such as external byte-addressable non-volatile memory, chip-internal scratchpad memory, or GPU-local memory, can be characterized by different speed grades and bandwidth limitations. Even within a single DRAM controller, one could separately track the bandwidth extracted from each bank, lowering the pessimism in the considered saturation thresholds. Using per-region cost factors allows combining different memory accesses into a single regulation scheme. Moreover, address awareness allows one to use *independent* regulation schemes and bandwidth budget settings for each memory type, similar to the read and write bandwidth regulation for *MemGuard* in [14].

### G. Global Regulation to Distribute Unused Bandwidth

*MemPol* provides a *global regulation* mechanism that distributes the PEs' unused bandwidth among PEs that are currently short of bandwidth. The same technique can be used for *MemCoRe* with adaptions.

The global regulation accumulates the sum of the consumed bandwidth of all PEs $A_g(t) = \sum_{\forall i} A_i(t)$. The budget is set to the sum of the all per-core budgets $B_g = \sum_{\forall i} B_i$ if the global regulation is enabled, or to $B_g = 0$ otherwise. The global regulation overrides the decision of the per-core regulation to halt a core if the global budget is underutilized. A core $i$ is then throttled if

$$A_i(t) < 0 \wedge A_g(t) < 0. \tag{4}$$

In the case of helping a halted core out, *MemPol*'s global regulation additionally needs to adjust the point of reference of the sliding window regulation of the core to prevent the core's over-utilization of bandwidth from becoming a penalty. We overwrite and reset the core's available budget to zero in an analogous step in *MemCoRe*, *i.e.* $A_i(t) = 0$.

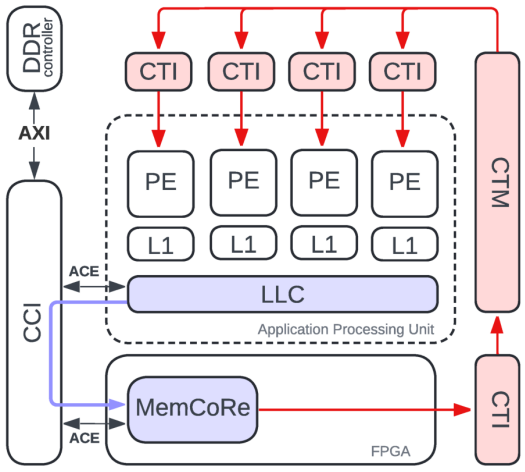


Fig. 3: Integration of *MemCoRe* into the SoC's cache coherency architecture. The blue arrow indicates the snoop direction. The red arrows indicate the directions of the enacting signals. The red components are from CoreSight.

## V. Implementation

To discuss our *MemCoRe* implementation, we consider the features available in the most widely used family of commercially available PS-PL platforms, namely the Xilinx Zynq UltraScale+ SoCs [11]. These platforms include multiple application processor units (APU) and a sizeable onboard programmable fabric (PL). The APU cores feature private L1 caches and a shared L2 cache (LLC). Importantly, the PL-side is also attached to the system's CCI via an ACE port. Thus, the PL-side can be placed in the same coherence domain as the LLC. The PL-side also features two high-performance (HPM) PS-to-PL ports for memory-mapped access. An instance of a standard ARM CCI-400 component is used as the system CCI [21].

CoreSight is Arm's solution for hardware debugging [22], which defines registers of memory-mapped debug devices accessible through a debug access port (DAP). The target family of PS-PL platforms includes a typical CoreSight infrastructure. A CoreSight component, the Cross-Trigger Interface (CTI), is generally used by external hardware debuggers to halt/resume cores. The PS-PL SoCs of reference include CTI lines on the PL-side, allowing the latter to act as an external

debugger. A Cross-Trigger Matrix (CTM) [11] is responsible for propagating signals among CTIs. Fig. 3 illustrates the hardware components employed and signal paths. Further details are provided in the subsequent sections.

### A. Accounting Module Implementation

To accurately monitor memory bandwidth in different regions of physical memory, *MemCoRe* is interfaced with the coherence interconnect through the ACE port. This configuration enables passive monitoring of coherence traffic.

To systematically measure memory bandwidth utilizing coherence in the PL, the following steps are needed:

- Integration of the PL into the coherence domain is done by modifying the Arm Trusted Firmware (ATF) to enable the PL-facing ACE port, which allows the PL to participate in the coherence protocol.
- Once integrated, the PL receives traffic via the CCI. This traffic is similar to that on the AXI interface and consists of requests that the PL can either respond to or simply observe. Some requests, such as Distributed Virtual Memory (DVM) operations and synchronization packets, require proper handling in terms of timely responses. Neglecting these requests will lead to system-wide freezes, as the coherence protocol mandates responses to such requests from all participants. For our purpose of passive observation, the PL issues simple acknowledgments to remain compliant.
- Only snoop requests are relevant for accounting purposes. When an LLC miss occurs, a snoop is broadcast by the CCI to query all coherence domain participants. The snoop contains the corresponding physical memory address of the cache line. The PL does not need to send a reply unless it intends to actively participate in the coherence protocol [16]. In our implementation, *MemCoRe* records the physical address associated with the snoop request. This address is checked against the configured address ranges under monitoring, and a counter for the matching address range is incremented accordingly.

### B. Decision Logic with Asynchronous-halt

The token bucket model described in Sec. IV-C is implemented on the PL. When a snoop request is received, the decision module decrements the associated bucket value and checks whether the value is negative. If so, a halt decision is made immediately, and the decision module signals the enacting module to execute it. The replenishment logic runs independently. Every $\Delta_t$ cycles, the replenishment value is added to the bucket. If this results in a negative bucket becoming positive, then a resume decision is made, with the decision module signaling the enacting module to execute it.

### C. Enacting via PL-side CTI Signaling

Zuepke *et al.* [10] describe in detail how to utilize the CTI via memory-mapped accesses to instruct a core to enter/leave debug state in software. In summary, to enter/leave, a write transaction to the corresponding CTI is necessary. Additionally, another write transaction to the CTI to acknowledge a previous debug request before leaving the debug state is also required. The regulation action can be done faster due to the existence of PL-side CTI. Indeed, the PL-side CTI contains lines that can propagate direct signals to the CTIs of other clusters on the platform (see Figure 3).

