Work-in-Progress: A First Practical Look at Arm's MPAM for Real-Time Systems

Ashutosh Pradhan, Daniele Ottaviano, Alexander Zuepke, Andrea Bastoni, Marco Caccamo
Technical University of Munich
{ashutosh.pradhan, daniele.ottaviano, alex.zuepke, andrea.bastoni, mcaccamo}@tum.de

***Abstract*—Arm's Memory Partitioning and Monitoring (MPAM) extension introduces standardized mechanisms for partitioning cache and memory bandwidth. From a real-time systems perspective, this can aid in improving predictability in heterogeneous MPSoCs. In this paper, we present the first practical evaluation of MPAM on a COTS platform—the Radxa Orion O6 with the CIX CD8180 SoC. We characterize the SoC's MPAM capabilities and experimentally assess cache portion partitioning and proportional stride memory bandwidth partitioning under controlled interference. Our results show that enabling MPAM features can reduce interference, but their behavior often diverges from expectations based on the specification, with anomalous effects observed across workloads and cores. These findings highlight both the promise of predictability from MPAM for real-time systems and the current challenges arising from optionality, heterogeneity, and limited documentation. We conclude that broader evaluation across future MPAM-enabled SoCs, aided by detailed performance counter analysis, is essential to establish MPAM's practical value for real-time practitioners.**

***Index Terms*—real-time system, multi-core, cache partitioning, memory bandwidth partitioning, Arm, MPAM**

I. Introduction and Motivation

Under real-time constraints, applications executing on heterogeneous MPSoCs suffer from a lack of predictability due to contention on shared physical resources. Primary sources of interference stem from simultaneous access to the cache and memory subsystems from multiple cores. Several hardware- and software-based approaches to mitigate these issues have been proposed in the past [1]. Software-based techniques typically rely on performance counters to track memory activity and throttle cores once a task exceeds its budget. This approach is inherently indirect, incurs overhead from start–stop control in software, and is constrained by the availability of suitable counters [2]. On the other hand, hardware-based proposals often require custom designs, making them impractical for COTS platforms where the hardware is not reconfigurable [3].

To mitigate interference, chip manufacturers have introduced hardware partitioning mechanisms, which can reduce the complexity faced when trying to build predictable systems. For example, Intel RDT [4] implements *Cache Allocation Technology* to partition the LLC per core and *Memory Bandwidth Allocation* to limit the memory bandwidth per core. However, prior analyses [5] have shown that these mechanisms do not always deliver consistent isolation in practice. Moreover, support for RDT remains limited to the Xeon family of processors [6], primarily aiming at Quality-of-Service (QoS) levels for server workloads, rather than embedded or mixed-criticality domains. In contrast, Arm introduced MPAM [7], explicitly designed to provide cache and memory bandwidth partitioning in a standardized manner. MPAM is increasingly being integrated as a default capability in the latest Arm cache controllers and interconnects. Given Arm's dominance in mobile, automotive, and embedded markets, this trend suggests that MPAM will soon become widely available on COTS heterogeneous MPSoCs. For real-time practitioners, such widespread hardware support is crucial as it enables predictable performance isolation directly at the hardware level, enabling more robust and certifiable real-time systems.

In an MPAM-capable Arm system, memory requests are managed by Memory-System Components (MSCs), such as caches, TLBs, SMMUs, and DRAM controllers. Each MSC can partition resource usage at the granularity of a task, using the `PARTID` to distinguish between tasks. The `PARTID` is configured via core-level MPAM system registers on Cortex-A processors and must be set correctly by the OS on each task switch. Depending on the partitioning scheme, each MSC can be programmed to reserve a specific share of resources for a given `PARTID`. From a Linux user perspective, software controls for partitioning and monitoring of memory resources will be supported in the mainline kernel with the `resctrl` interface in `sysfs` (unified for both Intel RDT and Arm MPAM).

