Predictable Memory Bandwidth Regulation for DynamIQ Arm Systems

Ashutosh Pradhan, Daniele Ottaviano, Yi Jiang, Haozheng Huang,
Jiajia Zhang, Alexander Zuepke, Andrea Bastoni, Marco Caccamo
Technical University of Munich
{ashutosh.pradhan, daniele.ottaviano, yi9.jiang, haozheng.huang, jiajia.zhang, alex.zuepke, andrea.bastoni, mcaccamo}@tum.de

***Abstract*—Modern real-time embedded systems increasingly rely on Arm-based Multiprocessor System-on-Chip (MPSoC) architectures to support demanding applications, such as AI workloads and high-speed control systems. To mitigate interference among co-running applications and enforce predictable behavior, software memory bandwidth regulation strategies based on hardware performance counters have been proposed to manage concurrent memory accesses among CPUs. However, such mechanisms (*e.g.*, *MemGuard*, *MemPol*) have been mostly evaluated on previous generations of 64-bit Arm systems, *i.e.*, Cortex-A53, A57, or A72.**

Newer Arm MPSoCs, featuring clusters of Cortex-A55, A76, and A78 cores, introduce (i) the Arm DynamIQ Shared Unit (DSU), (ii) additional cache levels, and (iii) new memory controllers optimized to manage high-bandwidth transactions from accelerators (*e.g.*, GPUs, TPUs). Extending software-based memory bandwidth regulation strategies to newer Arm cores is non-trivial and requires careful characterization of the newly introduced performance counters and the achievable worst-case memory bandwidth (sustainable memory bandwidth).

This paper systematically investigates software-based memory bandwidth regulation on DSU-equipped Arm MPSoCs, using the Rockchip RK3588 and NVIDIA AGX Orin as two representative platforms. We empirically evaluate the impact of the new memory hierarchy on sustainable memory bandwidth and the accuracy of performance monitoring counters. We derive updated memory bandwidth regulation models and assess their behavior when applied to *MemGuard* and *MemPol*. Our results show that software-based memory bandwidth regulation strategies can be successfully applied to newer platforms. However, the different selection of performance counters might result in pessimistic regulation models.

***Index Terms*—real-time system, multi-core, memory bandwidth regulation, performance monitoring counters, Arm**

I. Introduction

The increasing computational demands of modern industrial applications, such as AI-based workloads and high-speed controllers, have driven the adoption of highly complex embedded platforms. These systems are predominantly built upon Multiprocessor System-on-Chip (MPSoC) architectures that integrate a diverse array of processing elements, including multiple CPU cores and specialized accelerators (*e.g.*, GPUs, TPUs), connected to a complex shared memory subsystem. While these heterogeneous platforms enable remarkable performance and energy efficiency, their complexity introduces significant challenges in system analysis, particularly for real-time applications.

One of the most critical sources of unpredictability in MPSoCs is contention in the shared memory subsystem. When multiple applications execute concurrently on the same platform, they inevitably compete for access to the memory controller. This leads to unpredictable interference that can degrade system determinism and violate real-time constraints. To mitigate this issue, the real-time systems community has explored various techniques for regulating memory accesses. Among these, software-based memory bandwidth regulation leveraging performance monitoring counters (PMCs) has gained significant attention due to its applicability across a wide range of commercial-off-the-shelf (COTS) MPSoCs, which are typically equipped with performance monitoring units (PMUs). Notable software-based approaches such as *MemGuard* [1], [2] and *MemPol* [3], [4] have demonstrated the feasibility of enforcing PMC-based memory bandwidth regulation on COTS hardware with outstanding performance and limited overhead.

However, existing software-based memory bandwidth regulation mechanisms have primarily been designed and evaluated on MPSoCs featuring earlier generations of Arm processors, specifically Cortex-A53, A57, and A72 cores [1]–[4]. While these architectures have been widely deployed over the past decade, newer Arm core families, such as Cortex-A55, A76, and A78, are now becoming the standard for next-generation platforms. These newer cores introduce significant architectural changes, including an updated set of PMCs, redesigned memory controllers, and deeper cache hierarchies. A key innovation is the Arm *DynamIQ Shared Unit* (DSU) [5], which fundamentally changes how clusters of cores interact with shared memory. Acting as a central interconnect, the DSU integrates a per-cluster L3 cache, enabling more flexible core-cluster management and impacting the design and applicability of memory bandwidth regulation strategies.

This paper investigates the feasibility and performance of software-based memory bandwidth regulation on modern DSU-based MPSoCs. Specifically, we analyze, port, and evaluate *MemGuard* and *MemPol* on two recent Arm-based COTS platforms, the Rockchip RK3588 [6] and the NVIDIA AGX Orin [7]. Our study answers these key research questions:

- Analyzing how modern memory subsystems affect the worst-case memory bandwidth (sustainable bandwidth) compared to previous Arm-based MPSoCs, and the resulting impact on bandwidth utilization.

- Investigating PMCs on new Arm cores, comparing their behavior to earlier cores, and modeling their suitability for real-time memory bandwidth regulation strategies.
- Proposing new memory bandwidth regulation schemes for Cortex-A55, A76, and A78 cores.
- Evaluating the effectiveness of memory bandwidth regulation on DSU-equipped MPSoCs.

In the following, Sec. II discusses previous work and the architectural evolution of Arm MPSoCs. Sec. III presents an empirical assessment of the sustainable memory bandwidth of the evaluated platforms. Sec. IV analyzes the granularity and accuracy of PMCs available in recent Arm cores and the DSU. Sec. V explores different regulation models aligned with existing software-based mechanisms such as *MemGuard* and *MemPol*. Section VI details the implementation details and parameter settings of *MemGuard* and *MemPol* for the evaluation. Sec. VII evaluates the proposed regulation models by applying them to *MemGuard* and *MemPol* on the selected platforms. Conclusion are discussed in Sec. VIII.

II. Background and Related Works

We focus on memory bandwidth regulation techniques that do not require hardware modifications and that can be flexibly deployed on COTS MPSoCs. The approaches can be broadly categorized into hardware-assisted and software-based.

A. Hardware-Assisted Memory Bandwidth Regulation

Hardware-assisted memory bandwidth regulation aims to control system-wide memory traffic by integrating quality-of-service (QoS) features at the interconnect and memory controller levels. These mechanisms typically implement priority-based arbitration, dynamically adjusting memory access privileges for different cores and peripherals [8]. While these hardware primitives have been explored for bandwidth regulation in previous works [9]–[12], their effectiveness is often limited, as memory traffic is tracked at the interconnect level, making it difficult to attribute usage to individual cores.

Intel's Resource Director Technology (RDT) [13] and Arm's Memory Partitioning and Monitoring (MPAM) [14] offer configurable resource allocation policies, allowing memory access restrictions at finer levels of granularity. However, these mechanisms remain yet sparsely adopted in commercial embedded MPSoCs. Intel RDT is available in recent SoCs and incurs minimal software overhead, but it falls short of providing the expected level of protection for real-time systems [15]. Meanwhile, Arm MPAM has yet to be implemented in any commercially available SoC. Moreover, its specification defines all control interfaces as optional, leading to inconsistent implementations across platforms.

B. Software-Based Memory Bandwidth Regulation

Software-based memory bandwidth regulation techniques leverage PMCs as *proxies* to monitor and control memory bandwidth consumption of specific tasks or whole cores at the operating system (OS) or hypervisor level. These approaches do not require specialized hardware features and are applicable to a broad range of COTS MPSoCs. Furthermore, these approaches use per-core PMCs, allowing easier deployment and control compared to interconnect-level regulation. The key challenge of these techniques is balancing fine-grained enforcement with system overhead. While short regulation periods improve control accuracy, they also increase computational overhead, particularly when enforcement relies on frequent interrupts.

1) MemGuard*:* *MemGuard* [1], [2] is the first PMC-based software regulation mechanism designed to enforce per-core memory bandwidth constraints by monitoring memory traffic and applying throttling when necessary. Following the *MemGuard* approach, other works have been proposed over the years [16]–[18]. *MemGuard* assigns a fixed memory budget to each core, defining the maximum number of allowed memory transactions per regulation period. These transactions are tracked through PMCs, which measure key memory activities such as cache refills and DRAM accesses. When a core depletes its allocated budget, the PMU immediately triggers an interrupt and *MemGuard* forces the core to idle until the next replenishment cycle begins. This mechanism effectively reduces resource contention, ensuring that no single core monopolizes shared resources. However, the reliance on frequent interrupts introduces substantial overhead, particularly when attempting to regulate memory traffic at fine granularity (see Fig. 4 and [2]–[4], [19]). Moreover, *MemGuard* can only regulate using the contribution of one single metric (*e.g.*, cache write-backs, cache refills, or memory controller utilization) at a time. While different metrics can be observed using different PMCs, their memory contributions cannot be combined together [20]. This constraint often leads to pessimistic enforcement, as the system must throttle execution conservatively to prevent exceeding the assigned bandwidth.