To fully appreciate the advantage of utilizing the PL-side CTI, it is beneficial to explain the semantics of the aforementioned "write to CTI" step. A CTI consists of eight 1-bit input triggers and eight 1-bit output triggers. The connectivity between the triggers and the outside components is typically hardwired. To propagate signals among CTIs, a total of four shared channels are present and exposed to each CTI on the platform. Inside a CTI, the connectivity between the channels and triggers is programmable. If an input trigger connects to a channel, an outside event driving the trigger high will drive the channel high. If a channel connects to an output trigger, when the channel is driven high, the output trigger will also be driven high. By programming the connectivity in each CTI, a signal can be propagated to other CTIs via the shared channel[2]. Additionally, a channel can also be driven high for the duration of one cycle with a write to the `APPPULSE` register. Thus, the aforementioned "write" is a software write to `APPPULSE` setting the appropriate bit corresponding to a specific channel. The channel is, in turn, connected to output triggers that command a CPU to enter/leave the debug halt state. The software write is an AXI transaction.

By directly leveraging the PL-side CTI, *MemCoRe* can be designed to have pins connected to the input triggers. Thus, instead of an AXI transaction, the regulator can simply drive the corresponding trigger pin high to deliver a signal for entering/leaving the debug state. Note that the acknowledgment still needs to be done through an AXI transaction. Directly driving a pin high is significantly faster than issuing an AXI transaction. Our measurements show that the time it takes for an AXI transaction originating from the PL to complete entering/leaving/acknowledging actions is $420\ ns$. Directly driving a pin high for entering/leaving the debug state takes around $100\ ns$. Our observed worst-case was $350\ ns$.

### D. Implementation Variations

The modular implementation allows different choices for each of the three modules. The accounting module can rely on either polling the PMCs or listening to the snoops. For the decision module, the halt decision can be made either periodic or asynchronous. The decision to resume a halted core is made periodically at the end of each replenishment period. The enacting module involves signaling the halt/resume (either through AXI or CTI trigger) and acknowledgment (through AXI only). This again offers four options. In total, this provides 16 combinations. From our experiments, we find that the best performance is provided by the

[2]The gate register `GATE` of a CTI can be programmed so that the CTI will keep the channel state local, thus not propagating signals to other CTIs/CTMs.

configuration where (1) snoops are used in the accounting module, (2) asynchronous-halt for the decision module, and (3) halt/resume via PL-side CTI trigger was employed in the enacting module. From now on, *MemCoRe* refers to this specific configuration. We will also evaluate other representative implementations in Sec. VI.

## VI. Evaluation

This evaluation aims to study and ensure the capabilities of *MemCoRe* and its key variants against *MemPol*. To this end, we deploy *MemPol* and each variant of *MemCoRe* on the Xilinx Zynq UltraScale+ ZCU102 SoC [11]. The ZCU102 is a high-performance embedded PS-PL platform with one Application Processing Unit (APU), one Real-time Processing Unit, and one tightly integrated FPGA (PL). The application processing unit is composed of four ARM Cortex-A53 cores operating at 1.2 GHz and clustered together with a 1 MB LLC. In all our experiments, we chose the APU's cores as the PEs to be regulated. The Real-time Processing Unit is a smaller cluster of two ARM Cortex-R5F cores operating at 500 MHz. For the purpose of our evaluation, these PEs are only used when regulating with *MemPol*. The PL hosting *MemCoRe* and its variants is clocked at 100 MHz and is included in the same coherence domain as the APU as described in Sec. V.

For benchmarking purposes, the APU runs a full-fledged Linux kernel v6.1. We employ a set of memory-intensive benchmarks issued from the San-Diego Vision Benchmark Suite [23] (SD-VBS)[3] and the Isolbench's Bandwidth benchmark [24]. In particular, we use the RT-Bench [25] compatible version of these benchmarks.

To satisfy *MemCoRe*'s requirement for physically contiguous address chunks to distinguish between different cores and the way Linux dynamically allocates memory, we employ a tailor-made kernel module. This module holds several pointers to distinct and physically contiguous memory regions. When used in conjunction with RT-Bench's malloc wrapping feature, it allows the processes' heap to be seamlessly located in one of the regions. This means that all snoops can be immediately associated with the corresponding Linux process under regulation at runtime and during the offline analysis.

Finally, to analyze and conclude on the regulator's behavior, we deploy an ACE bus *tracer* on the PL-side to obtain clock-cycle-accurate insight into the bus activity. This module is inspired by the *silent-spy* presented in [16] and is attached to the ACE bus, linking the CCI to *MemCoRe* in a non-invasive way. It can be configured to monitor specific address ranges, such that when a snoop carrying this address is observed, a tracing packet containing a timestamp and the address is created. These packets are then sent and stored in the PL-side DRAM, where they can be recovered for later offline analysis.

TABLE I: Total memory usage (unit: MB).

| | *Actual-usage* | *PMU-global* | *PMU-targeted* | *Co-global* | *Co-targeted* |
|---|---|---|---|---|---|
| Bandwidth | 1024.0 | 1051.34 | 1026.91 | 1051.34 | 1022.08 |
| Disparity | N/A | 257.81 | 233.49 | 257.81 | 225.83 |
| MSER | N/A | 24.01 | 6.83 | 24.01 | 2.56 |
| Sift | N/A | 67.09 | 46.57 | 67.09 | 41.20 |
| Stitch | N/A | 29.39 | 10.39 | 29.39 | 6.79 |
| Tracking | N/A | 30.91 | 12.71 | 30.91 | 8.39 |
| Local. | N/A | 22.19 | 4.91 | 22.19 | 0.14 |

### A. Accounting Equivalence

This experiment aims to evaluate the precision with which *MemCoRe* can monitor the system memory traffic. We compare the number of transactions recorded by *MemCoRe* and the PMCs when benchmarks from RT-Bench are executed.