While MPAM promises a standardized way to control QoS of memory requests on Arm platforms, its current specification poses challenges for real-time use. First, many features are declared as `IMPLEMENTATION DEFINED`, leaving practitioners uncertain since the same partitioning mechanism may behave differently across SoCs. Zini *et al.* [8] discuss these ambiguities and propose clarifications for system designers. Second, many MPAM features are optional and might be implemented only by some MSCs. Even worse, supported MSCs may expose different partitioning schemes. Therefore, mixed configurations are possible where, for example, the DynamIQ Shared Unit (DSU) might implement *memory-bandwidth portion partitioning*, the interconnect may lack MPAM entirely, and the DRAM controller may support *minimum and maximum partitioning*. Such inconsistencies mandate careful analysis of the predictability and end-to-end guarantees that MPAM provides on each SoC.

In this paper, we present the first practical evaluation of Arm's MPAM partitioning features—focusing on cache and, in particular, memory bandwidth partitioning. Our study builds on the CIX CD8180 SoC hosted on the recent Radxa Orion

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

O6 platform [9], which provides MPAM support in hardware. We will describe the SoC and its MPAM capabilities, our experiments, and key findings. Finally, we discuss implications of MPAM for real-time systems and outline future directions.

II. System Description

The CD8180 is an Armv9.2 SoC that integrates Cortex-A520 cores [10], Cortex-A720 cores [11], and the latest DSU-120 [12]. As a coherent interconnect between cores, caches, and DRAM, it employs the CoreLink CI-700 [13]. The four Cortex-A520 cores serve as the high-efficiency, low-power *little* cores. Unlike their predecessors, such as the Cortex-A55, these cores support only 64-bit execution. Each Cortex-A520 core includes `64KiB` of L1 instruction and data caches, but lacks a private L2 cache. Complementing them, the SoC also features eight Cortex-A720 (four *big* and four *medium*) cores, which similarly omit 32-bit execution support. Each Cortex-A720 core is equipped with `64KiB` of L1 instruction and data caches and a private `512KiB` L2 cache. All twelve cores are part of the same DSU cluster and share a unified 12-way, `12MiB` L3 cache.

The DSU-120 implements MPAM-based mechanisms for cache and memory bandwidth partitioning. For cache partitioning, the CD8180 supports *Cache Portion Partitioning*. Although defined by the MPAM specification, this mechanism is conceptually similar to the L3 way partitioning supported by earlier DSUs on Armv8.2 SoCs [14]. It allows assignment of L3 cache ways per `PARTID` with a granularity of two ways. The SoC provides a 6-bit field, where each bit corresponds to two cache ways that may be allocated to a specific `PARTID`.

For memory bandwidth partitioning, the DSU-120 supports only the *Proportional Stride Memory Bandwidth Partitioning* method; other schemes such as *MIN*, *MAX*, or *Portion* partitioning [7] are not available.[1]According to the MPAM specification, proportional stride regulation is activated only under contention, i.e., when two or more cores simultaneously access the memory subsystem. Strides can be assigned in the range `0-63` (6 bits), where `0` corresponds to the highest quality of service and `63` to the lowest. Importantly, the scheme applies regulation only when demand exceeds available bandwidth; in the absence of contention, no throttling occurs. A detailed description of the scheme is provided in the MPAM component specification [7].

Although the CI-700 *could* support MPAM partitioning, in contrast to the DSU-120, probing the CI-700 configuration master registers on the CD8180 from the Cortex-A cores does not reveal any MPAM features. The technical brief of Radxa Orion O6 states that the board uses LPDDR5 DRAM. However, since no public TRM is available for the CD8180 SoC, the details of the DRAM controller and whether it supports MPAM remain unclear to us.

III. Experiments and Evaluation

Since the `resctrl` interface for MPAM is not yet in the mainline Linux kernel, we relied on the test drivers provided by Arm in the Radxa kernel repository for our evaluation. These drivers consist of two main components: (a) routines for identifying and configuring parameters of each MSC, and (b) a `sysfs` interface to assign a `PARTID` to each task or process `PID`. In their original form, the drivers supported only cache portion partitioning. We extended them to add functionality for configuring proportional stride values through bitfields.[2] In addition, we enabled the L3 DSU MSC in the Linux device tree. With Radxa firmware version 0.3.1-1, the modified drivers operated correctly without requiring any further firmware changes.