2) MemPol*:* To address such limitations, *MemPol* [3], [4] introduces a polling-based memory bandwidth regulation mechanism that eliminates the reliance on interrupts, allows the use of multiple performance counters, and significantly reduces overhead on the cores under regulation. *MemPol* continuously monitors memory consumption using a separate processing element to enforce external regulation. This external regulation logic runs on a dedicated core—*e.g.*, a Cortex-R or Cortex-M processor, or as IP block on an FPGA [21]—and continuously polls PMCs without interfering with application execution. If a core exceeds its assigned bandwidth, the regulator uses the cross trigger interfaces (CTI) of the Arm *CoreSight* debug infrastructure to throttle it. Unlike *MemGuard*, *MemPol* supports multi-dimensional monitoring, simultaneously combining metrics obtained from multiple PMC events to regulate the cores, enabling a less pessimistic regulation. Instead of completely stalling a core upon budget exhaustion, *MemPol* implements an on-off throttling mechanism that dynamically adjusts execution time using a sliding-window model or a token-bucket model [21], smoothing out memory usage while maintaining low-latency regulation. By shifting the monitoring and enforcement logic outside the main application cores, *MemPol* removes the high-frequency interrupt overhead and enables microsecond-scale regulation.

3) PMCs on Arm Cortex-A53, A57, and A72 Cores: On these cores, the memory bandwidth generated by each core can be measured reliably by two PMU events: `0x17 l2d_cache_refill` measures all cachelines coming from the memory controller, and `0x18 l2d_cache_wb` measures dirty cachelines eventually written back to memory. Both events are suitable *proxies* for the memory bandwidth generated by the individual cores.

C. Architectural Evolution of Arm MPSoCs

Recent Arm MPSoCs have shifted from traditional multi-cluster architectures to the DynamIQ Shared Unit (DSU)-based designs, introducing greater flexibility in core configurations [22]. Earlier architectures typically followed either homogeneous layouts, where all cores were identical, or *big.LITTLE* clusters, where high-performance and power-efficient cores were grouped separately [23]. The DSU unifies these approaches, enabling both homogeneous and heterogeneous cores to coexist within a single cluster and adding a shared L3 cache for all cores within the cluster. This introduces a more complex memory hierarchy, affecting cache management and software-based memory bandwidth regulation techniques. This architectural shift is accompanied by a transition to newer Arm cores, specifically Cortex-A55, A76, and A78, which replace the Cortex-A53, A57, and A72.

The Cortex-A55 [24] is the successor of the Cortex-A53 and maintains an in-order pipeline. Each core has private L1 instruction and data caches and an optional shared L2 cache, which functions as a victim cache. This configuration enforces an exclusive caching policy, meaning that data is allocated in the L2 cache only when evicted from the L1 data cache and does not reside in both caches simultaneously.

In contrast, Cortex-A76 [25] and A78 [26] are out-of-order cores optimized for performance, commonly paired with low-power A55 cores in heterogeneous DSU clusters. These cores feature inclusive L2 caches, *i.e.*, all L1-resident data also exist in the L2 cache. Cortex-A76AE [27] and A78AE [28] (automotive enhanced) further refine this architecture by introducing lockstep mode, making them suitable for high-reliability applications, such as safety-critical automotive systems.

The DSU acts as the interconnect for all cores in a cluster, integrating a shared L3 cache (256 KB–4 MB, 16-way set-associative). This marks a departure from pre-DSU MPSoCs, where cores in each cluster shared the L2 cache. The DSU features PMCs that provide a consistent view of the memory transactions of all its cores (See Sec. III). However, these PMCs aggregate cluster-wide activity rather than individual core usage, limiting their utility for per-core memory control.

III. Sustainable Memory Bandwidth

To systematically evaluate the impact of the new DSU-based architecture on memory bandwidth regulation, this study stresses the memory subsystem of different cores (*i.e.*, Cortex-A55, A76, and A78AE) available on representative MPSoCs: Rockchip RK3588 [6] and NVIDIA AGX Orin [7].

To characterize the memory access capabilities of each target platform, we evaluate the *sustainable memory bandwidth* [9], which is defined in [4] as the maximum bandwidth that a memory controller can sustain under worst-case memory workload. To assess the sustainable memory bandwidth, we employ the same benchmarking tool [29] that has been previously used in [4]. The benchmark systematically measures the bandwidth capabilities of the DRAM controller by executing a range of memory access patterns and progressively increasing access strides over a memory buffer to trigger worst-case DRAM behaviors, such as frequent row misses within the same DRAM bank [30].

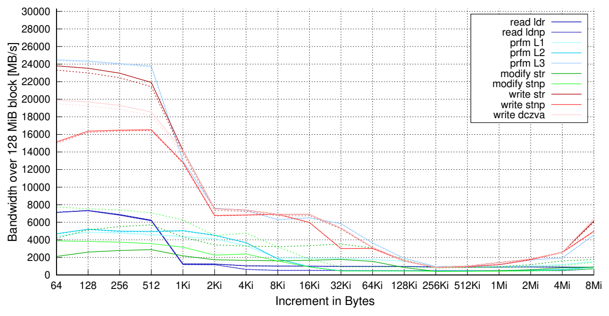
The benchmark iterates over a large memory buffer with different striding patterns, *e.g.*, first cacheline by cacheline, then only every second cacheline, then only every fourth cacheline, *etc.*, and measures the bandwidth for operations on cachelines, denoted as *read*, *write*, and *modify*. Note that Arm cores do not need to fetch a cacheline from memory when the full content of a cacheline is overwritten. We distinguish these "pure" *write* operations from *modify* operations that fetch cachelines from memory due to partial writes to the whole cacheline, *e.g.*, if just one byte in the cacheline is changed. We further test non-temporal loads and stores (*ldnp* and *stnp*), prefetches (*pfrm*) to L1, L2, or L3, and clearing of cachelines by the `dc zva` instruction (*dczva*). We run the benchmarks on Linux on both platforms, ensuring that only essential system processes remain active to minimize external interference. All tests are performed with power management settings configured for maximum performance. Investigation of the effect of power management on memory regulation, along with a comparison to older platforms such as in [31], is left for future work.

A. Rockchip RK3588

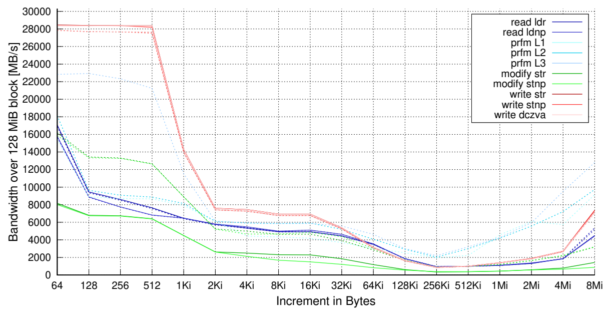
The RK3588 SoC on the Orange Pi 5 Plus board features a *single* DSU cluster combining four Cortex-A76 and four Cortex-A55 cores, offering a relevant testbed for studying memory interference in shared heterogeneous environments.

Our experiments use the mainline Linux kernel version 6.13.3, a buffer size of 128 MiB, mapped as huge pages, and additionally record two DSU PMCs, `0x60 bus_access_rd` and `0x61 bus_access_wr`, to validate the results.[1] We configure all cores to their maximum speed by setting the CPU governor to `performance`. Fig. 1 shows the results for a run on a single Cortex-A55 core resp. on a single Cortex-A76 core on the RK3588 while all other cores are idle. The solid lines show the achieved per-core bandwidth, while the dotted lines represent the sum of the PMC values retrieved at the DSU level for the whole cluster to validate the results. Compared to [3], [4], we improved the benchmark with the `--step-all` option to always access the full memory region during striding, as we discovered that the DSU would otherwise cache the memory accesses at higher step sizes. Note that modify operations

[1]We run `bench -s 128 --huge -c <core> --perf-config 0x60@arm_dsu_0/4,0x61@arm_dsu_0/4 --auto --step-all <op>` on core 0 for Cortex-A55 and on core 4 for Cortex-A76.