We consider four different ways, as shown in Table I, to count the number of LLC cache refills occurring in the system: (1) *PMU-global* uses the PMU for all cores; (2) *PMU-targeted* uses the PMU for the core under analysis; (3) *Co-global* uses *MemCoRe* for all cores; and (4) *Co-targeted* uses *MemCoRe* for the core under analysis. When derivable, *Actual-usage* reports on the ground-truth value.

As reported in Table I, the experiment's results show that the activity recorded by *MemCoRe* is in line with what the PMU reports. When system-wide activity is monitored, *Co-global* reports exactly the same amount of transactions (here expressed in MB) as *PMU-global*. However, discrepancies can be observed between the recording of *PMU-targeted* and *Co-targeted*, with the latter always reporting higher activity than the former. These discrepancies stem from a difference in the monitoring capabilities of *MemCoRe* and the PMUs. In fact, since the PMUs monitor all refill events occurring on the CPU core, it also accounts for other processes' activity. On the other hand, *MemCoRe* account refill events for and only for the address range(s) of interest. When looking at the results for Bandwidth, *Co-targeted* is just 1.92 MB away from the *Actual-usage* whereas *PMU-targeted* is 27.34 MB away. This indicates that the accounting via coherence is accurate.

### B. Phase-aware Throttling

The following experiment demonstrates that *MemCoRe* can effectively apply differential throttling even within a task during different execution phases. For evaluation purposes, we designed a template application in which two phases ($P_{\text{critical}}$ and $P_{\text{non-critical}}$) execute alternatively in a loop. $P_{\text{critical}}$ is assumed to be a mission-critical section whose QoS has to be guaranteed, while $P_{\text{non-critical}}$ can be executed following a best-effort approach. By instructing *MemCoRe* to only throttle the memory bandwidth occurring within the address range of $P_{\text{critical}}$, the differential regulation can be achieved. The trace of this experiment is shown in Figure 4. In general, *MemCoRe* supports the definition of arbitrarily many regions, each regulated at a configurable bandwidth setpoint.

[3]For all SD-VBS benchmarks, we use the *qcif* input size due to the *tracer* buffer size, except for Figure 7 in which *vga* is used to be consistent with MemPol's evaluation [10].

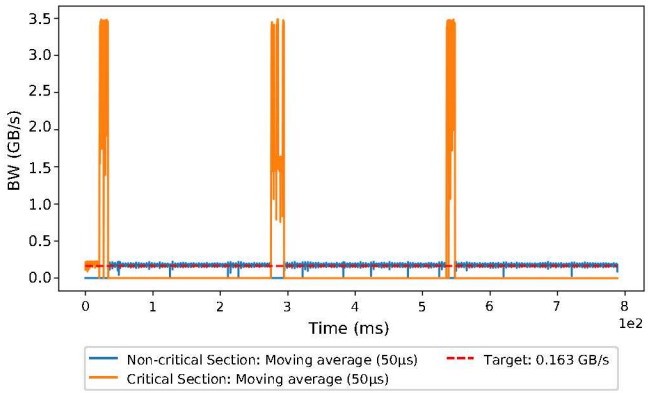


Fig. 4: Bandwidth trace of an application under regulation. There are two sections: one is critical, which is not regulated, and a regulated one with a setpoint BW at 165 MB/s.

### C. Taming Overshooting: Synthetic Benchmark

One key objective of *MemCoRe* is to tame overshooting. As discussed in Sec. IV, this is achieved by shortening the regulation period and allowing for asynchronous PE halting.

To compare our regulators, we devised a simple experiment where a core runs IsolBench's bandwidth as a read-only memory bomb while being regulated. The set of regulators includes *MemPol*, *MemCoRe*, and *MemCoRe-periodic*. *MemCoRe-periodic* refers to the variant that uses (i) coherence accounting, (ii) periodic halting decision, and (iii) halt/resume via CTI trigger. The experiments conducted are (1) *MemPol* with a 6.25 $\mu s$ polling period; (2) *MemCoRe-periodic* with 6.25 $\mu s$ for both halting decision and replenishment period; (3) *MemCoRe-periodic* with 2.2 $\mu s$; (4) *MemCoRe* with 6.25 $\mu s$ replenishment period and asynchronous halting; and (5) *MemCoRe* with 1 $\mu s$. For all regulators, the target bandwidth is set to 120 MB/s. During the runs, the tracer is tasked to collect the regulated bus activity.

Fig. 5 illustrates the bus activity as recorded by the tracer under the different regulators. In each inset, the y-axis shows the bandwidth achieved for each bin size of 1 $\mu s$. The target bandwidth is shown by the dashed red line, while the smoothed measured bandwidth is drawn in blue (moving average over 50 $\mu s$).

The first observation is that *MemPol* and *MemCoRe-periodic* 6.25 $\mu s$ behave similarly. More precisely, both regulators display large bandwidth fluctuations ranging from 0 MB/s to $\pm$200 MB/s. This is expected as *MemCoRe-periodic* 6.25 $\mu s$ is functionally equivalent to *MemPol* but implemented on the PL-side. The second observation is that running *MemCoRe-periodic* with a shorter period (here 2.2 $\mu s$) reduces the fluctuations to $\pm$85 MB/s to $\pm$150 MB/s. The third observation is that even for a large period (6.25 $\mu s$), enabling asynchronous halting is effective in reducing fluctuations. Finally, reducing the replenishment period to 1 $\mu s$ and enabling asynchronous halting provide the tightest regulation.

The results indicate that *MemCoRe* successfully tames the overshooting normally observable with *MemPol*.

### D. Taming Overshooting: Pragmatic Benchmarks

In addition to showing the *MemCoRe* behavior under a predictable synthetic scenario (see Sec. VI-C), we evaluate *MemCoRe* when regulating memory-intensive pragmatic benchmarks. The SD-VBS benchmarks are run with the *qcif* input size[4] and are (1) regulated by *MemPol* with a polling period of 6.25 $\mu s$; (2) regulated by *MemCoRe* with a polling period of 1.5 $\mu s$ and asynchronous halting; and (3) unregulated. As in Sec. VI-C, the tracer is configured to record the full bus activity.

