# Contending memory in heterogeneous SoCs: Evolution in NVIDIA Tegra embedded platforms

*(Draft/Preprint)*

Nicola Capodieci, Roberto Cavicchioli, Ignacio Sañudo Olmedo, Marco Solieri and Marko Bertogna
University of Modena and Reggio Emilia, Modena, Italy
*name.surname*@unimore.it

***Abstract*—Modern embedded platforms are known to be constrained by size, weight and power (SWaP) requirements. In such contexts, achieving the desired performance-per-watt target calls for increasing the number of processors rather than ramping up their voltage and frequency. Hence, generation after generation, modern heterogeneous System on Chips (SoC) present a higher number of cores within their CPU complexes as well as a wider variety of accelerators that leverages massively parallel compute architectures. Previous literature demonstrated that while increasing parallelism is theoretically optimal for improving on average performance, shared memory hierarchies (i.e. caches and system DRAM) act as a bottleneck by exposing the platform processors to severe contention on memory accesses, hence dramatically impacting performance and timing predictability. In this work we characterize how subsequent generations of embedded platforms from the NVIDIA Tegra family balanced the increasing parallelism of each platform's processors with the consequent higher potential on memory interference. We also present an open-source software for generating test scenarios aimed at measuring memory contention in highly heterogeneous SoCs.**

***Index Terms*—Memory Interference, GP-GPU, Compute Accelerators, Real-Time, Tegra.**

## I. Introduction

The system design paradigm of embedded platforms shifted unequivocally towards heterogeneity of their constituent processors and increasingly parallel computing capabilities. In such SoCs (System on Chips), heterogeneity is to be intended within the CPU complex and with the compute accelerators integrated in the same die. Within the CPU complex, differences might arise in the form of non-symmetrical CPU islands, i.e. CPU cores present significant architectural and performance differences as commonly occurs in ARM big.LITTLE complaint designs [1] or the more recent Big+Super NVIDIA design [2]. Heterogeneities are also observable in modern embedded platforms as a combination of diverse compute accelerators, such as graphic processing units for general purpose computing (GP-GPU), programmable logic (FPGA) and application-specific integrated circuitry (ASIC, e.g. neural network inference processors). Even though typical workloads for such accelerators present significant differences in terms of memory requirements, a common hardware design paradigm imposes that these accelerators are interconnected to the rest of the SoC through one or more levels within the memory hierarchy. For instance, caches might be shared among processing units, or more often, the system memory (usually implemented as a collection of DRAM banks) represents the interconnection layer among the different computing clusters. Previous literature highlighted [3], [4], [5], [6], [7] that these intersection points between clients and a memory hierarchy subcomponent represent a significant performance bottleneck in modern platforms. For instance, more than one CPU core sharing a common cache level causes uncontrolled eviction of useful cache lines; at system memory level, undisclosed arbitration policies in memory controllers might severely impact latencies when multiple clients are accessing memory in overlapping time windows. A platform-specific characterization of such problems is mandatory before attempting to design ad-hoc mitigation solutions for memory contention [8], [9].

In this work, we therefore investigate: whether platform vendors are aware of the memory bottleneck problem, and how their memory subsystem design evolved through subsequent generations. We provide a qualitative answer to these research questions by presenting an exhaustive set of measures with regard to memory contention in the three most recent generations of NVIDIA Tegra SoCs. This choice is motivated by the full availability for such platforms and their wide adoption in several latency-sensitive scenarios (e.g.: automotive [10], unmanned aerial vehicles [11], industrial automation [12], ...). In order to foster reproducibility of our tests, as an additional contribution we present *HeSoC-mark*, an extensible, open source test script generator and collection of benchmarks aimed at stressing shared memory accesses from the widest possible variety of compute engines in highly heterogeneous SoCs.

This paper is organized as follows: in the next section we provide an overview of previous related work regarding memory interference in modern architectures. In section III, a brief overview of the tested platforms is presented. In section IV our specifically-designed benchmark suite is described in details. In section V we characterize the tested platforms in terms of their memory bandwidths. In section VI we discuss and present the tests on memory interference with respect to memory access and interrupt response latencies. Discussion on results and related conclusive remarks are reported in sections VII and VIII.

## II. Related Work

In latency-sensitive scenarios such as Advanced Driver-Assistance Systems (ADAS), avionics and industrial automa-

tion, it is crucial to precisely estimate the execution time requirements of each task on a target computational platform [8]. This requirement stems from the need for an accurate Worst Case Execution Time (WCET) estimation that could inform the decision of job schedulers able to guarantee hard real-time requirements [13].

Through an experimental evaluation on the performance of a CPU+GPU system, Yamagiwa and Wada in [4] realized that the memory bandwidth requirements of each application is an important factor when designing a job scheduler. The authors observed that if one application intensively performs transactions over a shared memory device, the other applications' memory requests are stalled. Authors in [6] and [3] drew the same conclusion in heterogeneous embedded CPU+GPU systems. In order to have a better understanding on memory contention in multi-core systems, Tudor et al. observed in [5] that, in test scenarios involving both NUMA and UMA server-class processors, when data-hungry applications are co-running, there is a higher performance deterioration (in terms of number of cycles required to execute a program) when the number of active cores increases. They therefore propose an analytical model which manages to estimate the increase of required cycles in the presence of interference. Such model depends on the magnitude of the interference, which is estimated as the ratio between the total number of stall cycles incurred due to memory contention over the number of cycles required to execute the program with no interference. We argue that, as the complexity of the heterogeneous embedded SoCs grows in terms of sheer number of parallel processing engines, *the ability to model and predict memory interference is mandatory to derive accurate and tight WCET estimations* for the taskset of interest.

In this context, our contribution is to provide a complete characterization on memory interference on highly heterogeneous embedded systems. More specifically, our characterization will highlight how memory interference affects the recently released NVIDIA Tegra-based embedded platforms. We will measure latencies on memory accesses with variable working-set sizes and locality on memory access patterns. We will also investigate how compute accelerators are influenced by system-level memory contention as well as interrupt latency distribution in the presence of interference. We believe that a solid analytical model should start from an accurate platform characterization of the aforementioned quantities.

## III. Platforms description

In this section we describe the architectural features of the NVIDIA Tegra embedded boards we considered in our tests. We chose the three most recent NVIDIA Jetson modules that are going to be supported for at least the next five years[1].