(a) Cortex-A55



(b) Cortex-A76

Fig. 1: Memory bandwidth of RK3588. Solid (Dotted) lines show the bandwidth measured by one core (by the DSU PMU).

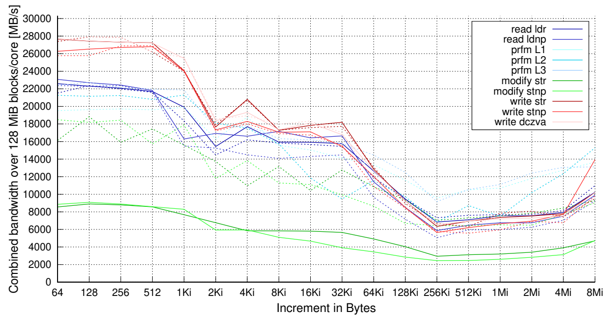


Fig. 2: Combined memory bandwidth of all Cortex-A55 and Cortex-A76 on RK3588. Solid lines show the bandwidth measured by the cores. Dotted lines show the bandwidth measured by the DSU PMU.

comprise both reads and writes, therefore the values reported by the PMC are twice as high as the bandwidth observed by the core. We will discuss this effect in detail in Sec. IV.

On both core types, we observe similar shapes, especially for writes, but larger differences for reads. Initially at a stable level, the bandwidth significantly drops after step size of 512 B. Step size 256 KiB shows a local minimum, after which the values increase again, most likely due to parallelism in the memory controller. The huge differences between min and max bandwidths indicate that the bottleneck is most likely in the memory controller and not at core- or interconnect-level.

On the A55, we observe that both prefetches to L3 and writes achieve the highest possible bandwidth, as both will be executed out-of-order *w.r.t.* the core's in-order pipeline. Instead, prefetches to L1 and L2, reads, and the read-part of modify operations block the core and hit the minimum already at a step size of 1 KiB. The A76 initially shows a higher write bandwidth than the A55, but then plunges similarly. The read bandwidth is significantly better and takes longer to converge to the minimum, most likely due to the out-of-order pipeline and load-store unit of the core.

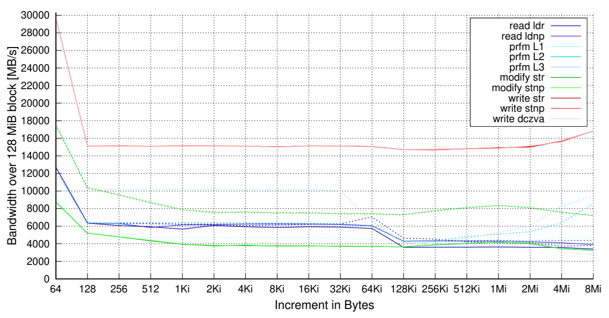
For both cores, at step size 256 KiB, we can observe a local minimum of around 900 MB/s for reading or writing. We assume that this is the sustainable bandwidth of RK3588 on the Orange Pi 5 Plus. However, our worst-case access patterns that lead to this low bandwidth are not representative of a multicore system running independent tasks. As the out-of-order read results of the A76 show, the memory controller *can* sustain a higher bandwidth for a mix of memory accesses from different cores. To test this hypothesis, we ran instances of the benchmark in parallel and in lockstep on all cores. The results are shown in Fig. 2. Notably, all cores achieve a total bandwidth of 2400 MB/s shared among the cores at the *same* step size 256 KiB. We see that the limitation of a core's bandwidth is due to the specific step size, and not due to interference from the other cores. However, these worst-case memory access patterns in this experiment are still much more pessimistic than the access patterns of real-world independent tasks. Therefore, looking at the distribution of bandwidth, in the rest of the paper we made the more realistic assumption that a bandwidth of 2 GB/s can be achieved in most cases. Note that the bandwidth assessment will likely show different results for other boards equipped with the RK3588.

B. NVIDIA Orin

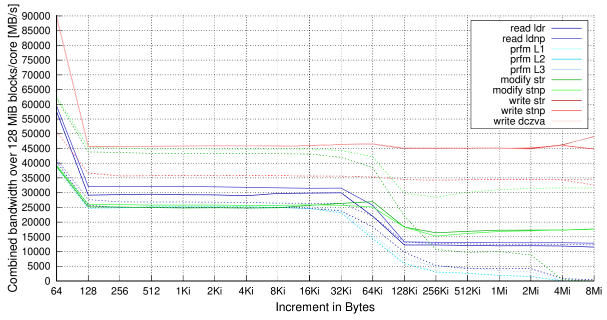
We use the NVIDIA Tegra T234 (Orin) SoC on the NVIDIA Jetson AGX Orin Developer Kit as it was also adopted in previous studies [3], [4]. The Orin incorporates three DSU clusters, each consisting of four Cortex-A78AE cores, providing an opportunity to assess how DSU-based memory bandwidth regulation scales across multiple clusters. Additionally, the platform includes a 4MB L4 victim cache (NVIDIA Orin SCF—System Cache Fabric), which is shared among the DSUs and is designed to enhance communication between core clusters and the GPU. For our evaluations, we use the 32GB variant of the Orin Developer Kit, the kernel version 5.10.216 (NVIDIA Jetson Linux r35.3.1), a larger buffer size of 2 GiB, and record the related PMCs of the L4 SCF.[2] We configured the system for maximum performance by setting `nvpmodel = 0`, enabling `jetson_clocks` to maximize all frequencies—including the memory bus—and setting the CPU frequency governor to `performance`.

Fig. 3a shows the results of our benchmark running on a single Cortex-A78AE core at maximum speed on the Orin

[2]We run `bench -s 2048 --huge -c 0 --perf-config 0x0600@scf_pmu,0x0610@scf_pmu --auto --step-all <op>`. The SCF PMCs are `0x0600 bus_access_rd` and `0x0610 bus_access_wr`.



(a) Single Cortex-A78AE



(b) Combined 12 Cortex-A78AE

Fig. 3: Single core and combined memory bandwidth of Cortex-A78AE on Orin. Solid lines show the bandwidth measured by one core. Dotted lines show the bandwidth measured by the SCF (L4 cache) PMU.

SoC. We see similar trends as the RK3588 but very different effects. The Orin constantly provides high levels of bandwidth regardless of the step size. We can observe two dips: a sharp one at step size 128 B where the bandwidth halves, which hints at an interleaving of two memory controllers, and another slight dip from 32 KiB to 128 KiB, where reads converge from around 6 GB/s to a minimum of 4 GB/s. Writes stay at around 15 GB/s after the first dip.

The Orin documentation reports a theoretical peak memory bandwidth of 204.8 GB/s. Therefore, one core alone *cannot* max out the memory controller. We, therefore, conduct another experiment where we run the benchmark in parallel and in lockstep on all 12 cores of the Orin. Fig. 3b shows the combined bandwidth shared among all cores. The achievable memory bandwidth shows the same trend and dips. We now observe a minimum of 12 to 13 GB/s for reads and 45 GB/s for writes. The constantly high write bandwidth hints at a large buffer or cache in the memory controller. Note that the combined peak write bandwidth of 90 GB/s is still limited by a bottleneck towards the cores. In fact, we were able to achieve even higher bandwidth, up to 150 GB/s, using memory bandwidth-bound STREAM kernels [32] on the GPU. Note that we cannot fully explain the measurements of the SCF PMCs for reads and modify at high step sizes, but it is likely that the effects correlate with caching in L3 and L4, as we did not run the benchmark with the `--step-all` flag enabled. Based on these results, we assume a sustainable bandwidth of 12 GB/s for reads and 45 GB/s for writes for the rest of this work.

IV. Performance Monitoring Counter Analysis

The type and accuracy of PMCs vary across hardware architectures [33]. We identify the most suitable counters to use for memory bandwidth regulation on DSU-based platforms by analyzing the PMCs available on Cortex-A55, A76, and A78AE cores, as well as the DSU. The documentation of Cortex-A76 and A76AE (Cortex-A78 and A78AE resp.) does not show differences in PMCs and CPU-errata between cores of the same family. We assume therefore that our results also apply to Cortex-A76AE and A78 cores. We evaluate PMCs' accuracy by comparing counter readings against the exact number of memory accesses imposed by controlled workloads.

A. Counter Selection Criteria

Arm processors expose hundreds of PMC event types per core and around forty for the DSU. These counters differ in measurement units: some track events based on cycles (*e.g.*, bus cycles, CPU cycles), while others count memory accesses or transactions. We focus on memory and cache-related events relevant to memory bandwidth regulation, prioritizing those that count memory accesses. Many PMCs count cacheline as a whole—*e.g.*, moving cachelines in the memory hierarchy—while others count each access to a cacheline, *e.g.*, individual loads and stores to a cacheline in the L1 cache. We term *data-access counters* the PMCs that count individual memory accesses, and *cacheline counters* those that track cacheline-level transactions.