Overall, the traces displayed in Figure 6 confirm the conclusion of the previous experiment: *MemCoRe* is more capable of taming overshooting than the coarser-grained *MemPol*. For instance, due to its similar memory access pattern to IsolBench's bandwidth, `disparity`'s traces resemble those shown in Figure 5. However, unlike the previous benchmark, these pragmatic benchmarks reveal the capability of *MemCoRe* to handle sudden request spikes. The `sift` benchmark is a prime example. As shown in its unregulated trace (top-most), the first half of its execution is characterized by six brief transaction bursts. The same bursts can still be observed in the *MemPol* regulated trace (middle inset) and the *MemCoRe* regulated trace (bottom-most). However, in the latter's case, the burst is mostly contained below the target bandwidth.

### E. Regulators Overhead

Inherently, PMC-based bandwidth regulators are likely to inflate the execution time of the tasks they regulate due to the overheads they introduce. This experiment aims to quantify the effect of *MemCoRe* operations on the execution of tasks compared to *MemPol*.

To this end, we express the execution time inflation as a slowdown compared to when running unregulated (*i.e.*, the ratio between the execution time under regulation and without regulation). In particular, we are interested in the slowdown of the benchmarks when regulated by *MemPol* and *MemCoRe* with the same bandwidth target. In this experiment, all SD-VBS benchmarks use a *vga* input size and a target bandwidth of 150 MB/s and 300 MB/s. For completeness, we conduct and present our experiment considering all the available benchmarks. Nonetheless, it can be observed that some of these are not memory-bound and, therefore, only exhibit negligible slowdown when running under regulation. This is the case for `sift`, `stitch`, `localization`, and `texture synthesis`.

The results in Figure 7 show that the largest gains were obtained when running `disparity` and `tracking`. In the case of `disparity`, a sizeable slowdown reduction can be observed for the lowest bandwidth target. The same effect is also visible, albeit harder to quantify, by looking at the length of the traces in Figure 6. All other benchmarks achieve

[4]The size was selected to prevent the tracer's buffer from overflowing.

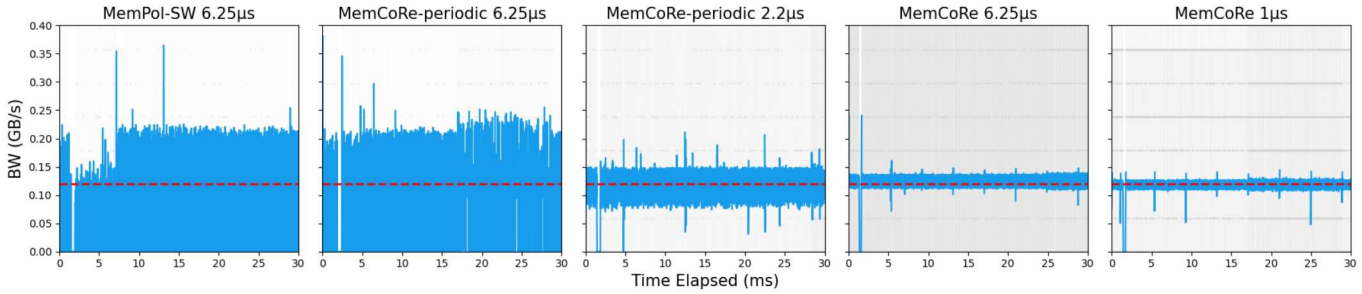


Fig. 5: Bandwidth traces of different regulators. The horizontal red line is the setpoint, the vertical blue lines are the $50\mu s$ moving average of the memory bandwidth, and the vertical gray lines are individual cache line transactions. From left to right, the performance shows the gradual improvement in setpoint maintenance from *MemPol-SW* to *MemCoRe*. The first major improvement is due to the higher regulation frequency enabled by *MemCoRe-periodic* (form $6.25\mu s$ to $2.2\mu s$) achieved by using coherence to monitor BW instead of PMC. The second major improvement is achieved by leveraging the asynchronous halt capabilities of *MemCoRe*.

comparable performance under the operation of the two regulators. All in all, these results highlight that *MemCoRe* always behaves identically or better than *MemPol*.

### F. PL-side Resource Utilization

TABLE II: Resource utilization of the PL-side to implement *MemCoRe* (Sec. VI-D version) on the ZCU102's PL-side for a target frequency of 100 MHz.

| | LUT | | FF | | BRAM | | DSP | |
|---|---|---|---|---|---|---|---|---|
| | Total | % | Total | % | Total | % | Total | % |
| *MemCoRe* w/ Tracer | 17747 | 6.48 | 23116 | 4.22 | 25.5 | 2.8 | 3 | 0.12 |
| *MemCoRe* | 1545 | 0.56 | 1364 | 0.25 | 0 | 0 | 0 | 0 |
| Memory Controller | 15400 | 5.62 | 20917 | 3.81 | 25.5 | 2.8 | 3 | 0.12 |
| Tracer Logic | 802 | 0.29 | 835 | 0.15 | 0 | 0 | 0 | 0 |

*MemCoRe* uses a marginal amount of resources on the PL-side. In fact, once the tracer and the PL-side memory controller are removed[5], only 0.56% of the available look-up tables (LUT) are used. This negligible footprint means that *MemCoRe* can be added to any existing PL-side design with virtually no impact. A breakdown of the PL-side's resource utilization for *MemCoRe* that includes the tracer used in Sec. VI-D is reported in Table II. Likewise, the ACE port does not have to be exclusively allocated to *MemCoRe*. The latter is a passive observer and, hence, can be attached as a probe to an actively coherent hardware module. However, *MemCoRe* needs two AXI ports for its configuration port and to access the CoreSight registers located on the PS-side. These two ports do not need to be exclusively allocated for *MemCoRe*. Interconnects can be used to multiplex the *MemCoRe*'s AXI ports with other PL-side components.

## VII. Discussion

The evaluation showcases many of *MemCoRe*'s advantages over *MemPol*. Our approach is particularly effective in reducing the amplitude of overshooting. However, *MemCoRe* cannot completely eliminate them. The traces' raw data reveals that small overshooting still occurs at a scale that cannot easily be visualized. This might be due to a small delay between an LLC miss occurring and *MemCoRe* receiving the snoop.