### A. Jetson Nano

Although the NVIDIA Jetson nano is the most recently released NVIDIA Jetson development board, its SoC is characterized by fairly older designs compared to the other platforms considered in our tests. Specifically, the CPU multi-core host is an ARM A57 quad-core, which can be clocked up to 1.43 GHz. The A57 cluster features a 2 MiB L2 Last Level Cache (LLC) shared among all cores. The integrated GPU (iGPU) is a half-cut down version of the now discontinued Tegra X1 Jetson, hence featuring 128 processing elements (called CUDA[2] cores) clustered into a single Streaming Multiprocessor (SM). This accelerator can be clocked up to 921 MHz and it complies with the NVIDIA Maxwell GPU $\mu$architecture. Both the CPU host and the iGPU share system DRAM, which is a 64-bit, 4 GiB LPDDR4 capable to reach a theoretical maximum bandwidth of 25.6 GiB/s. A schematic representation of the Nano's compute engines and related memory subsystem is visible in the left inset of Fig. 1.

### B. Jetson TX2

The NVIDIA Jetson TX2 is a commercial SoC released in March 2017. Its CPU complex differs from the Jetson Nano CPU host as it is composed of two different islands: a quad-core ARMv8 Cortex-A57 and a dual-core ARMv8-compliant Denver processor. Such an unusual coupling of high performance processors has been termed as Big+Super design [2]. The ARM cluster is the same as the one in the Nano, but can reach higher frequencies (2 GHz), as the entire SoC relies on a more powerful energy dissipation system. The dual-core Denver is a NVIDIA proprietary implementation of an ARM v8.2 processor [15], characterized by aggressive Out of Order (OoO) instruction issue in which a 128 MiB optimization cache stored in main memory acts in concert with the CPU instruction cache. The dual-core Denver clock is the same as the A57 island. Each of the six cores integrates a 32 KiB L1 data cache and a 48 KiB L1 instruction cache, whereas the two islands feature a separate L2 LLC of 2 MiB. The iGPU features a Pascal based $\mu$architecture, in which 256 CUDA cores are split evenly into two SMs. Its peak clock is 1.3 GHz. GPU and CPU complexes share central memory, which is a 8GB 128-bit LPDDR4 DRAM clocked at most at 1866 MHz, hence providing a peak theoretical bandwidth of 59.7 GiB/s. A schematic representation of the TX2's compute engines and related memory subsystem is visible in the right inset of Fig. 1.

### C. Jetson Xavier

The NVIDIA Jetson Xavier platform has been released in June 2018. The CPU-side features a NVIDIA-proprietary design compliant with the ARM v8.2 architecture, codenamed Carmel, which is an evolution over the precedent generation of NVIDIA Denver CPUs. In the Jetson platform, the CPU complex is composed of eight identical cores, which can be clocked up to 2265 MHz. While aggressive OoO execution policies are still a notable Xavier feature, differences with respect to the Denver island as implemented in the TX2 are to be found in the adaptive cache prefetching ability and the CPU cache hierarchy. Denver prefetcher closely followed

[1] https://developer.nvidia.com/embedded/community/lifecycle

[2] CUDA [14] is the most commonly used CPU-GPU programming model and API in NVIDIA platforms

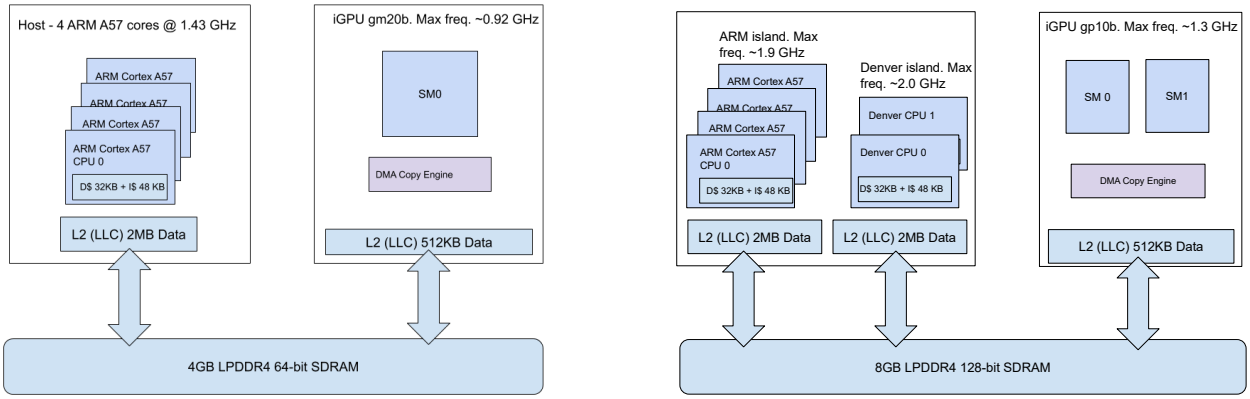


**Fig. 1:** Compute engines and related memory contention points in NVIDIA Jetson Nano (left) and TX2 (right).

the ARM implementation, in which a cache line miss would trigger a fixed amount of line requests streamed to the memory controller (2, 4 and 8 lines) [16], whereas the Xavier Technical Reference Manual (TRM)[3] hints that the Carmel prefetcher is adaptive, hence becoming more similar to the more complex prefetching policies typically adopted by x86 processors [17], [18], [19].

Regarding the cache hierarchy, the Carmel CPU complex in Xavier is composed of 4 identical islands, with each island incorporating two cores with a private L1 (64 KiB data cache) and a shared L2 cache (2 MiB). The LLC is instead shared among all islands, and it is a 4 MiB L3 victim cache. From the TRM, we understand that CPU L3 can be statically partitioned so to assign a portion of it to act as an additional cache level for the iGPU. However, we did not experimented on the L3 CPU/iGPU partitioning, as we used the default configuration that exclusively assigns L3 to the CPU complex.

The iGPU is a Volta $\mu$architecture graphic processor featuring a peak clock of 1377 MHz, in which its constituent 512 CUDA cores are grouped in 8 SMs. Notable differences with respect to the older Pascal generation lie in the smaller number of CUDA cores per SM (from 128 to 64) and the presence of tensor cores, i.e. application specific circuitry for tensor processing operations typically used in neural network inference and training [20].

Xavier's main memory is a 16 GiB 256-bit LPDDR4 DRAM clocked at most at 2133 MHz, hence reaching a bandwidth of close to 137 GiB/s. Such remarkably large memory bandwidth figure accounts for the fact that Xavier is the most complex heterogeneous SoC in our analysis.

Indeed, compared to the other platforms, Xavier also features some ASICs accelerators that act as additional memory controller clients: the Programmable Vision Accelerator (PVA) and two Deep Learning Accelerators (DLA). The former is an independent compute engine composed of two Vector Processing Units (VPU), each coupled with a DMA engine for host-device data transfers as well as private on-chip memory. It is able to efficiently compute video processing algorithms on fixed data sizes and seamlessly integrate with the other platform's accelerators. Its maximum clock is 1088 MHz and the list of supported algorithms includes Stereo Disparity Estimator, KLT Bounding Box Tracker, Gaussian Pyramid Generator, Image Convolver, Separable Image Convolver, Box Image Filter and Gaussian Image Filter.