To evaluate the effectiveness of MPAM Cache Partitioning and Memory Bandwidth Partitioning in the DSU, we follow an approach similar to [14] and [2], respectively. We use a combination of real-world benchmarks (as evaluation target) and synthetic benchmarks (as interference) to gain insights into the behavior of the features. As the real-world benchmark, we use San Diego Vision Benchmarks (`SD-VBS`) [15] under the RT-Bench [16] wrapper. Specifically, we use the *disparity* and *mser* tasks (identified as memory-intensive in [14]), together with the large *vga* and *fullhd* image sizes to intentionally generate heavy memory traffic. To generate worst-case access patterns as interference, we use the memory benchmark tool "`bench`" [17], previously used also in [2] [14] [18].

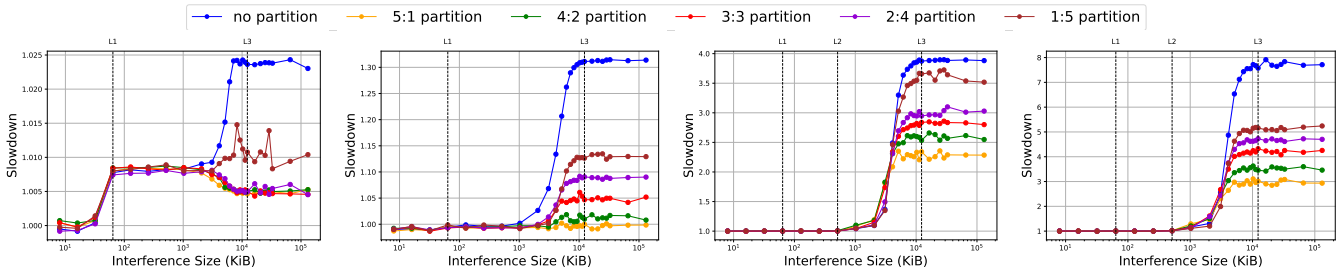
First, we evaluate the Cache Portion Partitioning feature of the DSU-120. We measure the slowdown of `SD-VBS` workloads when they are executed alongside interfering access patterns generated using `bench`. The `SD-VBS` runs on a designated "Target Core" (*`TC`*), while synthetic interference workloads run on the "Interference Cores" (*`ICs`*). The access type of `bench` is varied as *read*, *modify*, *write*, and *prefetch*. *prefetch* uses the `PRFM` instruction to prefetch cachelines into the L3 cache. *read* (*write*) repeatedly load (store) full cachelines from memory. *modify* changes only a single byte within a cacheline, which forces the core to first load the entire line before an eventual write-back.

Compared to most of the platforms studied in [14], the CD8180 features a significantly faster memory subsystem. As a result, a single *`IC`* executing `bench` is often unable to saturate the available memory bandwidth, leading to minimal or no observable slowdown on the *`TC`* running `SD-VBS`. To ensure that the memory subsystem operates under sufficient stress to reveal the effects of interference and partitioning, we employ multiple *`ICs`*. Specifically, for evaluating Cache Portion Partitioning, we consider two configurations: (1) one Cortex-A520 *`TC`* executes `SD-VBS`, while the remaining three Cortex-A520 *`ICs`* execute *modify*; (2) one Cortex-A720 *`TC`* runs `SD-VBS`, while seven Cortex-A720 *`ICs`* execute *modify*.

To sufficiently stress the system, the size of the interference is varied in steps from `8KiB` to `128MiB`. The slowdown is computed as the ratio of `SD-VBS` execution time on the *`TC`* under interference to the execution time with no interference in the same configuration. Fig. 1 shows the resulting slowdown for two memory-intensive `SD-VBS` benchmarks, *disparity*/*vga*

[1]For a concise overview of the MPAM mechanisms, see Section 4 of [8].