Despite the positive results presented in Section VI, *MemCoRe* lacks some of *MemPol*'s and *MemGuard*'s functionalities. For instance, unlike *MemPol*, *MemCoRe* cannot immediately take LLC cache write-backs into account for its regulation as write-back events are not broadcasted to the members of the coherence domain. On the one hand, this limitation can be partially overcome by monitoring the PMCs counting only write operations from *MemCoRe*, thus operating *MemCoRe* in a *mixed mode*—refills counted via coherence traffic and write-backs counted via PMCs. On the other hand, it might be possible to *manipulate* snoop requests to inform the PL of write-back events. The feasibility of this approach, however, requires additional research. We leave this extension of *MemCoRe* as future work.

Moreover, the utilization of interrupts (à-la *MemGuard*) instead of CTI trigger lines might be necessary to regulate the activity of all the application cores in future platforms with an ever-increasing number of PEs.

In particular, the number of debug triggers limits the approach's scalability to only a few cores and a few signals. With that regard, using interrupts to throttle the (additional) cores may be the way forward. In this case, CTI-based regulation can be applied to the cores hosting applications that exhibit significant bandwidth peaks, allowing *MemCoRe* 's rapid response time to be optimally utilized for overshoot prevention. Meanwhile, the remaining cores can be regulated using FIQs/IRQs.

Furthermore, the idea of adopting interrupts-based throttling is reinforced by the absence of a mechanism for the software to mask debug halt/resume signals during specific execution phases (*e.g.*, critical sections). However, in light of *MemCoRe*'s reduced regulation period (and lack of visible overshooting), the need for a masking mechanism is not a pressing concern. Evaluating the cost trade-off of utilizing interrupts for many-core architectures is left as future work.

[5]Their presence is only necessary for the purpose of our evaluation.

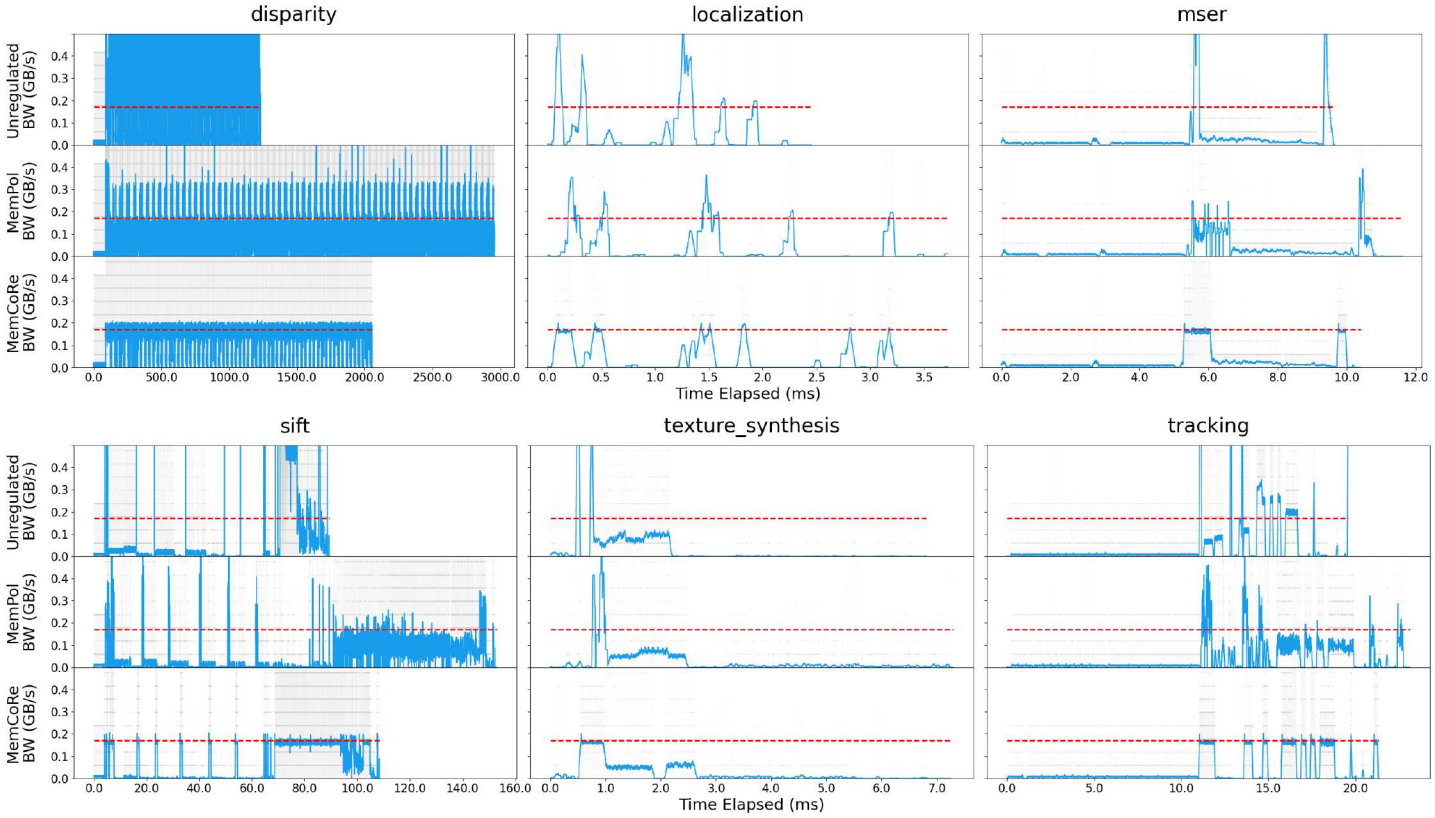


Fig. 6: Bandwidth traces of SD-VBS. The red line indicates the setpoint and the blue line is the 50 $\mu s$ moving average of the memory bandwidth. Both regulations have the same bandwidth setpoint. The top row shows the memory bandwidth activity when no regulation is present, the middle when *MemPol* regulates, and the bottom when *MemCoRe* regulates.