For memory bandwidth regulation, cacheline counters are to be preferred because they consistently measure memory and cache transactions with a fixed granularity of 64 bytes. Related are the counters that count memory transactions on the bus. As most memory transactions are to cached memory, and cachelines are typically broken down into a fixed number of transactions, *e.g.*, four transactions of 128 bit widths, these counters are also suitable for accounting. In contrast, *data-access counters* vary in granularity (*e.g.*, 8-bit or 64-bit), making them less reliable for enforcing bandwidth constraints. We focused therefore on *cacheline counters* and we identified such counters among those available on Cortex-A55, A76, A78AE, and in the Arm architecture.

B. PMC Evaluation

We use our benchmarking tool [29] to measure the correlation of cacheline counters with actual memory transactions. We run read, write, and modify operations on a buffer of 64 MiB size. On the RK3588 platform, we run the benchmarks on four out of eight cores (2×A55 and 2×A76) and on the Orin, we run it on two cores on a single DSU cluster (2×A78AE). Running workloads on only a subset of cores allows us to verify that the DSU counters do not record activity from unrelated cores. On the Orin, we limit execution to a

event_id	event_name	read			read_ldnp			prefetch_l1			prefetch_l2			prefetch_l3			modify			modify_prefetch			modify_stnp			write			write_stnp			write_dczva		
		A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78
0x0004	l1d_cache	1	1	1	2	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	4	4	1	4	4	0	1	1
0x0040	l1d_cache_rd	1	1	1	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0041	l1d_cache_wr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	4	4	1	4	4	0	1	1
0x0020	l1d_ws_mode	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	⋈	-	-	0	-	-	⋈	-	-
0x0003	l1d_cache_refill	-	1	1	-	7/8	1	-	0	0	-	0	0	-	0	0	-	1	1	-	3/4	3/8	-	1	1	-	0	0	-	0	0	-	0	0
0x0039	l1d_cache_lmiss_rd	-	-	1	-	-	1	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0
0x0042	l1d_cache_refill_rd	-	1	1	-	7/8	1	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0
0x0043	l1d_cache_refill_wr	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	1	1	-	3/4	3/8	-	1	1	-	0	0	-	0	0	-	0	0
0x00c2	l1d_cache_refill_prefetch	1	-	-	1	-	-	0	-	-	0	-	-	0	-	-	1	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
0x001f	l1d_cache_allocate	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0x0015	l1d_cache_wb	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0
0x0047	l1d_cache_wb_clean	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0
0x0046	l1d_cache_wb_victim	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	1	1	-	1	1	-	1	1	-	0	0	-	0	0	-	0	0
0x0016	l2d_cache	2	2	2	2	2	2	2	0	⋈	1	0	0	0	0	0	2	2	2	2	2	2	1	2	2	1	1	1	1	1	1	1	1	1
0x0050	l2d_cache_rd	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0
0x0051	l2d_cache_wr	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x0017	l2d_cache_refill	1	3/8	1/2	1	1/2	1/2	1	0	0	0	0	0	0	0	0	1	0	0	1	⋈	⋈	1	0	0	0	0	0	0	0	0	0	0	0
0x4009	l2d_cache_lmiss_rd	-	-	1/2	-	-	1/2	-	-	0	-	-	0	-	-	0	-	-	0	-	-	⋈	-	-	0	-	-	0	-	-	0	-	-	0
0x0052	l2d_cache_refill_rd	1	3/8	1/2	1	1/2	1/2	1	0	0	0	0	0	0	0	0	1	0	0	1	1/4	⋈	0	0	0	0	0	0	0	0	0	0	0	0
0x0020	l2d_cache_allocate	1	-	-	1	-	-	1	-	-	0	-	-	0	-	-	1	-	-	1	-	-	0	-	-	0	-	-	1	-	-	0	-	-
0x0018	l2d_cache_wb	1	1	1/8	1	1/2	0	1	⋈	0	1	1/8	⋈	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0x0057	l2d_cache_wb_clean	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0
0x0056	l2d_cache_wb_victim	-	1	1/8	-	1/2	0	-	⋈	0	-	1/8	⋈	-	0	0	-	1	1	-	1	1	-	1	1	-	0	0	-	0	0	-	0	0
0x002b	l3d_cache	2	2	9/8	2	3/2	1	2	1/4	1/8	2	1/4	⋈	0	1/8	1/8	2	2	2	2	2	2	2	1	1	1	0	0	1	0	0	1	0	0
0x00a0	l3d_cache_rd	1	1	1	1	1	1	1	⋈	⋈	1	⋈	⋈	0	⋈	1/8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0x0036	ll_cache_rd	1	1	1	1	1	1	1	⋈	⋈	1	1/8	⋈	0	1/8	1/8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0x002a	l3d_cache_refill	3/4	1	1	7/8	1	1	1	1/8	⋈	1	1/8	⋈	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0x0037	ll_cache_miss_rd	3/4	1	1	7/8	1	1	1	1/8	⋈	1	1/8	⋈	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0x400b	l3d_cache_lmiss_rd	-	-	1	-	-	1	-	-	⋈	-	-	⋈	-	-	0	-	-	1	-	-	1	-	-	1	-	-	0	-	-	0	-	-	0
0x00a2	l3d_cache_refill_rd	7/8	-	-	7/8	-	-	1	-	-	1	-	-	0	-	-	1	-	-	1	-	-	1	-	-	0	-	-	0	-	-	0	-	-
0x00c0	l3d_cache_refill_prefetch	1	-	-	5/8	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
0x0029	l3d_cache_allocate	1	1	1/8	1	1/2	0	1	⋈	0	1	1/8	⋈	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0
0x002c	l3d_cache_wb	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0x0013	mem_access	1	1	1	2	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	1	1
0x0066	mem_access_rd	1	1	1	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0067	mem_access_wr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	1	1
0x0019	bus_access	8	8	9/2	8	47/8	33/8	8	3/4	3/8	8	1	3/4	0	0	0	8	8	8	8	8	8	8	8	8	4	4	4	4	4	4	4	4	4
0x0060	bus_access_rd	4	4	4	4	4	4	4	1/2	1/4	4	1/2	3/8	0	0	0	4	4	4	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0
0x0061	bus_access_wr	4	4	1/2	4	15/8	⋈	4	3/8	1/8	4	1/2	3/8	0	0	0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

TABLE I: Per-core PMC evaluation results. Each counter's measured memory transactions are reported as fractions of 64 MiB with an 8 MiB granularity. Counters with values deviating by more than ±15% from the expected fraction are marked as '⋈' (unreliable). Counters unavailable on a specific core are marked as '-'.

event_id	event_name	read			read_ldnp			prefetch_l1			prefetch_l2			prefetch_l3			modify			modify_prefetch			modify_stnp			write			write_stnp			write_dczva		
		A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78	A55	A76	A78
0x002b	l3d_cache	2	2	5/4	2	3/2	⋈	2	⋈	0	2	⋈	0	0	0	0	2	2	1	2	2	5/8	2	2	7/8	1	1	0	1	1	0	1	1	0
0x00a0	l3d_cache_rd	1	1	1/4	1	1	⋈	1	⋈	0	1	⋈	0	0	0	0	1	1	1/2	1	1	3/8	1	1	1/2	0	0	0	0	0	0	0	0	0
0x00a1	l3d_cache_wr	1	1	0	1	1/2	0	1	⋈	0	1	⋈	0	0	0	0	1	1	1/2	1	1	3/8	1	1	1/2	1	1	0	1	1	0	1	1	0
0x002a	l3d_cache_refill	7/8	1	1	7/8	1	⋈	1	⋈	0	1	⋈	0	0	0	0	0	1	1/2	0	3/4	⋈	0	1	1/2	0	0	0	0	0	0	0	0	0
0x00a2	l3d_cache_refill_rd	7/8	1	1/4	7/8	1	⋈	1	⋈	0	1	⋈	0	0	0	0	0	1	1/2	0	3/4	3/8	0	1	1/2	0	0	0	0	0	0	0	0	0
0x00a3	l3d_cache_refill_wr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0029	l3d_cache_allocate	1	1	1/4	1	1/2	0	1	⋈	0	1	⋈	0	0	0	0	1	1	1/2	1	1	⋈	1	1	1/2	1	1	0	1	1	0	1	1	0
0x002c	l3d_cache_wb	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1/2	1	1	3/8	1	1	1/2	1	1	0	1	1	0	1	1	0
0x0019	bus_access	4	4	39/8	4	4	3/4	33/8	⋈	0	33/8	3/8	0	4	3/8	0	8	63/8	4	8	63/8	19/8	8	63/8	29/8	4	31/8	0	4	31/8	0	31/8	31/8	0
0x0064	bus_access_normal	4	4	9/8	4	4	3/4	33/8	1/4	0	4	3/8	0	4	3/8	0	65/8	8	35/8	65/8	8	11/4	8	8	31/8	31/8	31/8	0	4	4	0	4	31/8	0
0x0060	bus_access_rd	4	4	7/8	4	4	3/4	4	1/4	0	4	3/8	0	4	3/8	0	33/8	4	17/8	33/8	4	11/8	4	4	2	0	0	0	0	0	0	0	0	0
0x0061	bus_access_wr	0	0	1/4	0	0	0	0	0	0	0	0	0	0	0	0	4	4	17/8	4	4	11/8	4	4	2	31/8	31/8	0	4	4	0	4	31/8	0