[2]The drivers can be found at: https://github.com/rtsl-cps-tum/kernel-radxa.



(a) Disparity A520/A520 (b) Mser A520/A520 (c) Dispartiy A720/A720 (d) Mser A720/A720

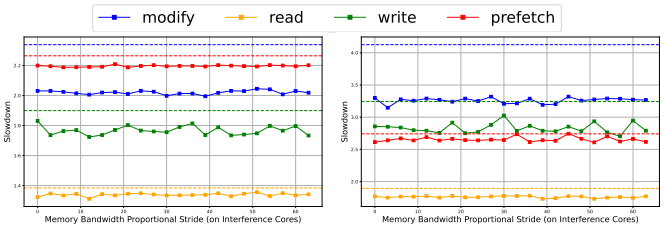
Fig. 1: Cache portion partitioning on *disparity/vga* (a, c) *mser/vga* (b, d) benchmark running on A520 and A720 cores. Each plot reports the execution slowdown w.r.t. the *no-interference* case with increasing size (KiB) of *modify* interference on three A520 cores (a, b) and seven A720 cores (c, d). L1, L2 (if applicable), and L3 sizes are marked. Note the different y-scale.

and *mser/vga*, under the effect of *modify* on the `ICs`. Since the partitioning scheme is configured through a 6-bit field, we plot the slowdown for each possible exclusive partitioning between `TC` and `ICs`. A `5:1` partition assigns 5 bits (10 cache ways) to the `TC` and 1 bit (2 cache ways) to the `IC`. Similarly, `4:2` allocates of 8 cache ways to `TC` and 4 cache ways to `ICs`, and so on.

We observe that Cache Portion Partitioning on the DSU-120 behaves consistently with per-way L3 partitioning on the DSU studied on the previous generation of Armv8.2 SoCs [14]. Partitioning significantly reduces slowdown from interference, and larger cache portions for the `TC` yield larger performance gains. The improvement is more pronounced on Cortex-A720 cores, where the slowdown for *mser* reduces from 8× to 3× between the `no partition` and `5:1` cases. On Cortex-A520 cores, we observe no significant slowdown for *disparity* (maximum slowdown of 2.5%). However, for *mser*, a `5:1` partition eliminates the 1.3× maximum slowdown observed without partitioning. Compared to [14], however, the slowdown begins much earlier due to memory controller accesses. This follows from using multiple `ICs`, which shrinks the effective cache space per core in an already partitioned L3, forcing higher main memory accesses.

Before evaluating memory partitioning on the DSU-120, we first characterized the memory subsystem by measuring its *sustainable memory bandwidth*, defined as the maximum bandwidth that the controller can maintain under worst-case access patterns [18]. To measure this, a large buffer is accessed with varying cacheline strides for *read*, *modify*, and *write*, triggering worst-case DRAM patterns. The SoC behavior closely matches the RK3588 studied in [2], though the CD8180 is faster, making its bandwidth a scaled and shifted version of the RK3588. A single Cortex-A520 core exhibits a worst-case *modify* bandwidth of ∼900MB/s around a `512KiB` step. Similarly, on a single Cortex-A720 core, we observe the worst-case memory bandwidth of ∼1400MB/s at the step-size of `8MiB`. Overall, CD8180 sustains ∼3800MB/s when all cores execute *modify* with `1MiB` stride.

With the sustainable bandwidth in place, we evaluate the *proportional stride memory bandwidth partitioning* feature of the DSU-120. This feature is controlled via a 6-bit stride field



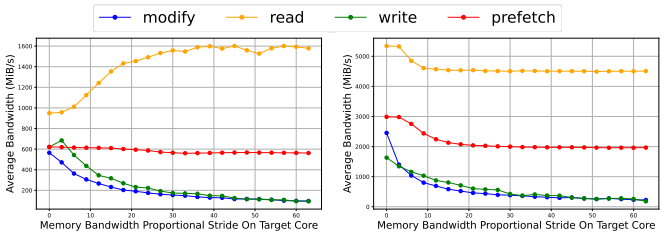
(a) `TC` A520 (b) `TC` A720

Fig. 2: Effect of increasing proportional stride on `ICs`. Interference is caused by seven A720 cores executing *read*, *modify*, *write*, and *prefetch*. Plots report slowdown of *mser/vga* compared to *no-interference*. Dotted lines show the slowdown without enabling memory bandwidth proportional partitioning.