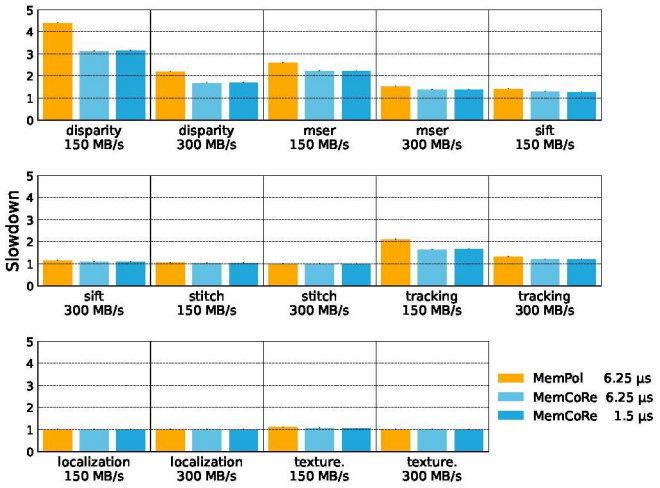


Fig. 7: Relative slowdown by regulation. Each benchmark is evaluated under two bandwidth targets (150 MB/s, 300 MB/s). The regulation period of *MemPol* is 6.25 $\mu s$. Due to *MemCoRe* employing async-halting, the period for *MemCoRe* is for resumption (6.25 $\mu s$, 1.5 $\mu s$). The $y$-axis represents the slowdown ($\times$) factor where $1\times$ means the application has the same runtime as in the unregulated run.

## VIII. Related Works

Memory inference regulation techniques for MPSoC platforms can be classified into software and hardware approaches. Software techniques offer greater flexibility and wider applicability across commercial off-the-shelf (COTS) platforms, whereas hardware approaches enable finer monitoring granularity.

*MemGuard* [1], a PMC-based regulation, is the first software approach, which has sparked numerous subsequent investigations [18], [19], [26], including the implementations of *MemGuard* at the hypervisor level without modifying the host OS. The support for the separate regulation on each core for reading or writing memory traffic is extended by the work [14]. The work [27] also adds support for the protection against Cache Bank-Aware Denial-of-Service Attacks.

While the memory bandwidth derived from PMCs serves as a proxy for the effective workload on the interconnect and DRAM memory controller, leveraging the dedicated performance counters exposed by the memory controller itself enables more precise measurement of memory utilization [28], [29]. However, hardware vendors seldom expose the internals of memory controllers, and only a limited number of MPSoC platforms (primarily from NXP *e.g.*, [30], [31]) provide access to a subset of memory controller-specific performance counters. The aforementioned works typically use PMCs as count-

down counters that generate interrupts upon depletion to enact regulation actions. Instead, methods that periodically *sample* the PMCs' values [29] enable broader regulation strategies, such as building distribution-driven memory regulation [32]. Another sampling-based method, *MemPol*, not only takes consideration of the global memory bandwidth while distributing the bandwidth to each core, but it also moves the controlling logic to other processing elements on the SoC to reduce the overhead [10].

Modern MPSoCs offer QoS enforcement mechanisms such as the one from Arm [7], [33]. Works exist that utilize those offered primitives to implement bandwidth regulators [13], [28], [34]–[36]. However, since these primitives still monitor bandwidth consumption at the platform interconnect level, they cannot be immediately used to monitor/regulate the traffic of a specific core.

From the hardware side, the work [4] develops a custom drop-in hardware module to regulate the bandwidth directly at the hardware level to achieve finer monitoring granularity. The work [37] proposes an FPGA module that can monitor and regulate different types of requests and is deployed on a prototype RISC-V design [38]. Conversely, our work requires no architectural modifications to commercially available PS-PL platforms.

To reduce the worst-case latency of memory transactions facing multi-core contention, various adaptations for the memory controller are proposed [39]–[43]. To improve the timely predictability at the memory interconnect level, Time Division Multiplexing hardware is also proposed [44]–[47].

On PS-PL platforms, the design principle of *cache coherence backstabbing* inspires our work. *SchIM* [48] can schedule individual memory transactions by redirecting CPU memory transactions through the FPGA. The work [49] proposes an FPGA-based closed-loop controller. In this case, the authors propose attaching an external FPGA to the debug-trace port of the multi-core system to be regulated. As such, *MemCoRe* is the first work that leverages the passive analysis of coherence traffic and direct halt/resume signaling to push the envelope of memory bandwidth regulation in tightly coupled PS-PL platforms.

## IX. Conclusion

*MemCoRe* is a novel hardware-assisted memory bandwidth regulation technique that leverages the PL on modern SoCs to monitor cache coherence traffic and throttle cores accordingly. By observing coherence snoops directly in hardware, *MemCoRe* avoids the bottleneck of serially reading performance counters faced by prior software approaches.

*MemCoRe* improves over state-of-the-art memory bandwidth regulators that can be instantiated in commercially available PS-PL platforms, pushing the regulation granularity to *nanosecond*-scale, solving the overshooting problem, and enabling address-aware bandwidth throttling strategies. Overall, *MemCoRe* makes a compelling case for the potential of leveraging the PL in modern SoCs to enable fine-grained, precise, and flexible memory bandwidth regulation.

## X. Acknowledgments

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CNS-2238476. Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

## References

[1] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Trans. Computers*, vol. 65, no. 2, pp. 562–576, 2016. [Online]. Available: https://doi.org/10.1109/TC.2015.2425889

[2] H. Yun, W. Ali, S. Gondi, and S. Biswas, "BWLOCK: A dynamic memory access control framework for soft real-time applications on multicore platforms," *IEEE Trans. Computers*, vol. 66, no. 7, pp. 1247–1252, 2017. [Online]. Available: https://doi.org/10.1109/TC.2016.2640961

[3] Y. Zhou and D. Wentzlaff, "MITTS: memory inter-arrival time traffic shaping," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 2016, pp. 532–544. [Online]. Available: https://doi.org/10.1109/ISCA.2016.53

[4] F. Farshchi, Q. Huang, and H. Yun, "BRU: bandwidth regulation unit for real-time multicore processors," in *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2020, Sydney, Australia, April 21-24, 2020*. IEEE, 2020, pp. 364–375. [Online]. Available: https://doi.org/10.1109/RTAS48715.2020.00011