The DLA is an accelerator capable of efficiently performing neural network inference [21], [22] over network topologies featuring convolution, deconvolution, pooling, fully connected, normalization, scaling and element-wise layers over a limited set of data precision. Supported activation functions are ReLU, Sigmoid and Hyperbolic Tangent. Its peak clock is 1395.2 MHz. A schematic representation of the Xavier's compute engines and related memory subsystem is visible in Fig. 2.

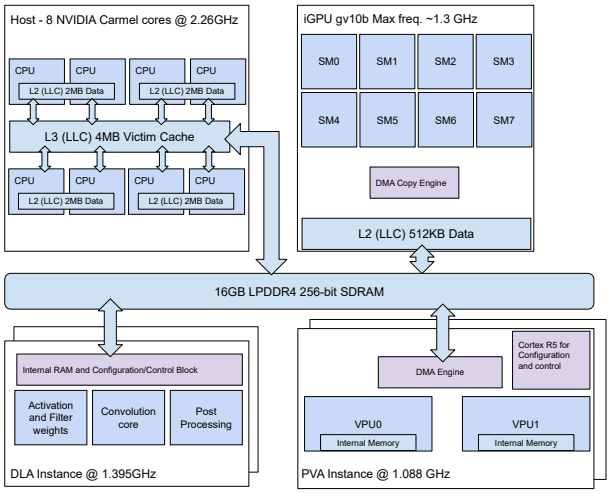


**Fig. 2:** Compute engines and related memory contention points in the NVIDIA Jetson Xavier.

## IV. *HeSoC-mark*: AN OPEN-SOURCE TOOL FOR MEASURING MEMORY CONTENTION

In this paper we introduce *HeSoC-mark*, which is an extensible, open-source collection of applications aimed at stressing the available compute engines of a selected platform

[3]Available at https://developer.nvidia.com/embedded/downloads

with memory intensive jobs. *HeSoC-mark* has been developed by accounting for the unprecedented variety of compute engines accessible in the Xavier SoC. However, the applications contained in *HeSoC-mark* are designed to be independent modules, and they can be utilized in different heterogeneous systems. The constituent applications within *HeSoC-mark* are able to generate interference and memory-bound workloads from which we can measure both bandwidth and latencies. The applications are summarized as follows:

- `membench`: measures latencies in different points of the CPU complex cache hierarchy by performing memory read accesses over a pre-allocated 50 MiB buffer. Latencies are recorded from repeated pointer walks on a variable working-set size (WSS) within the buffer. In order to evaluate the CPU complex prefetching abilities and the memory controller ability to reorder memory request transactions, memory access pattern can be tuned to be sequential or random. Reported values are average latencies of reading a single integer word.
- `meminterf`: generates high-memory traffic by iteratively executing *memcpy* or *memset* POSIX-defined functions over buffers of parameterized dimension.
- `cudainterf` and `cudameasure`: The first one can be used to generate interference from the GPU, whereas the second one measures the latencies of GPU workloads' execution time. They both generate memory traffic from the GPU by exploiting the device's copy engine and/or SMs. More specifically, the user can select to submit CUDA 100% memory bound kernels with/without UVM [23] as the interfering SM activity. Available Copy Engine activities includes memset and memcpy (*d2d, d2h/h2d*) over buffers of parameterized dimension.
- `vpipva`: submits computer-vision related algorithms through the NVIDIA Vision Programming Interface (VPI) [24]. Such algorithms can be executed on CPU, iGPU and the PVAs (if available in the tested platform).
- `trtdla`: creates Fully Connected Neural Networks and performs inference on a single instance of the DLA by exploiting the TensorRT API [25]. Neural Network creation parameters are number of layers and input tensor size.

Additional tools for *HeSoC-mark* are `cpubench`, which is a collection of single-core CPU-only workloads taken from known benchmark suites and `interfgen`, which is able to generate test scripts starting from an XML representation of the possible set of tasks for which we assess the impact of memory contention. For space constraints, we refer the reader to consult the documentation in the *HeSoC-mark* `git` repository, as we release *HeSoC-mark* as on open-source contribution[4]. Unless otherwise specified, all the experiments for the analysed platforms have been performed using *HeSoC-mark* and by setting the platforms' best performing (i.e. highest frequencies) power profiles. Installed *JetPacks* (i.e. the Operating System and the SDK libraries maintained by NVIDIA) are updated to the latest available release (4.3).

[4]available at https://git.hipert.unimore.it/mem-prof/hesoc-mark

## V. CPU and Accelerators' bandwidth characterization

The first set of experiments we performed was aimed at assessing how memory bandwidth is split among different CPU cores and the iGPU. With specific reference to the Xavier SoC, we also estimated the maximum bandwidth requested by the application specific compute engines (i.e. DLA and PVA).

### A. Multicore CPU bandwidth composition

In order to measure the bandwidth composition on each CPU core, the `bw_mem` utility provided by the LMBench suite [26] has been used. Resulting measures are depicted in Fig. 3.

On the CPU-side, we highlight that the `bw_mem` utility allows the user to specify: (1) the number of parallel threads that will perform memory accesses; (2) the kind of operation to be performed on memory; and (3) the working set size (WSS) of the interested memory buffer. In our tests, we varied the number of cores from 1 to $N$, with $N$ being the number of available cores in the tested platform. In order to measure the maximum value of bandwidth requested by the entire CPU complex (and to minimize the effect of shared caches), one process instance of `bw_mem` has been launched per island. This implied launching a single instance for the Jetson Nano, up to two for the TX2, and up to four for the Xavier SoC. Affinities for the relevant CPU cores are then defined through the `taskset` utility. Performed operations are *read* and *write* over a 128 MiB buffer, that is large enough to measure bandwidths from/to system DRAM.

In the left inset of Fig. 3 the maximum read and write CPU bandwidth are reported for each platform. Measurements are obtained by aggregating bandwidths from all the available cores. In order to understand how CPU memory read bandwidth is contented by each CPU core, the right inset of Fig. 3 shows how the measured bandwidth grows as the number of cores increases. The A57 island of the Jetson TX2 performs slightly better than the whole Nano CPU complex: this is due to the fact that the ARM cluster implementation is the same on both these platforms, except for slightly higher clocks for CPU and DRAM in the TX2. The effect on the aggregated bandwidth when TX2's Denver cores are active is dramatic (a 3.5x increase, as visible in Fig. 3, right inset, after the activation of *c3*).