(Sec. II), where higher stride values are intended to reduce the quality of service. We reuse the same setup as in the cache portion partitioning experiments. One Cortex-A520 `TC` executes `SD-VBS`, while seven Cortex-A720 `ICs` run *read*, *modify*, *write*, and *prefetch* in separate experiment runs, with an access size of `128MiB`. Using multiple Cortex-A720 `ICs` creates heavy pressure on the memory subsystem. To eliminate cache-level interference, we fix the L3 partitioning at `3:3`. We then vary the proportional stride of the `ICs` from 0 to 63, keeping the `TC` at stride 0. The goal is to measure the degree of isolation achieved by a real-time task on the `TC`. We repeat the same procedure with one Cortex-A720 `TC` running `SD-VBS`.

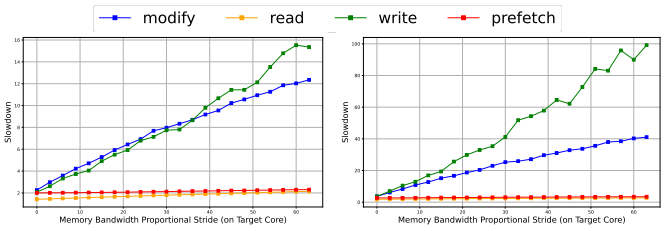
Fig. 2 presents the slowdown as the `ICs`' stride increases. The reference dotted lines indicate the baseline configuration—with `3:3` cache portion partitioning, but *proportional stride memory bandwidth partitioning* disabled (`enable` bit set to 0). We observe that simply enabling the feature substantially reduces the slowdown. However, increasing the stride beyond zero has almost no additional effect. This outcome contrasts with the specification, but may be explained by `bench` running on multiple cores saturating the DRAM bandwidth, rendering stride variations ineffective.

To analyze this further, we reversed the experiment: instead of varying on the `ICs`, we vary the proportional stride of the `TC`. To ensure the target generates heavy DRAM traffic, we execute *modify* on it, which serves as a realistic proxy for



(a) TC A520 (b) TC A720

Fig. 3: Effect of increasing proportional stride on TC. Average bandwidth of *modify* on TC under interference from seven A720 cores executing *read*, *modify*, *write*, and *prefetch*.



(a) TC A520 (b) TC A720

Fig. 4: Effect of increasing proportional stride on TC. Interference is caused by seven A720 cores executing *read*, *modify*, *write*, and *prefetch*. Plots report slowdown of *mser/fullhd* compared to *no-interference*.

memory-intensive workloads. Cache partitioning is again fixed at 3:3, and all cores use a buffer size of 128MiB. Fig. 3 shows the average DRAM bandwidth of the *TC* under interference from seven Cortex-A720 cores. Here, increasing the stride reduces the *TC*'s bandwidth in nearly all cases, with severe drops under *modify* and *write* interference. Unexpectedly, when the *TC* is a Cortex-A520 core under *read* interference, bandwidth almost doubles as the stride increases.

We further investigate this anomaly with a slightly modified setup. On *TC*, instead of *modify*, we run the *mser* benchmark with a larger input size of *fullhd*, maximizing the memory pressure. Fig. 4 shows the resulting slowdown relative to execution without interference, for both Cortex-A520 and Cortex-A720 *TCs*. In this case, *read* no longer speeds up the Cortex-A520, though the slowdown it causes remains smaller than with *modify* and *write* (as high as 16x on Cortex-A520 or even 100x on Cortex-A720 cores). Taken together, these results provide inconclusive evidence regarding the true effect of proportional strides. The unexpected speed-up could stem from the lack of MPAM support in the interconnect or artifacts of our methodology. A definitive answer would require more information from the SoC TRM or extensive reverse engineering with core and DSU performance counters, which we leave for future work.