[5] Intel, "Resource Director Technology," Accessed: 2024-01-01, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html

[6] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A closer look at intel resource director technology (rdt)," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 127–139. [Online]. Available: https://doi.org/10.1145/3534879.3534882

[7] ARM, "Quality of Service in ARM Systems: An Overview," Accessed: 2024-01-01, 2014. [Online]. Available: https://community.arm.com/arm-community-blogs/b/soc-design-and-simulation-blog/posts/quality-of-service-in-arm-systems-an-overview

[8] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso, "Profile-driven memory bandwidth management for accelerators and cpus in qos-enabled platforms," *Real-Time Syst.*, vol. 58, no. 3, p. 235–274, Sep. 2022. [Online]. Available: https://doi.org/10.1007/s11241-022-09382-x

[9] ARM, "Arm Architecture Reference Manual Supplement. Memory System Resource Partitioning and Monitoring (MPAM) for Armv8-A," Accessed: 2024-01-01, 2022. [Online]. Available: https://developer.arm.com/docs/ddi0598/db/

[10] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "Mempol: Policing core memory bandwidth from outside of the cores," in *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2023, San Antonio, TX, USA, May 9-12, 2023*. IEEE, 2023, pp. 235–248. [Online]. Available: https://doi.org/10.1109/RTAS58335.2023.00026

[11] Xilinx, "Zynq UltraScale+ Device Technical Reference Manual," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf

[12] G. Schwäricke, R. Tabish, R. Pellizzoni, R. Mancuso, A. Bastoni, A. Zuepke, and M. Caccamo, "A real-time virtio-based framework for predictable inter-vm communication," in *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*. IEEE, 2021, pp. 27–40. [Online]. Available: https://doi.org/10.1109/RTSS52674.2021.00015

[13] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla, "Leveraging hardware qos to control contention in the xilinx zynq ultrascale+ mpsoc," in *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, ser. LIPIcs, B. B. Brandenburg, Ed., vol. 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 3:1–3:26. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2021.3

[14] M. G. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, B. B. Brandenburg, Ed. IEEE, 2019, pp. 357–367. [Online]. Available: https://doi.org/10.1109/RTAS.2019.00037

[15] ARM, "AMBA AXI and ACE Protocol Specification," 2013. [Online]. Available: https://developer.arm.com/documentation/ihi0022/e/

[16] S. Roozkhosh, D. Hoornaert, and R. Mancuso, "CAESAR: coherence-aided elective and seamless alternative routing via on-chip FPGA," in *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*. IEEE, 2022, pp. 356–369. [Online]. Available: https://doi.org/10.1109/RTSS55097.2022.00038

[17] C. Ravishankar and J. Goodman, "Cache implementation for multiple microprocessors," in *Proceedings of IEEE COMPCON*, February 1983, pp. 346–350. [Online]. Available: https://www.cs.ucr.edu/~ravi/Papers/NWConf/ravishankar_83.pdf

[18] P. Modica, A. Biondi, G. C. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *IEEE International Conference on Industrial Technology, ICIT 2018, Lyon, France, February 20-22, 2018*. IEEE, 2018, pp. 1651–1657. [Online]. Available: https://doi.org/10.1109/ICIT.2018.8352429

[19] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Workshop on Next Generation Real-Time Embedded Systems, NG-RES@HiPEAC 2020, January 21, 2020, Bologna, Italy*, ser. OASIcs, M. Bertogna and F. Terraneo, Eds., vol. 77. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 3:1–3:14. [Online]. Available: https://doi.org/10.4230/OASIcs.NG-RES.2020.3

[20] ARM, "Arm Architecture Reference Manual for A-profile architecture," Accessed: 2024-01-01, 2016. [Online]. Available: https://developer.arm.com/docs/ddi0487/ak/

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

[22] ARM, "Arm coresight architectural specification v3.0," 2022. [Online]. Available: https://developer.arm.com/documentation/ihi0029/latest/

[23] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. M. Louie, S. Garcia, S. J. Belongie, and M. B. Taylor, "SD-VBS: the san diego vision benchmark suite," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009, October 4-6, 2009, Austin, TX, USA*. IEEE Computer Society, 2009, pp. 55–64. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306794

[24] 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), Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, pp. 161–172. [Online]. Available: https://doi.org/10.1109/RTAS.2016.7461361

[25] 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 *RTNS 2022: The 30th International Conference on Real-Time Networks and Systems, Paris, France, June 7 - 8, 2022*, Y. Abdeddaïm, L. Cucu-Grosjean, G. Nelissen, and L. Pautet, Eds. ACM, 2022, pp. 184–195. [Online]. Available: https://doi.org/10.1145/3534879.3534888

[26] N. Dagieu, A. Spyridakis, and D. Raho, "Memguard: A memory bandwith management in mixed criticality virtualized systems memguard KVM scheduling," in *10th Int. Conf. on Mobile Ubiquitous Comput., Syst., Services and Technologies (UBICOMM)*, 2016, pp. 21–27. [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=ubicomm_2016_1_40_10072

[27] M. G. Bechtel and H. Yun, "Cache bank-aware denial-of-service attacks on multicore ARM processors," in *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2023, San Antonio, TX, USA, May 9-12, 2023*. IEEE, 2023, pp. 198–208. [Online]. Available: https://doi.org/10.1109/RTAS58335.2023.00023

[28] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso, "E-warp: A system-wide framework for memory bandwidth profiling and management," in *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*. IEEE, 2020, pp. 345–357. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00039

[29] A. Saeed, D. Dasari, D. Ziegenbein, V. Rajasekaran, F. Rehm, M. Pressler, A. Hamann, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Memory utilization-based dynamic bandwidth regulation for temporal isolation in multi-cores," in *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, Milano, Italy, May 4-6, 2022*. IEEE, 2022, pp. 133–145. [Online]. Available: https://doi.org/10.1109/RTAS54340.2022.00019

[30] NXP, "NXP S32V234SBC," Accessed: 2024-01-01. [Online]. Available: https://www.nxp.com/design/development-boards/automotive-development-platforms/s32v-mpu-platforms/s32v2-vision-and-sensor-fusion-low-cost-evaluation-board:SBC-S32V234