Regarding the Xavier SoC, the added L3 cache, the higher memory and CPU frequency, and the larger bus width to DRAM enables the Xavier's Carmel CPU complex to reach significantly higher peak bandwidths. It is interesting to notice that bandwidth is partitioned on an island-basis: a single CPU core is able to consume 20 GiB/s, but this value is halved when the companion core (the other core within the same island) is active. More precisely, the total bandwidth from an entire island is around 18.5 GiB/s. The culprit for such a deterioration is the shared L2 cache. These effects are noticeable in the right

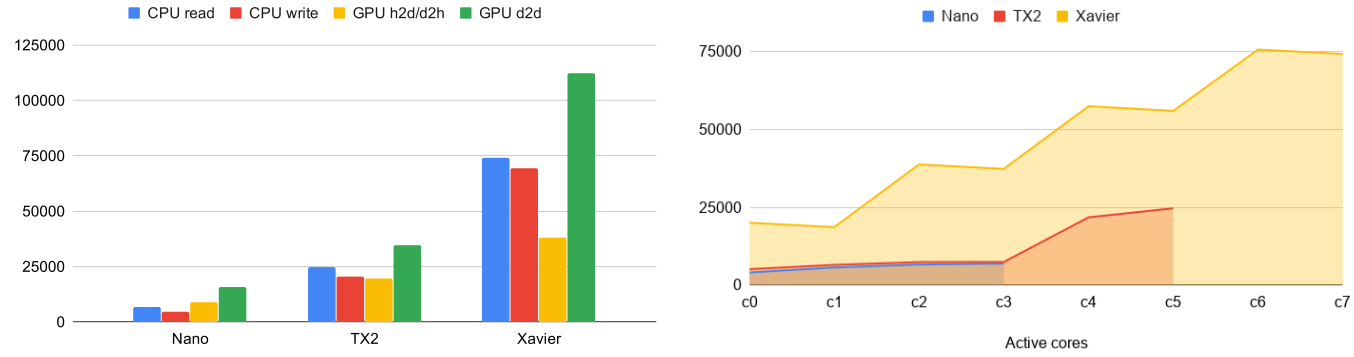


**Fig. 3:** Maximum bandwidth [MiB/s] from CPU and iGPU (right) and Multicore CPU memory read bandwidth composition (left).

inset of Fig. 3, in which the Xavier bandwidth composition resembles a step function, in which rising edges are found at each island activation.

### B. iGPU and Xavier's accelerators bandwidth

The peak bandwidth for the iGPU is calculated using the `bandwidthTest` utility provided in the pre-installed CUDA SDK samples. In this setting, measurements involve copying memory from a CPU (host)-visible memory area to a GPU (device)-only address space; host memory is managed through pinned allocations, i.e., non-pageable buffers in which read/write operations are able to saturate the available bandwidth measured from the GPU DMA engines. This is indicated in the left inset of Fig. 3 as *h2d/d2h*. Maximum achievable bandwidth is registered when memory is copied between two device-only buffers (labelled as *d2d* in the left inset of Fig. 3). Even in this iGPU scenario, WSS abundantly exceeds the size of the processing unit's LLC.

For the sake of completeness, we also provide a bandwidth estimation for the DLAs and the PVAs in the Xavier SoC. Due to the limited programmability of such compute engines, it is not possible to precisely time their memory transactions. We therefore measure a *memory and compute bandwidth* by estimating the time of completion of a submitted task in relation to the total amount of data processed. Using `trtdla` from *HeSoC-mark* we measured that a single DLA can process a neural network composed of 200 fully-connected layers and 200 sigmoidal activation layers, in fp16 precision having an input tensor of size 32x32 in 18.31 ms (average). Considering that such a network requires 402 MiB of memory, we therefore infer a bandwidth of close to 21 GiB/s. Similarly and by using our `vpipva` application, both the PVA engines are able to process in parallel eight instances of a Gaussian Image filter, which involves reading an input image and writing to an output image, both sized 3200x2400 pixel in a 2-byte-per-pixel format on an average of 12.7 ms. Considering that the amount of processed data is close to 228 MiB, the compute and memory bandwidth of both the PVA instances should be close to 18 GiB/s.

## VI. Memory interference: effect on latencies

In order to measure the effect on memory interference on the platform performance and predictability we used the `membench` utility from our *HeSoC-mark* suite. In these tests and for the sake of completeness, TX2 related measurements are separated between latencies observed by a core belonging to the A57 island and a core within the Denver island. In Fig. 4 we can see the baseline (i.e. no interference) values for sequential reads (left) and random access pattern (right). Unless otherwise specified, all the X-axis in these latency graphs are logarithmic.

In the left inset of Fig. 4 we can see how the TX2's ARM island performs slightly better than nano's A57 core, which is an effect that we noticed in the previous set of experiments (see section V-A) and justified by TX2's higher clocks. The Denver core presents half the latency for accessing main memory compared to its ARM A57 island (from 15 to 7 ns). Not surprisingly, the Xavier Carmel core presents the smallest latency value in main memory (4.2 ns). It is worth noticing the latency spike that occurs in every platform at the boundaries between the L2 and the subsequent memory hierarchy level (i.e. main memory for nano and TX2, L3 for Xavier). This effect was also noted in previous generation of NVIDIA Tegra platforms [3].

Related to the prefetching and memory request reordering capabilities for each SoC, it is important to notice the dramatic increase on latencies when the memory access pattern is random (right inset of Fig. 4): the A57 islands in the nano and TX2 now present identical average latency values (almost 10x w.r.t. the sequential baseline), whereas the NVIDIA proprietary designed CPU cores feature a latency increase of almost 20x for Denver and 30x for Carmel compared to sequential memory read accesses.

### A. Interference from the CPU complex

The second batch of tests involved measuring sequential and random read access pattern across the memory hierarchy from one core while all the other CPU cores are heavily performing memory transactions. The core under observation (i.e. the one in which we measure latencies) executes one instance of the `membench` application, whereas the other interfering cores

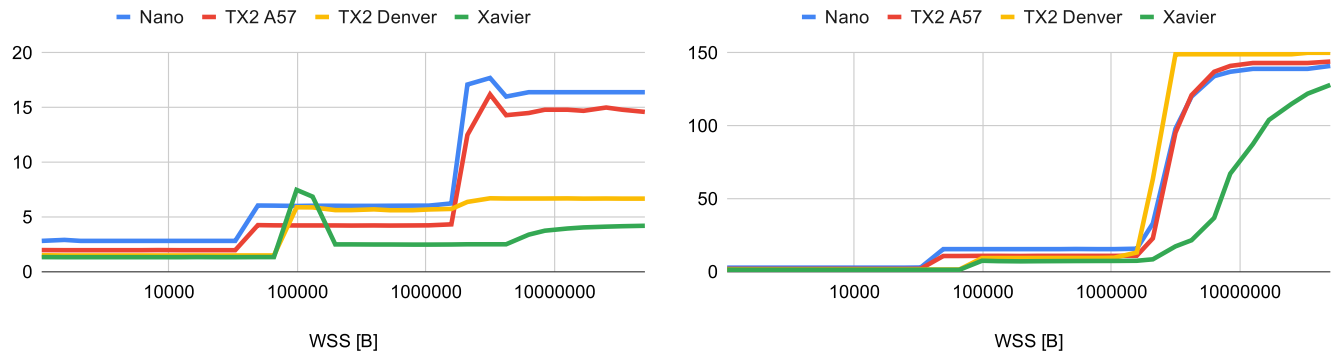


**Fig. 4:** Single-core baseline latencies [ns] with no interference. Sequential (left) and random memory read access patterns (right).

are each executing an instance of the `meminterf` application. Test results are visible in Fig. 5.