IV. Conclusion and Future Work

In this work, we presented the first practical evaluation of Arm's MPAM cache- and bandwidth-partitioning features on the Radxa Orion O6 (CIX CD8180 SoC), highlighting their potential and current limitations for real-time systems. On the one hand, DSU-120's cache portion partitioning and proportional stride memory bandwidth partitioning can reduce interference and improve isolation across cores. On the other hand, our experiments revealed unexpected behaviors, such as negligible benefit from increasing stride values and even anomalous bandwidth increases under certain interference patterns. These results show the complexity practitioners face when relying on MPAM in the absence of detailed documentation and consistent feature support across MSCs.

Looking forward, the dependence of MPAM behavior on SoC specifics calls for access to a broader range of platforms to enable systematic validation of the specification. Limited documentation and the optional, heterogeneous nature of MPAM implementations mean that results may differ significantly across future designs. Future work will focus on analyzing these behaviors using core- and DSU-level performance counters to better understand interactions between MPAM and the memory subsystem. Ultimately, a clearer picture of MPAM's role in real-time systems will only emerge as more SoCs adopt the extension and enable comprehensive experimental studies.

References

[1] T. Lugo, S. Lozano, J. Fernández, and J. Carretero, "A Survey of Techniques for Reducing Interference in Real-Time Applications on Multicore Platforms," *IEEE Access*, vol. 10, pp. 21 853–21 882, 2022.
[2] A. Pradhan, D. Ottaviano, Y. Jiang, H. Huang, J. Zhang, A. Zuepke, A. Bastoni, and M. Caccamo, "Predictable Memory Bandwidth Regulation for DynamIQ Arm Systems," in *RTCSA*, 2025.
[3] D. Guo, M. Hassan, R. Pellizzoni, and H. Patel, "A Comparative Study of Predictable DRAM Controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 2, 2018.
[4] Intel, "Resource Director Technology," https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.
[5] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A Closer Look at Intel Resource Director Technology (RDT)," in *RTNS*, 2022.
[6] Intel, "Is Intel® Resource Director Technology Only Supported on Intel® Xeon® Processors?" https://www.intel.com/content/www/us/en/support/articles/000093676/processors/intel-xeon-processors.html.
[7] Arm, "Arm Memory System Resource Partitioning and Monitoring (MPAM) System Component Specification," https://developer.arm.com/docs/ihi0099/aa.
[8] M. Zini, D. Casini, and A. Biondi, "Analyzing Arm's MPAM From the Perspective of Time Predictability," *IEEE Trans. Comput.*, vol. 72, no. 1, pp. 168–182, 2022.
[9] Radxa, "Radxa Orion O6," https://radxa.com/products/orion/o6/.
[10] Arm, "Arm® Cortex-A520 Core Technical Reference Manual," https://developer.arm.com/docs/102517/0002.
[11] ——, "Arm® Cortex-A720 Core Technical Reference Manual," https://developer.arm.com/docs/102530/0002.
[12] ——, "Arm® DynamIQ™Shared Unit-120 Technical Reference Manual," https://developer.arm.com/docs/102547/0201.
[13] ——, "Arm®CoreLink™CI-700 Coherent Interconnect Technical Reference Manual," https://developer.arm.com/docs/101569/0300/.
[14] 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 *RTAS*, 2025.
[15] 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 *IISWC*, 2009, pp. 55–64.
[16] 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, p. 184–195.
[17] A. Zuepke, "Memory Benchmark," https://gitlab.com/azuepke/bench.
[18] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "MemPol: polling-based microsecond-scale per-core memory bandwidth regulation," *Real Time Syst.*, vol. 60, no. 3, pp. 369–412, 2024.