[31] ——, "NXP S32G2," Accessed: 2024-01-01., 2024. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/s32g-vehicle-network-processors/s32g2-processors-for-vehicle-networking:S32G2

[32] A. Saeed, D. Hoornaert, D. Dasari, D. Ziegenbein, D. Mueller-Gritschneder, U. Schlichtmann, A. Gerstlauer, and R. Mancuso, "Memory latency distribution-driven regulation for temporal isolation in mpsocs," in *35th Euromicro Conference on Real-Time Systems, ECRTS 2023, July 11-14, 2023, Vienna, Austria*, ser. LIPIcs, A. V. Papadopoulos, Ed., vol. 262. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 4:1–4:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2023.4

[33] ARM, "ARM CoreLink QoS-400 Network Interconnect Advanced Quality of Service r1p0," Accessed: 2024-01-01, 2016. [Online]. Available: https://developer.arm.com/docs/dsu0026/f/

[34] P. Houdek, M. Sojka, and Z. Hanzálek, "Towards predictable execution model on arm-based heterogeneous platforms," in *26th IEEE International Symposium on Industrial Electronics, ISIE 2017, Edinburgh, United Kingdom, June 19-21, 2017*. IEEE, 2017, pp. 1297–1302. [Online]. Available: https://doi.org/10.1109/ISIE.2017.8001432

[35] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling i/o-related memory contention in COTS heterogeneous platforms," *Softw. Pract. Exp.*, vol. 52, no. 5, pp. 1095–1113, 2022. [Online]. Available: https://doi.org/10.1002/spe.3053

[36] S. Garcia-Esteban, A. Serrano-Cases, J. Abella, E. Mezzetti, and F. J. Cazorla, "Quasi isolation qos setups to control mpsoc contention in integrated software architectures," in *35th Euromicro Conference on Real-Time Systems, ECRTS 2023, July 11-14, 2023, Vienna, Austria*, ser. LIPIcs, A. V. Papadopoulos, Ed., vol. 262. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 5:1–5:25. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2023.5

[37] J. Cardona, C. Hernández, J. Abella, and F. J. Cazorla, "Maximum-contention control unit (MCCU): resource access count and contention time enforcement," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, J. Teich and F. Fummi, Eds. IEEE, 2019, pp. 710–715. [Online]. Available: https://doi.org/10.23919/DATE.2019.8715155

[38] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. Le Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, "De-RISC: the first RISC-V space-grade platform for safety-critical systems," in *2021 IEEE Space Computing Conference (SCC)*, 2021, pp. 17–26. [Online]. Available: https://ieeexplore.ieee.org/document/9546286

[39] R. Mirosanlou, M. Hassan, and R. Pellizzoni, "Drambulism: Balancing performance and predictability through dynamic pipelining," in *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2020, Sydney, Australia, April 21-24, 2020*. IEEE, 2020, pp. 82–94. [Online]. Available: https://doi.org/10.1109/RTAS48715.2020.00-15

[40] M. Hassan, H. D. Patel, and R. Pellizzoni, "PMC: A requirement-aware DRAM controller for multicore mixed criticality systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, pp. 100:1–100:28, 2017. [Online]. Available: https://doi.org/10.1145/3019611

[41] 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, CPSNA 2015, Kowloon, Hong Kong, China, August 19-21, 2015*. IEEE Computer Society, 2015, pp. 86–93. [Online]. Available: https://doi.org/10.1109/CPSNA.2015.24

[42] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a predictable SDRAM memory controller," in *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2007, Salzburg, Austria, September 30 - October 3, 2007*, S. Ha, K. Choi, N. D. Dutt, and J. Teich, Eds. ACM, 2007, pp. 251–256. [Online]. Available: https://doi.org/10.1145/1289816.1289877

[43] A. F. de Lecea, M. Hassan, E. Mezzetti, J. Abella, and F. J. Cazorla, "Improving timing-related guarantees for main memory in multicore critical embedded systems," in *IEEE Real-Time Systems Symposium, RTSS 2023, Taipei, Taiwan, December 5-8, 2023*. IEEE, 2023, pp. 265–278. [Online]. Available: https://doi.org/10.1109/RTSS59052.2023.00031

[44] F. Hebbache, M. Jan, F. Brandner, and L. Pautet, "Shedding the shackles of time-division multiplexing," in *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 2018, pp. 456–468. [Online]. Available: https://doi.org/10.1109/RTSS.2018.00059

[45] M. Jun, K. Bang, H. Lee, N. Chang, and E. Chung, "Slack-based bus arbitration scheme for soft real-time constrained embedded systems," in *Proceedings of the 12th Conference on Asia South Pacific Design Automation, ASP-DAC 2007, Yokohama, Japan, January 23-26, 2007*. IEEE Computer Society, 2007, pp. 159–164. [Online]. Available: https://doi.org/10.1109/ASPDAC.2007.357979

[46] Y. Li, B. Akesson, and K. Goossens, "Architecture and analysis of a dynamically-scheduled real-time memory controller," *Real Time Syst.*, vol. 52, no. 5, pp. 675–729, 2016. [Online]. Available: https://doi.org/10.1007/s11241-015-9235-y

[47] A. Kostrzewa, S. Saidi, and R. Ernst, "Slack-based resource arbitration for real-time networks-on-chip," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, L. Fanucci and J. Teich, Eds. IEEE, 2016, pp. 1012–1017. [Online]. Available: https://ieeexplore.ieee.org/document/7459454/

[48] 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, July 5-9, 2021, Virtual Conference*, ser. LIPIcs, B. B. Brandenburg, Ed., vol. 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 2:1–2:22. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2021.2

[49] J. Freitag and S. Uhrig, "Closed loop controller for multicore real-time systems," in *Architecture of Computing Systems - ARCS 2018 - 31st International Conference, Braunschweig, Germany, April 9-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, M. Berekovic, R. Buchty, H. Hamann, D. Koch, and T. Pionteck, Eds., vol. 10793. Springer, 2018, pp. 45–56. [Online]. Available: https://doi.org/10.1007/978-3-319-77610-1_4