In the left inset of Fig. 5 (sequential reads) we can see that, when accessing system DRAM (WSS $\geq$ 2-4 MiB), the most robust complex towards CPU interference is the Denver island within the TX2 SoC. The Denver is able to contain the latency increase w.r.t. its baseline within a 60%, whereas the more recent Carmel core in Xavier presents a latency deterioration that rises up to a 5x compared to its baseline.

Concerning the interference on L2 for both sequential and random reads (left and right inset of Fig. 5), we can see the opposite situation: the Xavier's Carmel holds on latency values that are close to its baseline up until the L2/L3 boundary, whereas all the other CPU complexes show a deterioration on latencies that ranges from an almost 2x for Denver sequential to values close to 20x for both the TX2's islands for random accesses.

Regarding the latencies measured for the A57 islands in the nano and TX2, the A57 in the TX2 is more sensible to interference, as can be seen in Fig. 5 by looking at both sequential and random patterns. This might be explained by the fact that the TX2 has a higher number of cores contending to memory. However, this is not sufficient to explain the low latencies seen with the TX2's Denver for sequential reads. We therefore decided to investigate the effect of inter- and intra-CPU complex interference in the presence of heterogeneous CPU islands, presenting the results in Fig. 6.

In the left inset of Fig. 6, we notice the effect of the separated LLCs between the Denver and the A57 complexes. More specifically, results show that inter-island interference is negligible compared to the intra-island one, e.g. a single A57 core vs all the Denvers, or vs the remaining A57 cores. Such a partitioning, however, is ineffective for the A57 when accessing system DRAM, as both inter- and intra-island interference seems to add on latencies, as seen in Fig. 5. In the right inset of Fig. 6, we see the same effect on Xavier. Recall that a Xavier island is a pair of cores sharing L2. From the results, we infer that interference caused by CPU2 (a core belonging to a different island compared to the observed CPU) is negligible. By adding 6 more interfering cores from the all the other islands (i.e. CPU from 2 to 7), the latency increase accounts for less than 2x compared to CPU0 baseline in sequential accesses in DRAM. The huge bulk of interference is therefore provided by CPU1 alone (i.e. the observed CPU's companion core), able to cause a latency increase of almost 3x compared to the same non-interfered Carmel core.

### B. *Interference from the accelerators*

In these tests, memory reads are performed by one CPU core, while one or more cores located in different CPU islands are submitting memory intensive jobs to the compute accelerators available in the examined SoCs. The observed CPU core is still executing the `membench` utility, whereas different applications in the *HeSoC-mark* suite have been used to generate the highest possible memory traffic from all the other engines (i.e. `trtdla` and `vpipva`).

In the left inset of Fig. 7, we can see plots related to the Nano and TX2, both for random and sequential read transactions. In all these plots, the interfering accelerator is the iGPU that performs the equivalent of the *memset* function on device-only visible buffers through the `cudainterf` utility in *HeSoC-mark*. The iGPU memory traffic generated by such a function occurs through a DMA engine, hence no effects on latency are visible for WSS $\leq$ CPU LLC size. Compared to the non-interfered baseline, latency increases range from 1.87x (sequential reads from a TX2's Denver) to 3.8x (random reads from a nano's CPU). This is a substantial improvement over previous generation Jetsons [3].

Xavier results have been highlighted in a dedicated graph (Fig. 7, right inset) because of the wider variety of application-specific accelerators available. No matter if the observed core is accessing memory in a sequential or random pattern, the Carmel CPU complex is only negligibly perturbed by each of the other engines. Such negligible latency deterioration only applies to accesses in DRAM, as all the other engines' memory transactions operate in DMA. This improvement over the other analysed SoCs can be explained by the notable total bandwidth made available by the memory controller: if a single Carmel core consumes up to 20 GiB/s and both the DLAs account for around 40 GiB/s, there is still plenty of unused bandwidth towards the DRAM (almost 80 GiB/s). Notable effects on latencies are dramatic only if all or most of these memory clients are active within the same time window.

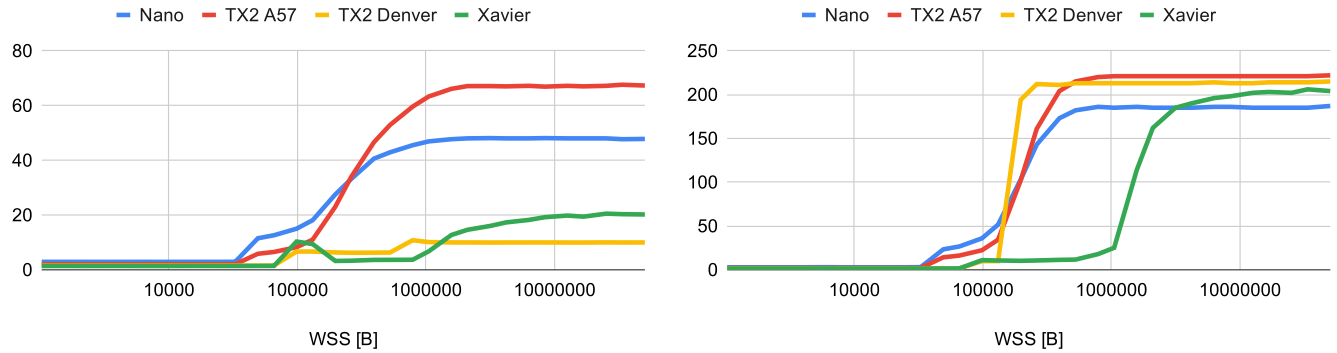


**Fig. 5:** Single-core latencies against interference from the rest of the CPU complex. Sequential (left) and random memory read access patterns (right).