TABLE II: DSU PMC evaluation results. Each counter's measured memory transactions are reported as fractions of 128 MiB with a 16 MiB granularity. Benchmarks are executed on homogeneous core pairs. Unreliable counters (deviation of ±15%) are marked as '⋈', while unavailable counters are marked as '-'.

single DSU cluster to prevent interference from the Nvidia SCF cache shared between CPU clusters and GPU. Since the hardware limits capturing six counters per run, a complete experiment covering all counters requires multiple executions.

Table I shows the results for per-core counters. To improve readability, we convert counter values into accessed megabytes and compare them against the expected benchmark-defined access size (64 MiB). The results are categorized as follows:

- *Reliable*: Counter that measure within ±15% a fraction of 64 MiB with a granularity of 8 MiB. (Reported as $N/8$.)
- *Unreliable*: Counter whose deviation exceeds 15%. Indicated with "⋈". It does not provide a reliable estimate for bandwidth regulation.
- *Unavailable*: Counter that is unavailable on a specific core. (Reported as "-".)

The 15% threshold was selected empirically: lower values caused unstable classifications due to counter fluctuation, while results stabilized beyond this point. We still consider many prefetching counters as unreliable, especially on Cortex-A76 and A78, where the access size inferred from performance counters does not match the actual memory footprint. The same issue was also observed and discussed in [34] (Obs. 10). In contrast, Cortex-A55 cores tend to report software prefetch activity more accurately.

The table reflects the flow of memory operations through the cache hierarchy. Counters are grouped by cache level (L1, L2, L3, and system bus), with operations within each level ordered based on their functional role:

- *Access Counters*: Track generic accesses to the cache level without necessarily changing it.
- *Refill Counters*: Measure cacheline refills from lower levels of the hierarchy.
- *Allocate Counters*: Track cache allocations, marking data movement into a specific cache level from higher levels.
- *Write-Back Counters*: Measure cachelines evicted from the cache and written back to lower levels.

We also evaluated the correlation of DSU counters—which aggregate memory transactions across all cores. Since DSU counters do not distinguish per-core activity, benchmarks are executed on homogeneous core pairs (*e.g.*, two A55) accessing distinct 64 MiB memory regions (128 MiB per test). Table II presents the results for DSU counters, where the reference access size is 128 MiB, and the granularity fraction is 16 MiB.

The results of the PMC analysis help identify which counters reliably estimate memory transactions. Specifically, we select candidates for memory bandwidth regulation based on two rules that match the behavior of prior Arm core generations (Sec. II): (i) all refills from memory are accounted as reads, and (ii) write-backs of *dirty* cachelines to memory are accounted as writes. However, we have to compensate for the effects that not all cache refills are accounted correctly, and that we cannot observe if cachelines are dirty when they are written back to the L3 cache, as the `l3d_cache_wb` counter is not implemented on the analyzed Arm cores.

V. Regulation Strategies

We establish memory bandwidth regulation models for DSU-based platforms featuring Cortex-A55, A76, and A78AE cores (as well as A76AE and A78, see Sec. IV).

Effective memory bandwidth regulation requires accurately assessing the impact of all memory operations: *read*, *write*, and *modify*. However, no single per-core counter can capture this information comprehensively and important counters that measure write-backs from the L3 cache are not implemented on the analyzed Arm cores. Consequently, *MemGuard*-based mechanisms, which regulate using single metrics, must either adopt conservative estimates or perform profiling-based statistical corrections [9] to prevent cores from exceeding their allocated bandwidth. Instead, although *MemPol*-based regulations can simultaneously combine multiple metrics, selecting the optimal set of counters is needed to minimize estimation errors. Additionally, under *MemPol*, using a large number of PMCs impacts the polling period, affecting the selection of the regulation strategy.

We first define a pessimistic model using a single counter, ensuring robustness and compatibility with both *MemGuard* and *MemPol*. Then, we refine the model using additional counters for improved accuracy in *MemPol*-based regulation.

Cortex-A55: On A55, a pessimistic model assumes that every allocated cacheline interacts two times with memory, which is true in the case of modify operations, but doubles the estimated bandwidth usage for read and write operations:

$$B_{A55}^{\text{pessimistic}} = 2 \times \text{l3d\_cache\_allocate} \tag{1}$$

This model ensures worst-case isolation but overestimates read and write traffic by 100%. Alternatively, we can use the bus access counters that monitor the traffic between a core and the DSU. This can perform well in scenarios where the data expelled from the per-core caches to the DSU are not reused and not loaded back to the per-core caches:

$$B_{A55}^{\text{moderate\_1}} = \frac{1}{4} \times \text{bus\_access} \tag{2}$$

This model accurately regulates modifies and writes but overestimates reads by 100%. Another option to remove the overestimation of writes and partly of reads is to consider the L3 cache refills, which better reflect the actual memory transactions for read operations (see Table I):

$$B_{A55}^{\text{moderate\_2}} = \text{l3d\_cache\_allocate} + \text{l3d\_cache\_refill} \tag{3}$$

This model accurately regulates modifies and writes but overestimates reads by 50%.

Cortex-A76: On this platform, no per-core counter captures the write operations, but based on the observation in Table I, we *experimentally verified* that `l2d_cache_wr` counter counts the number of cachelines instead of cache data accesses. Therefore:

$$B_{A76}^{\text{pessimistic}} = 2 \times \text{l2d\_cache\_wr} \tag{4}$$

This model accurately regulates modifies but overestimates reads and writes by 100%. We can refine the memory budget estimation of the pessimistic model by combining the `l2d_cache_wr` counter, which can detect any reads, writes, and modifies for a cacheline by once, with the `l3d_cache_refill` counter, which counts only reads and modifies for a cacheline by once:

$$B_{A76}^{\text{moderate\_1}} = \text{l2d\_cache\_wr} + \text{l3d\_cache\_refill} \tag{5}$$

This model accurately regulates modifies and writes but overestimates reads by 100%. However, our results show that the `l3d_cache_allocate` counter captures 1/2 LDNP cachelines, while the `l3d_cache_refill` counter detects every LDNP cacheline. Therefore, a model using the `l3d_cache_allocate` counter corrects LDNP deviations:

$$B_{A76}^{\text{moderate\_2}} = \text{l2d\_cache\_wr} + \text{l3d\_cache\_allocate} \tag{6}$$

This model accurately regulates modifies and writes but overestimates reads by less than 100%.

Cortex-A78AE: The regulation strategy aligns closely with A76. For the pessimistic model, we use a similar strategy:[3]

$$B_{A78}^{\text{pessimistic}} = 2 \times \text{l2d\_cache\_wr} \tag{7}$$

Just as in A76, this model accurately regulates modifies but overestimates reads and writes by 100%. To improve on the previous model, we can combine `l2d_cache_wr` with `l3d_cache_refill`, effectively calculating reads and modifies twice but writes only once:

$$B_{A78}^{\text{moderate\_1}} = \text{l2d\_cache\_wr} + \text{l3d\_cache\_refill} \tag{8}$$

This model accurately regulates modifies and writes but overestimates reads by 100%. We can also combine `bus_access_wr` with `l3d_cache_refill`, the counters accounting for writes and reads respectively. This is slightly less pessimistic on reads than the previous model:

$$B_{A78}^{\text{moderate\_2}} = \frac{1}{4} \times \text{bus\_access\_wr} + \text{l3d\_cache\_refill} \tag{9}$$

This model accurately regulates modifies and writes but overestimate reads by a small error.

