embeddedworld2019
Exhibition&Conference
... it's a smarter world

# *Handling Mixed Criticality on Modern Multi-core Systems: the* HERCULES *Project*

Paolo Gai, Claudio Scordino
Evidence Srl
Pisa, Italy
Email: {pj,claudio}@evidence.eu.com

Marko Bertogna, Marco Solieri, Tomasz Kloda, Luca Miccio
Universitá di Modena e Reggio Emilia
Modena, Italy
Email: marko.bertogna@unimore.it, ms@xt3.it,
tomasz.kloda@unimore.it, 206497@studenti.unimore.it

***Abstract*—Recently, the industry has shown a growing interest for executing activities with different levels of criticality on the same platform, aiming at reducing the time-to-market and the recurrent hardware costs. In this context, we illustrate the results achieved by the three-year HERCULES project for building a high-performance real-time architecture using modern low-power ARM platforms. In particular, we present a system consisting of both the Linux OS and an automotive RTOS on top of an open-source hypervisor. Besides presenting the overall software architecture, we highlight the issues related to the interference on shared hardware resources, illustrating some techniques specifically designed for mitigating or preventing such problems. Finally, we provide some experimental results about the effectiveness of the proposed approach.**

***Keywords*—AUTOSAR, embedded, hypervisor, Linux, multi-core, RTOS.**

## I. INTRODUCTION

During the recent years, the industry has shown a growing interest for running activities with different levels of criticality on the same platform. These could consist, for example, of non-critical activities (e.g., monitoring, logging, human-machine interface, multimedia, data backup) together with safety-critical real-time tasks. The rationale behind this interest is the continuous need for reducing the time-to-market as well as the recurrent hardware costs. This is particularly suitable for the automotive market, where new infotainment and dash-board functionalities might be coupled with traditional (e.g., engine/brake control) or innovative (e.g., assisted/autonomous driving) safety-critical tasks.

Linux is a full-featured operating system (OS), originally designed to be used in server or desktop environments. Since then, Linux has evolved and grown to be used in almost all computer areas, including embedded devices. Thanks to its high modularity and scalability, in fact, Linux can run on either a small microcontroller or a supercomputer. Unfortunately, the standard mainline Linux kernel is not suitable for being used as an RTOS because it has been designed to be general-purpose. The main design goal has in fact concerned the optimization of the average throughput rather than meeting the timing constraints of each running application. As a result, the standard Linux kernel does not provide good real-time performance.

In the course of the years, several open-source projects have aimed at creating a real-time version of the Linux operating system. The dual-kernel approach proposed by some projects, for example, creates a Hardware Abstraction Layer (HAL) and modifies the interrupt handling routines for executing a tiny RTOS prior than Linux ([1], [2], [3]). The Linux kernel and its tasks are thus executed at a priority lower than the real-time tasks. Despite these projects have successfully proven the feasibility of the proposed approach, they have been unable of getting enough traction and visibility to become widespread. A more popular project, PREEMPT_RT [4] aims at reducing the maximum latencies experienced by a real-time task on the mainline Linux. Unfortunately, all these projects present issues for the certification needed in some industrial domains (e.g., automotive, avionics, medical): the huge code size of the Linux kernel, in fact, discourages, or even prevents, a proper certification of the system.

Aiming at a potential certification of the safety-critical part of the software stack, the HERCULES project has therefore taken a different approach: run a certified RTOS on top of a tiny hypervisor with a reasonable code size; then, use the hypervisor for running also a general-purpose OS (i.e., Linux); finally, use the latest research results on predictable execution in order to achieve better predictability on COTS multicore hardware.

## II. THE HERCULES ARCHITECTURE

Figure 1 illustrates the overall architecture of the HERCULES software stack. The next paragraphs will describe the most important components.

### A. Hypervisor

A hypervisor is a (software or hardware) component for running multiple virtual machines on the same platform. Depending on the specific scenario, these virtual machines can either contain an operating system or run bare-metal applications.

Historically, hypervisors have been classified as either:

- Type 1: *native* (or bare-metal) hypervisors, directly running on hardware.

This work has been funded by the European Commission through the HERCULES H2020 project (GA-688860).

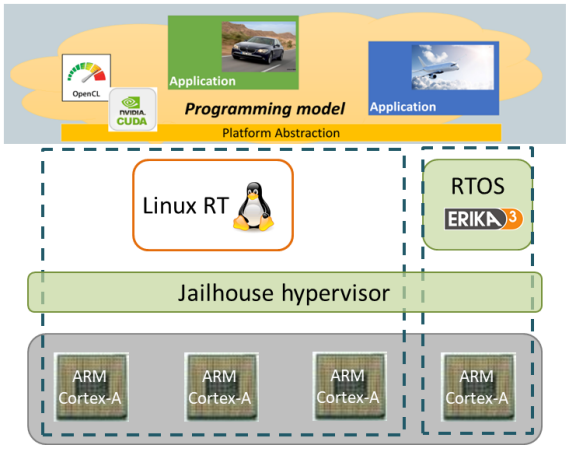


Fig. 1. The HERCULES software architecture

- Type 2: *hosted* hypervisors, running on a conventional "host" operating system.