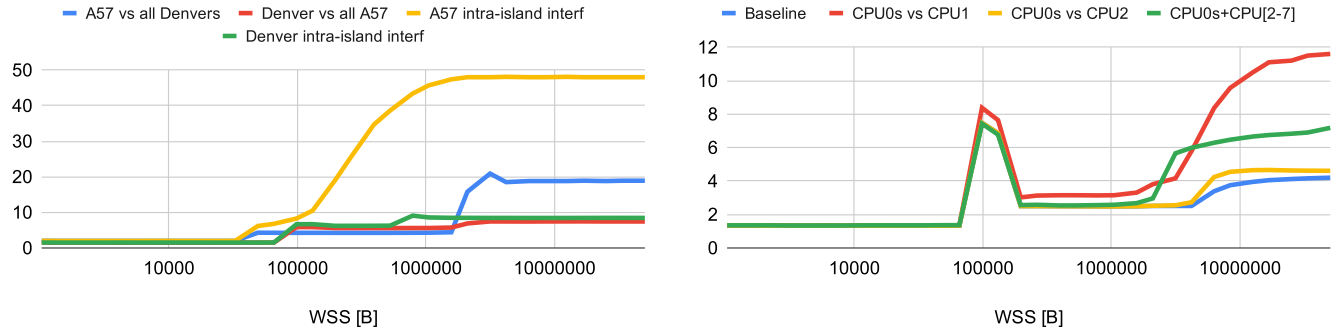


**Fig. 6:** Inter and Intra CPU interference in heterogenous CPU complexes for sequential memory accesses. TX2 (left) and Xavier (right).

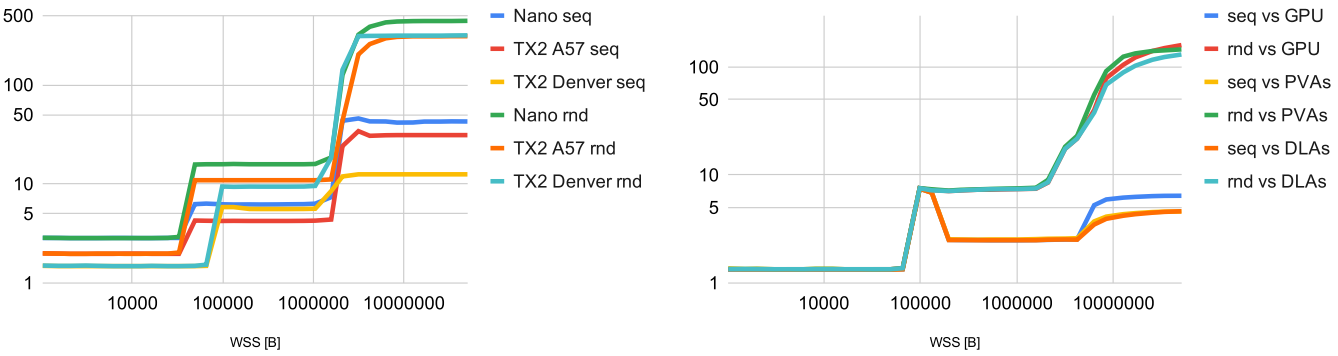


**Fig. 7:** Single-core CPU memory read latencies [ns] against interference from iGPU on nano and TX2 (left) and Xavier single-core CPU memory read latencies [ns] against interference from accelerators (right). Both axes are logarithmic.

The outcome of the experiments in which both the CPUs and the available accelerators are interfering an observed CPU core are presented in Fig. 8. In nano and TX2, such tests have been performed by having one instance of `membench` on the observed core, the same GPU interference detailed in this section launched from another core, and each remaining core executing one instance of `meminterf`. We maintain the same settings for Xavier, but only three cores are executing `meminterf`, while two cores are dedicated to submit work to each DLA, and one other core is submitting work to both the PVAs. Not surprisingly, combined accesses from the different memory clients causes the highest possible magnitude of interference in all the tested settings. Latency deterioration compared to the non-interfered baseline ranges from 2.6x for a TX2 Denver sequential access in DRAM, to a dramatic 45x for the same core reading memory on a random pattern at the boundary between its LLC and DRAM.

For better readability, the worst case scenarios for memory interference are summarized in Fig. 9. In all the depicted configurations, the highest magnitude for interference occurs in correspondence to a certain range of WSS values, that we call *boundaries*: $\pm$ 25% of CPU LLC for nano and TX2, and between Carmels' L2 and L3 for Xavier. Since no accelerator is capable of interfering in the CPU cache hierarchy, such a result can be then explained by considering that the DRAM bandwidth is oversubscribed due to all the other cores and accelerators accessing system memory, so that additional cache misses caused by interfering CPU cores incur in extra latencies

that would otherwise be absent.

### C. Interference on interrupts' response time

A key requirement for real-time systems is to maintain a small and predictable interrupt response time. To this purpose, we studied the effects of memory interference on interrupt latencies using the Linux `cyclictest` utility [27]. The test measures the difference between the generic-timer-triggered wake-up time of a thread, and the time at which it actually wakes up. We run the test in a given CPU core for 900K iterations with a 200 $\mu$s interval, while interference from all the other engines is optionally activated as in the experiments shown in Section VI-B. The latency distributions plotted in Fig. 10 shows a clear increase both in the average and maximum recorded latency (6x to 10x), as well as in the width of the curve. As corroborated by previous findings [9], the main cause is the increased probability of a miss in the L2 shared and contended cache for the interrupt service routine's instructions and data, but also from other contention points in the memory hierarchy. Memory access contention is a significant threat also to the worst-case response time.

In Fig. 10, two plots per CPU complex are shown: the non-interfered baseline and the same core under combined interference. The first effect to notice is the difference in the overall shape of the distributions. NVIDIA-designed CPU (TX2's Denver and Xavier's Carmel) plots almost resemble a bimodal distribution, which is in contrast with the Weibull-like plots for the A57. This might be an effect of the aggressive, and therefore unpredictable, OoO execution policies for such processors. Regarding the TX2, the A57 island is again more sensible to the combined interference than the Denver island, showing an average deterioration of response time compared to the baseline of 8x on A57 versus 3.8x on Denver. The Carmel CPU in Xavier shows a similar performance deterioration as on the Denver. However, absolute baseline latency values are 3.5x lower. The largest recorded latencies for all the CPUs are located within the 450-550 $\mu$s range.

### D. Interference on the iGPU

While all the previous tests were observing a single CPU core, we now measure the iGPU latency of executing a CUDA kernel when all the other memory clients are acting as interference. More specifically, the benchmark application is `cudameasure` from our *HeSoC-mark* suite. This application periodically submits a 100% memory-bound CUDA kernel to the iGPU's SM. This kernel touches 200 MiB/s of data. We observe the variation of the kernels execution times while all the other cores are executing a combination of all the interfering processes from *HeSoC-mark*.