[3]For the A78 family [26], [28], counters `l2d_cache_rd` and `l2d_cache_wr` have been renamed to `cache_access_rd` and `cache_access_wr`. We use the former names for consistency.

VI. Implementation

Deploying *MemGuard* and *MemPol* required significant modifications to support the target platforms. Full implementation details can be found in [35].

MemGuard *Implementation:* On the Orin, we extended the implementation of *MemGuard* provided by the Minerva fork of *Jailhouse* [36]. Specifically, we re-implemented the regulator to accommodate platform-specific hardware details, such as the GIC-600AE and the different PMUs and Timers. The work was upstreamed [36].

The primary constraint for *MemGuard* is the frequency of interrupts. A shorter regulation period results in more frequent interrupts, increasing system overhead. Similarly to [1]–[4], we evaluated the overhead of *MemGuard* on our platforms to select suitable regulation periods. We conducted an experiment where a pinned core on Linux runs a *modify* benchmark from [29]. The workload iterates over a 128 MiB working set, and we measured the slowdown as a function of the replenishment period. Each core follows the budget defined by the regulation models (1), (4), and (7).

Fig. 4 illustrates the impact on each core type while showing the effects of the timer and regulation (PMU) interrupt. The results indicate that at a regulation period of $10\mu s$, the maximum observed slowdown is 1.03 for Cortex-A55 and 1.11 on Cortex-A76 cores, while there is practically no overhead for Cortex-A78AE. Reducing the period to $5\mu s$ results in a slowdown of 1.06 on the A55, 1.24 on the A76, and 1.31 on the A78AE. The differences to Cortex-A53 [4] could be attributed to the regulator implementation and the transition from GICv2 (ZCU102) to GICv3 (RK3588, Orin), which may affect interrupt latency. The overhead for shorter periods is significantly lower compared to the ZCU102 [4]. However, our experiments showed that for periods shorter than 100 μs, *MemGuard* failed to regulate memory bandwidth effectively, while at 1 ms, regulation remained functional. The cause remains unclear, but one hypothesis is that PMU interrupts are triggered but not handled correctly in our test cases. A thorough investigation of this behavior is left as future work.

MemPol *Implementation:* Unlike previous platforms, we did not use the small cores on the RK3588 and the Orin for *MemPol*, as the documentation for the integrated small cores is not publicly available on the RK3588,[4] and the small cores in the Orin SoC are all in use or not user-programmable with the default firmware.[5] Therefore, we ported the *MemPol* regulator [37] to run as a bare-metal VM on top of *Jailhouse* on the A55 resp. A78AE. The current implementation, available at [35], dedicates an entire application-level core to the regulator, but future implementations could enable *MemPol* to run on the small cores of the systems. Note that, running *MemPol* on an application core instead of a small core introduces the risk of interference in I/O-intensive scenarios

[4]The RK3588 data sheets mention three Cortex-M0 cores.

[5]The Orin has dedicated Cortex-R5 cores for power management, camera management, display management, and sensor processing; Cortex-R52 cores in an automotive safety island; and Cortex-A9 cores for audio processing.

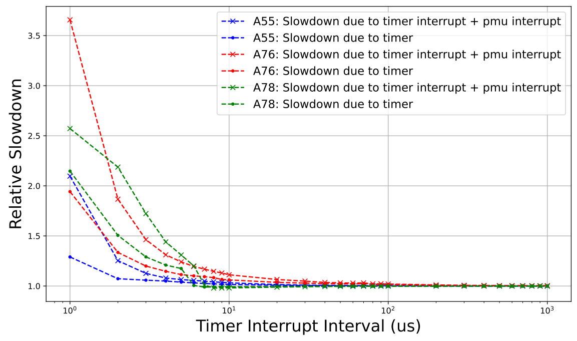


Fig. 4: Overhead of MemGuard on A55 and A76 cores on RK3588, on A78 cores on Orin. The slowdown is calculated on a modify workload running in a tight loop over 128 MiB.

where all the remaining application cores concurrently perform I/O operations, potentially delaying the regulator's execution. However, in all our experiments, I/O activity on the application cores was minimal, ensuring that the observed behavior remains representative of a setup where *MemPol* is offloaded to a small core.

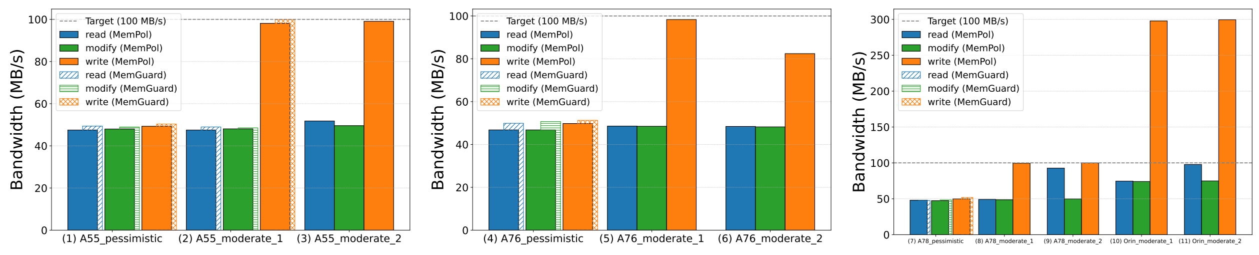
For *MemPol*, the primary tuning parameter is the polling period. To define an appropriate value, we stress the systems, measure the time required for regulation actions, and include a safety margin to ensure timely enforcement. According to the *MemPol* model [3], [4], in each polling cycle, the regulator reads PMCs from each core, processes the regulation logic, and issues two CoreSight CTI transactions per core to enforce execution throttling when necessary. Since these operations involve accessing *CoreSight* registers, their timing depends on platform characteristics such as bus speed and core frequency.

On the RK3588 platform, memory bandwidth regulation applies to eight cores. Since the regulator runs from the private caches of the core, we assigned the regulating core an infinite bandwidth. Given that the regulation models use two PMCs, each polling period involves two read accesses for PMC counters and two write accesses to the CTI per core. The measured overhead for each *CoreSight* access is $0.383\mu s$ for reads and $0.285\mu s$ for writes, leading to a polling period of $8.016\mu s$. We use $10\mu s$.

On the Orin, regulation extends to 12 cores, while the number of accesses per core remains unchanged. The measured overhead for each *CoreSight* access is $0.697\mu s$ for reads and $0.392\mu s$ for writes. We use a $50\mu s$ polling period.

Due to the differences in sustainable read and write bandwidth on the Orin, we can refine the models of the A78. We observed in Sec. III that the Orin can sustain a $3\times$ higher write bandwidth than the read bandwidth. As $B_{A78}^{\text{moderate\_1}}$ and $B_{A78}^{\text{moderate\_2}}$ comprise explicit read and write counters, we can apply a factor of $1/3$ to the write counter to account for the relaxed write bandwidth, similar to the α_r and α_w factors in the original *MemPol* [4]. We therefore improve $B_{A78}^{\text{moderate\_1}}$ to:

$$B_{Orin}^{\text{moderate\_1}} = \frac{1}{3} \times \text{l2d\_cache\_wr} + \text{l3d\_cache\_refill} \quad (10)$$



(a) A55 (b) A76 (c) A78AE

Fig. 5: Bandwidth under regulation by both *MemGuard* and *MemPol* for synthetic benchmark running *read*, *write*, and *modify* for different regulation models on A55, A76, and A78AE. Regulations (1), (2), (4), and (7) use a single PMC and are evaluated with *MemGuard* and *MemPol*. Regulations (3), (5), (6), (8), (9), (10), and (11) use two PMCs and are evaluated with *MemPol* only. The core is regulated at a target bandwidth of 100 MB/s. Note: A78AE has 3× sustainable bandwidth for *write* by using the Orin-specific models (10) and (11) which account for the higher+ sustainable write bandwidth.

And we apply a similar change to $B_{A78}^{\text{moderate\_2}}$ as well:

$$B_{Orin}^{\text{moderate\_2}} = \frac{1}{12} \times \text{bus\_access\_wr} + \text{l3d\_cache\_refill} \quad (11)$$

Both models have the same behavior for reads as the related A78 models, but penalize writes less.

For *MemPol* on the Orin, we deliberately use the older Jetson Linux version 35.3.1 firmware, as this firmware version has the memory-mapped access *CoreSight* registers enabled by default. Earlier and later firmware versions disable access to *CoreSight* registers for security reasons, see *e.g.*, [38].