Results are depicted in Fig. 11. The maximum kernel execution times compared to a non-interfered baseline go from 1.4x for the nano to 1.66x for the TX2 when all the other CPU cores are fully utilizing their memory bandwidth. In the Xavier iGPU, we observe something similar to what occurs to a Carmel core when only a single accelerator is accessing memory, i.e. little to no interference (e.g. 3% when PVAs are active). The highest magnitude of contention occurs if interference is generated by more than one engine: from 1.58x to 2.18x when CPUs and PVAs or when all (CPUs, PVAs and DLAs) the engines are active. The interference contribution given by the DLAs alone is the highest. However, this is a parasite effect given by the TensorRT API implementation, which submits input and output network conversion layers to the iGPU. In this way, the measured latencies for the GPU are also perturbed by the GPU context scheduler [28].

## VII. Discussion

From the experimental results regarding single memory accesses, significant improvements on both sequential baseline and interfered latencies are observable on the newest CPU complexes when compared to older designs, as witnessed by the impressively low average latencies for Denver and Carmel cores. A higher performance deterioration is instead observed in case of random access patterns. In this case, while Xavier performs better than both the Nano and the TX2, it can still suffer from a performance deterioration due to memory interference between 8x and 15x, depending on the stress imposed on the memory subsystem. Excluding the parasite effect of the Xavier DLA towards the GPU scheduler, there is no substantial difference among the three considered platforms in terms of GPU kernel execution times when the iGPU is interfered by the rest of the system.

Focusing on interrupt handling latencies, we highlight that the NVIDIA-designed CPUs (i.e. TX2's Denver and Xavier's Carmel cores) present a significantly wider plot on the interrupt response time distribution, as shown in section VI-C. This is an effect of the aggressive and undocumented OoO execution policies that characterizes the NVIDIA CPU designs since Denver. While the performance improvement of Xavier over previous generation Tegras is evident, it is unclear whether such an improvement is due to the enhancements through the whole memory hierarchy (i.e. the impressive total available memory bandwidth and the additional CPU-cache level) or to the more sophisticated arbitration policies at memory controller level.

## VIII. Conclusion

In this work, we presented an extensive set of measurements with the aim of quantifying the impact of memory interference on modern embedded architectures. More specifically, we investigated how subsequent generations of the widely available NVIDIA Jetson platforms evolved on the point of view of arbitration of their heterogeneous memory clients. For this purpose, we also presented *HeSoC-mark*, an extensible benchmark suite able to define workloads for the wide variety of compute engines made available in the Jetson embedded boards. In all the three most recently released NVIDIA Tegra-based SoCs that we investigated, we tuned the relevant *HeSoC-mark* applications to either act as a memory interference generator, or to emulate the behaviour of a memory-bound latency sensitive task.

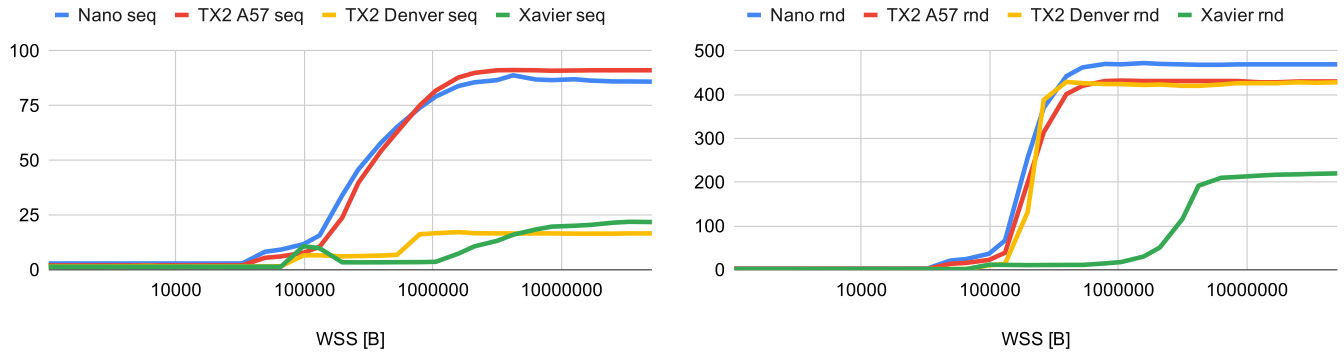


**Fig. 8:** Single-core latencies against combined interference from the rest of the SoC (iGPUs, CPU complex and DLA and PVA in Xavier). Sequential (left) and random memory read access patterns (right).

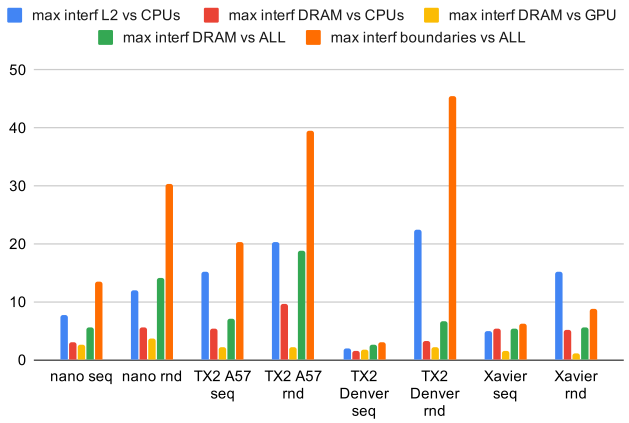


**Fig. 9:** Worst case scenarios for memory interference for all the tested platforms and configurations.

In summary, memory contention remains the most important issue on both a performance and predictability perspective. The results of our tests calls for mitigation strategies that could reduce the impact of memory contention, especially when predictable timing guarantees are to be given. Memory-aware scheduling policies could be adopted in this sense [29], [30]. The relevant hardware features described in this paper are to be considered to efficiently distribute tasks within the CPU complexes. We highlight that memory-centric scheduling often implies some sort of partitioning within DRAM banks or portions of shared caches [31], [32]. In order to minimize the overhead of such software mitigation approaches, we will investigate how novel, hardware-based mechanisms can be dynamically tuned for solving the memory contention bottleneck in heterogeneous embedded SoCs [33], [34], [35].

## ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's ECSEL JU Programme under the NEW CONTROL Project https://www.newcontrol-project.eu grant agreement n. 826653.

## REFERENCES

[1] M. Hähnel and H. Härtig, "Heterogeneity by the numbers: A study of the {ODROID} xu+ e big. little platform," in *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, 2014.

[2] A. Skende, "Introducing "parker": Next-generation tegra system-on-chip," in *2016 IEEE Hot Chips 28 Symposium (HCS)*. IEEE, 2016, pp. 1–17.

[3] R. Cavicchioli, N. Capodieci, and M. Bertogna, "Memory interference characterization between cpu cores and integrated gpus in mixed-criticality platforms," in *22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA)*, 2017.

[4] S. Yamagiwa and K. Wada, "Performance study of interference on gpu and cpu resources with multiple applications," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8.