Lastly, we use the Arm `MDCR_EL2` register to partition the six available PMCs on each core between OS and hypervisor. Specifically, we assigned four PMCs to Linux and two regulation-PMCs to the hypervisor. This allows Linux to use a safe subset of the PMCs for *e.g.*, `perf` without interfering with the memory bandwidth regulation.

VII. Experimental Evaluation

We evaluate the regulation models proposed in Sec. V and the effectiveness of the *MemGuard* and *MemPol* implementations discussed in Sec. VI, focusing on their ability to regulate memory bandwidth and provide performance isolation.

A. Single Core Regulation

We first verify the correct behavior of the different regulation models (Sec. V), evaluating that cores do not exceed their allocated budget. The tested core executes the benchmark tool (Sec. III), while all other cores remain idle. The benchmark tool runs read, write, and modify workloads in a tight loop over an access size of 128 MiB. We assign a bandwidth of 100 MB/s well below the maximum sustainable bandwidths (Sec. III) to the tested core, and we repeat the experiment on each type of core (Cortex-A55, A76, and A78AE) with each applicable regulation model on *MemGuard* and *MemPol*.

The results in Fig. 5 demonstrate that the allocated budget is never exceeded, confirming that both *MemGuard* and *MemPol* successfully enforce bandwidth constraints. We evaluated models using a single PMC with both *MemPol* and *MemGuard*. (Fig. 5 shows the results of *MemGuard* that are similar to those of *MemPol*.) Models using two PMCs were evaluated with *MemPol* only. As expected, the models yield different conservative levels of bandwidth enforcement. When using the pessimistic models (1), (4), and (7), the actual memory bandwidth usage remains well below the allocated budget, as only modifies are accounted precisely,[6] and reads and writes are overcounted. Conversely, for the moderate models (2), (3), (5), (6), (8) resp. (10), and (9) resp. (11)—of which only the first is available to *MemGuard*—estimated and actual bandwidths are more closely aligned. These models effectively only overcount reads. Figs. 5a and 5b show that the moderate models produce similar results on A55 and A76. However, Fig. 5c shows improvements for reads on A78AE. Fig. 5c also shows how the benefit of the Orin-specific models (10) and (11) over the standard models (8) and (9), which cannot leverage the higher sustainable write bandwidth on the Orin.

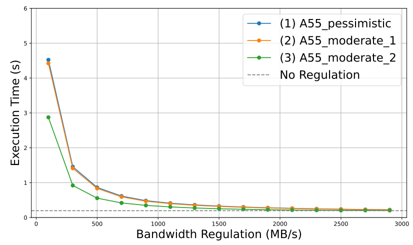
B. Application Sensitivity to Memory Bandwidth Regulation

To assess the impact of memory bandwidth regulation on real-world applications, we observe the execution time of *disparity*, *tracking*, and *mser* benchmarks from the *San Diego Vision Benchmark Suite* (*SD-VBS*) [39], [40]. We run the benchmarks on images in VGA resolution on a single core and under bandwidth regulation with a variable bandwidth from 100 MB/s to 3000 MB/s in increments of 200 MB/s using all regulation models on the three core types.

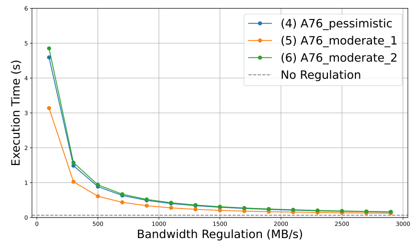
Due to space constraints, Fig. 6 only shows the results for one exemplary benchmark, *mser/vga*.[7] The shorter the execution time, the less pessimistic is the model. For all three cores, we can observe that bandwidths below 500 MB/s lead to high slowdowns. Bandwidth levels below 1000 MB/s (for A55) and 2000 MB/s (for A76 and A78AE) lead to slowdowns of at least factor two. A system integrator must therefore carefully select bandwidth levels and decide the trade-offs of slowdown

[6]Recall from Sec. III that modify operations comprise both reads and writes. The shown bandwidth of 50 MB/s for modify comprises effectively 50 MB/s reads and 50 MB/s writes, maxing out the target bandwidth of 100 MB/s.

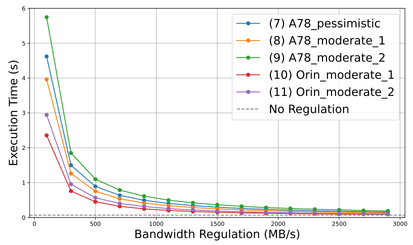
[7]All results will be published as technical report.



(a) A55

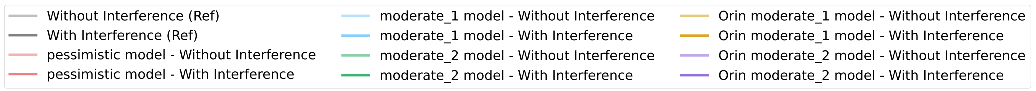


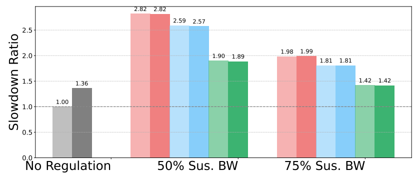
(b) A76



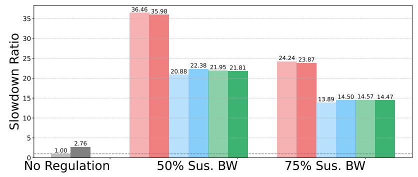
(c) A78AE

Fig. 6: Execution time (s) when varying the regulation bandwidth under different regulation models on A55, A76, A78AE running *mser/vga* from the *SD-VBS*. The dashed horizontal line shows the execution time without regulation.

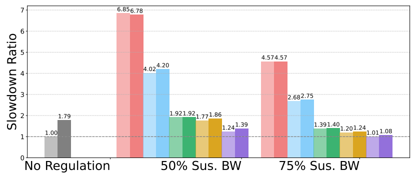




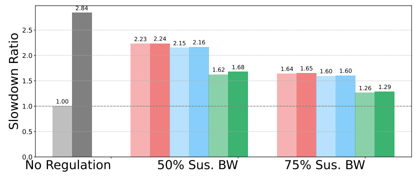
(a) A55 *disparity/vga*



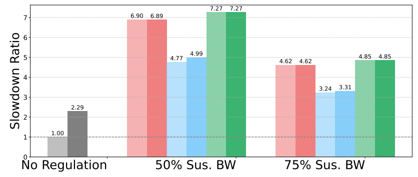
(b) A76 *disparity/vga*



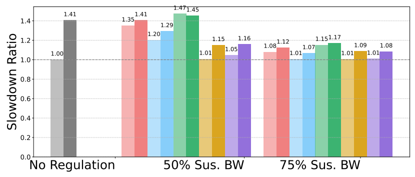
(c) A78AE *disparity/vga*



(d) A55 *mser/vga*



(e) A76 *mser/vga*



(f) A78AE *mser/vga*

Fig. 7: Execution time slowdown *w.r.t.* no-regulation/no-interference under different regulation models for *SD-VBS* targets. The target core is regulated at b percent of the sustainable bandwidth, the other cores split evenly $100\% - b$ remaining bandwidth.

vs. isolation. On all cores, the $B^{\text{moderate\_1}}$ models (2), (5), (8), and (10) yield the fastest execution.[8]

C. Isolation Performance

Memory bandwidth regulation should protect target applications from memory interference caused by concurrently executing workloads. We evaluate the isolation effectiveness of the different regulation models using benchmarks from *SD-VBS* [39], [40] as target, while the remaining cores generate *modify* memory interference from the benchmarking tool [29]. The interfering workload continuously accesses an 8 MiB working set, exerting sustained pressure on the memory system. We verified that the 8 MiB working set size generates sufficient interference. In the test scenarios, the sustainable memory bandwidth is distributed among all cores. The target core receives a fixed percentage b of the bandwidth, while the remaining bandwidth $100\% - b$ is evenly distributed among the interfering cores.

[8]An issue in our experimental setup was causing a glitch for 100 MB/s regulation on A78AE, resulting in an unexpectedly faster execution time. The current version of the setup—also committed on GitHub—reflects our new setup that corrects the issue.

Fig. 7 shows examples of the isolation capability of *MemGuard* and *MemPol* under the different regulation models for *disparity* and *mser* target workloads. The figure reports the execution time slowdown ratio *w.r.t.* the no-regulation, no-interference case. Like before, the (single counter) pessimistic models and $B^{\text{moderate\_1}}_{A55}$ are evaluated under *MemGuard*, while other moderate ones use *MemPol*. In all insets, the difference between the cases with and without interference is very limited under regulation, while is very pronounced when the system is not regulated. This is expected and indicates that regulation (independently from the model) is effective in guaranteeing the isolation of the target workloads.

Instead, slowdowns vary considerably depending on the core and benchmark type. Here, *disparity* is memory intensive, and its slowdown heavily depends on the assigned target bandwidth and on the model. Insets 7b and 7c show that the pessimistic models cause higher slowdown, while moderate ones impact A76 more than A78AE. Compared to the A76, the $B^{\text{moderate\_2}}$ models work better on A78AE, and the optimization for the

Orin reduces the slowdowns (inset 7c). The in-order A55 core closely tracks the regulation expectations, and slowdown gradually declines as models become less pessimistic.

Being less memory intensive, *mser* presents different results: moderate regulation models have comparable (sometimes even worse) performance than pessimistic ones (insets 7e, 7f), and differences between levels of assigned target bandwidth are (as expected) less pronounced. The Cortex-A55 shows instead (inset 7d) similar trends as *disparity*.

VIII. Discussion and Conclusion

In this study, we aim to bridge the gap between existing memory bandwidth regulation techniques and the latest advancements in Arm-based MPSoCs, ultimately providing insights into the feasibility of software-enforced memory bandwidth control in next-generation real-time embedded systems.

The assessment of the sustainable bandwidth shows that memory bandwidth regulation is still a reasonable building block for predictable real-time systems, even when using Arm DSU-based systems, but its applicability depends highly on the workload *and* the platform. Compared to prior platforms [4], the difference between peak bandwidth and lowest bandwidth is exacerbated on DSU-based platforms. Especially the read performance remains critical, and cores like Cortex-A76 and A78 are hungry for bandwidth. Writes are less problematic, as the out-of-order memory architecture can hide latency, even for in-order cores such as the Cortex-A55.

The newer Arm cores and the DSU introduced a shared L3 cache, and with that, many new PMCs. We analyzed the PMCs for their applicability in memory bandwidth regulation. Unfortunately, the architecture does not provide 100% suitable PMCs. We derived different *practical* models based on the available PMCs that approximate the memory bandwidth behavior. For *MemGuard*, we presented models using a single PMC, and for *MemPol*, models using two PMCs. The models provide usable memory bandwidth regulation for real-world deployment, balancing accuracy and feasibility, and apply to the Cortex-A55, A76, and A78 core families in general. However, as the cores in DSU-based system have much larger private caches and can thus handle a larger working sets without interference from other cores, soft real-time applications are less likely to require memory bandwidth regulation.

Our sensitivity analysis of *SD-VBS* with different bandwidth levels (Sec. VII) shows that setting low budget causes high slowdowns. Using the observed *minimum* bandwidth as a base is particularly problematic as it starves the cores. Therefore, finding a *reasonable* sustainable bandwidth higher than the minimum one is of paramount importance for practical regulation strategies. However, this process is complex and requires rigorous testing and knowledge of *all* applications running on the cores to exclude the simultaneous occurrence of worst-case memory access behavior that leads to interference. This observation follows the general trend in timing analysis that additional levels of caches increase the discrepancy between the average observed and worst-case execution time (WCET). System integrators must carefully assign budgets and balance isolation and predictability vs. performance.

Acknowledgments

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, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *19th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2013, Philadelphia, PA, USA, April 9-11, 2013*. IEEE Computer Society, 2013, pp. 55–64. [Online]. Available: https://doi.org/10.1109/RTAS.2013.6531079

[2] ——, "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

[3] 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

[4] ——, "Mempol: polling-based microsecond-scale per-core memory bandwidth regulation," *Real Time Syst.*, vol. 60, no. 3, pp. 369–412, 2024. [Online]. Available: https://doi.org/10.1007/s11241-024-09422-8

[5] Arm, "Arm DynamIQ Shared Unit-AE Technical Reference Manual," https://developer.arm.com/documentation/101322/ Accessed: 2025-04-05.

[6] Rockchip, "Rockchip RK3588," https://www.rock-chips.com/a/en/products/RK35_Series/2022/0926/1660.html Accessed: 2025-04-05.

[7] NVIDIA, "NVIDIA Jetson AGX Orin," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ Accessed: 2025-04-05.

[8] Arm, "Quality of Service in ARM Systems: An Overview," https://community.arm.com/arm-community-blogs/b/soc-design-and-simulation-blog/posts/quality-of-service-in-arm-systems-an-overview Accessed: 2025-04-05.

[9] 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

[10] 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)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 3:1–3:26. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13934

[11] 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

[12] 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://doi.org/10.1002/spe.3053

[13] Intel, "Resource Director Technology," https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html Accessed: 2025-04-05.

[14] Arm, "Arm Memory System Resource Partitioning and Monitoring (MPAM) System Component Specification," https://developer.arm.com/documentation/ihi0099/ Accessed: 2025-04-05.

[15] 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 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 127–139. [Online]. Available: https://doi.org/10.1145/3534879.3534882

[16] 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

[17] 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

[18] 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 2020)*, ser. OpenAccess Series in Informatics (OASIcs), M. Bertogna and F. Terraneo, Eds., vol. 77. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 3:1–3:14. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11779

[19] 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

[20] 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

[21] I. Izhbirdeev, D. Hoornaert, W. Chen, A. Zuepke, Y. Hammad, M. Caccamo, and R. Mancuso, "Coherence-aided memory bandwidth regulation," in *IEEE Real-Time Systems Symposium, RTSS 2024, York, UK, December 10-13, 2024*. IEEE, 2024, pp. 322–335. [Online]. Available: https://doi.org/10.1109/RTSS62706.2024.00035

[22] Arm, "Arm DynamIQ Shared Unit Technical Reference Manual," https://developer.arm.com/documentation/100453/ Accessed: 2025-04-05.

[23] ——, "Arm big.LITTLE Technology," https://www.arm.com/technologies/big-little Accessed: 2025-04-05.

[24] ——, "Arm Cortex-A55 Core Technical Reference Manual," https://developer.arm.com/docs/100442/ Accessed: 2025-04-05.

[25] ——, "Arm Cortex-A76 Core Technical Reference Manual," https://developer.arm.com/docs/100798/ Accessed: 2025-04-05.

[26] ——, "Arm Cortex-A78 Core Technical Reference Manual," https://developer.arm.com/docs/101430/ Accessed: 2025-04-05.

[27] ——, "Arm Cortex-A76AE Core Technical Reference Manual," https://developer.arm.com/docs/101392/ Accessed: 2025-04-05.

[28] ——, "Arm Cortex-A78AE Core Technical Reference Manual," https://developer.arm.com/docs/101779/ Accessed: 2025-04-05.

[29] A. Zuepke, "Memory Benchmark," https://gitlab.com/azuepke/bench Accessed: 2025-04-05.

[30] 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

[31] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU Power Management for 3D Mobile Games," in *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*. ACM, 2014, pp. 40:1–40:6. [Online]. Available: https://doi.org/10.1145/2593069.2593151

[32] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith, "Evaluating attainable memory bandwidth of parallel programming models via babelstream," *Int. J. Comput. Sci. Eng.*, vol. 17, no. 3, pp. 247–262, 2018. [Online]. Available: https://doi.org/10.1504/IJCSE.2018.095847

[33] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla, "High-integrity performance monitoring units in automotive chips for reliable timing v&v," *IEEE Micro*, vol. 38, no. 1, pp. 56–65, 2018. [Online]. Available: https://doi.org/10.1109/MM.2018.112130235

[34] A. Pradhan, D. Ottaviano, Y. Jiang, H. Huang, A. Zuepke, A. Bastoni, and M. Caccamo, "Arm DynamIQ Shared Unit and Real-Time: An Empirical Evaluation," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 269–282. [Online]. Available: https://doi.org/10.1109/RTAS65571.2025.00032

[35] Chair of Cyber-Physical Systems in Production Engineering, "RTCSA2025 MemGuard Mempol," https://github.com/rtsl-cps-tum/rtcsa2025-memguard-mempol Accessed: 2025-06-05.

[36] Minerva Systems, "Memory-aware Jailhouse hypervisor," https://github.com/Minervasys/jailhouse Accessed: 2025-04-05.

[37] A. Zuepke, "MemPol Implementation," https://gitlab.com/azuepke/mempol Accessed: 2025-04-05.

[38] Z. Ning, C. Wang, Y. Chen, F. Zhang, and J. Cao, "Revisiting ARM debugging features: Nailgun and its defense," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 574–589, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2021.3139840

[39] 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

[40] 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