[5] B. M. Tudor, Y. M. Teo, and S. See, "Understanding off-chip memory contention of parallel programs in multicore systems," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 602–611.

[6] H. Wen and Z. Wei, "Interference evaluation in cpu-gpu heterogeneous computing," in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

[7] Z. Majo and T. R. Gross, "Memory management in numa multicore systems: trapped between cache contention and interconnect overhead," in *Proceedings of the international symposium on Memory management*, 2011, pp. 11–20.

[8] I. S. Olmedo, N. Capodieci, and R. Cavicchioli, "A perspective on safety and real-time issues for gpu accelerated adas," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 4071–4077.

[9] I. Sanudo, P. Cortimiglia, L. Miccio, M. Solieri, P. Burgio, C. D. Biagio, F. Felici, G. Nuzzo, and M. Bertogna, "The key role of memory in next-generation embedded systems for military applications," in *Proceedings of 6th International Conference in Software Engineering for Defence Applications, SEDA 2018, Rome, Italy, June 7-8, 2018*, 2018, pp. 275–287.

[10] F. Mazzocchetti, P. Benedicte, H. Tabani, L. Kosmidis, J. Abella, and F. J. Cazorla, "Performance analysis and optimization of automotive gpus," in *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2019, pp. 96–103.

[11] P. Vivekanandan, G. Garcia, H. Yun, and S. Keshmiri, "A simplex architecture for intelligent and safe unmanned aerial vehicles," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 69–75.

[12] S.-H. Lee and C.-S. Yang, "A real time object recognition and counting system for smart industrial camera sensor," *IEEE Sensors Journal*, vol. 17, no. 8, pp. 2516–2523, 2017.

[13] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, pp. 1–44, 2011.

[14] NVIDIA, *CUDA Toolkit Documentation*, NVIDIA. [Online]. Available: https://docs.nvidia.com/cuda/

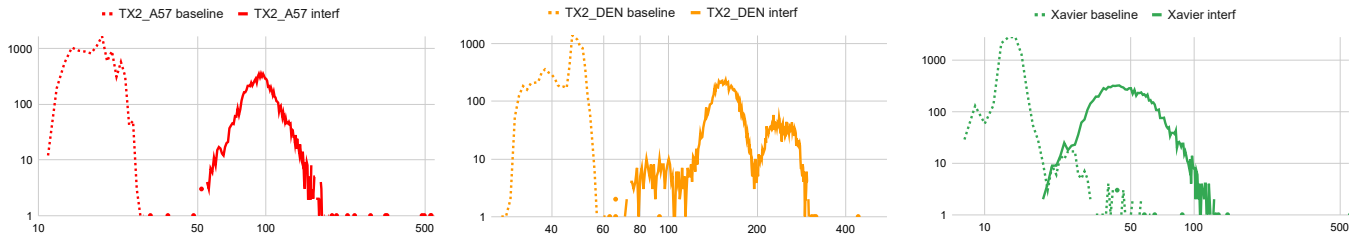


**Fig. 10:** Interrupt response time distribution [$\mu$s/occurrences]: `cyclictest` output of the CPU complexes within the tested platforms.

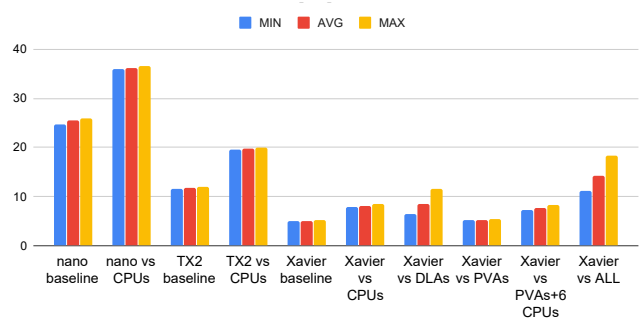


**Fig. 11:** Interference to the iGPU. Copy Kernel execution time [ms].


[15] , "NVIDIA Charts Its Own Path to ARMv8, White Paper," Tirias Research, Tech. Rep., 2014.

[16] A. Holdings, "Arm cortex-a57 mpcore processor, technical reference manual, 2014."

[17] R. Krishnaiyer and W. Li, "Adaptive prefetch for irregular access patterns," Dec. 26 2006, uS Patent 7,155,575.

[18] Z.-n. Cai, W. G. Auld, and J. D. Gilbert, "Apparatus and method for an adaptive multiple line prefetcher," Jun. 5 2007, uS Patent 7,228,387.

[19] C. Escuín Blasco, "Analysis and simulation of data prefetching algorithms for last-level cache memory," Master's thesis, Universitat Politècnica de Catalunya, 2018.

[20] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, vol. 23, 2019.

[21] F. Farshchi, Q. Huang, and H. Yun, "Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim," *arXiv preprint arXiv:1903.06495*, 2019.

[22] W.-F. Lin, C.-T. Hsieh, and C.-Y. Chou, "Onnc-based software development platform for configurable nvdla designs," in *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2019, pp. 1–2.

[23] S. Chien, I. Peng, and S. Markidis, "Performance evaluation of advanced features in cuda unified memory," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 2019, pp. 50–57.

[24] NVIDIA, *VPI - Vision Programming Interface, API Reference*, NVIDIA. [Online]. Available: https://docs.nvidia.com/vpi/index.html

[25] ——, *TensorRT Developer Guide*, NVIDIA. [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html

[26] L. W. McVoy, C. Staelin *et al.*, "lmbench: Portable tools for performance analysis." in *USENIX annual technical conference*. San Diego, CA, USA, 1996, pp. 279–294.

[27] T. Gleixner, "cyclictest 1.0-3," 2016. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

[28] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 119–130.

[29] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo, "Global real-time memory-centric scheduling for multicore systems," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2739–2751, 2015.

[30] P. Burgio, A. Marongiu, P. Valente, and M. Bertogna, "A memory-centric approach to enable timing-predictability within embedded many-core accelerators," in *2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*. IEEE, 2015, pp. 1–8.

[31] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 1–14.

[32] N. Suzuki, H. Kim, D. De Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *2013 IEEE 16th International Conference on Computational Science and Engineering*. IEEE, 2013, pp. 685–692.

[33] L. Pons, V. Selfa, J. Sahuquillo, S. Petit, and J. Pons, "Improving system turnaround time with intel cat by identifying llc critical applications," in *European Conference on Parallel Processing*. Springer, 2018, pp. 603–615.

[34] K. Sivakumaran and A. Siromoney, "Cache control techniques to provide qos on real systems," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 5161–5188, 2019.

[35] ARM, *Arm Architecture Reference Manual Supplement. Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A*, ARM. [Online]. Available: https://developer.arm.com/docs/ddi0598/latest