A further classification depends on the API exposed to the guests:

- In case of *full virtualization*, the guest OS is not aware of virtualization, as it uses the same API used when running on the bare-metal platform.
- In case of *paravirtualization*, instead, the guest OS must use a different API, designed for easing certain operations or improving performance.

Jailhouse [5] is a young project aiming at creating a small and lightweight hypervisor for industrial-grade applications. The hypervisor is developed by Siemens but released as open-source software under a mix of GPL and BSD licensing. Like some other novel hypervisors, Jailhouse is not type-1 nor type-2, as it is hosted by the Linux OS, but converts it to a type-1 hypervisor.

To pave the way to a potential certification in safety-critical contexts, the code is kept to a minimum. For this reason, the hypervisor does not implement paravirtualization and partitions the hardware resources among the different guests, without adding emulation or scheduling. A core cannot thus be shared among different guests; moreover, the hypervisor cannot run unmodified OSs (like Windows or VxWorks).

In HERCULES we have used Jailhouse for running an RTOS alongside the Linux OS. This way, we have been able of executing real-time safety-critical control applications and general-purpose non-critical tasks concurrently on the same platform.

To maximize performance and reduce the code size, Jailhouse leverages hardware-assisted virtualization available in the instruction set of the latest x86-64 and ARM Cortex-A architectures. In particular, the software stack of HERCULES has been ported onto the following hardware platforms:

- Nvidia Jetson TX2 [6], including a quad-core ARM Cortex-A57, a dual-core Denver2, three Cortex-R5 and a Nvidia Pascal GPGPU.
- Xilinx ZCU102 [7], including a quad-core ARM Cortex-A53, a dual-core ARM Cortex-R5, an FPGA fabric and a Mali-400 GPGPU.

### B. ERIKA *Enterprise RTOS*

ERIKA Enterprise [8] is an RTOS specifically designed for automotive Electronic Control Units (ECUs). It is based on the OSEK/VDX and AUTOSAR standards, with an on-going ISO26262 certification process. It already supports several hardware architectures, including ARM Cortex-M/R/A, Infineon Tricore AURIX TC2xx/TC3xx and x86-64.

The RTOS, which has minimal footprint (i.e. a few KBs) and multi-core support, is already used by renowned companies operating in the automotive and household appliances markets as well as by several international research projects. The source code is provided under dual licensing: an open-source GPL license, plus an optional linking exception fee for industrial products.

In HERCULES, the ERIKA RTOS has been ported on top of the Jailhouse hypervisor for the reference platforms illustrated above. ERIKA has been then used for running the real-time tasks with tight timing requirements of an autonomous driving use-case.

### C. *Inter-guest communication*

Jailhouse provides a sophisticated mechanism for sharing information between different guests. The mechanism is based on the ivshmem model of the Qemu emulator [9]. In HERCULES, we borrowed a library developed in the context of the RETINA EUROSTARS project [10]. This library, built on top of Jailhouse's mechanism, provides an API based on the AUTOSAR COM standard [11], easing the development of the applications and the integration with other components.

In particular, the library has been integrated with an AUTOSAR Run-Time Environment (RTE) generator for ERIKA Enterprise. This RTE Generator, developed in the HERCULES project, runs on Eclipse and it is based on the Acceleo [12] and ARTop [13] technologies.

### D. *Linux OS*

Although safety-critical activities are handled by the ERIKA Enterprise RTOS, nevertheless we have improved the standard Linux kernel for having better timing performance. The first step has been the integration of the PREEMPT_RT patch [4]. Then, we have improved the SCHED_DEADLINE scheduler [14]. SCHED_DEADLINE is a real-time CPU scheduler available by default in any standard Linux kernel since release 3.14. Unlike other CPU schedulers, it allows to specify the amount of CPU time reserved to a real-time task with a per-task timing granularity. In the context of HERCULES, we have improved the SCHED_DEADLINE policy by adding two new features [15]:

- The possibility for a real-time task to reclaim the CPU bandwidth unused by other real-time tasks.
- The integration with ARM's CPU frequency scaling mechanism for reducing the energy consumption whenever the real-time tasks do not fully utilize the CPU.

Both features, deeply described in [15], have been integrated into the official Linux kernel since releases 4.13 and 4.16, respectively.

## III. Predictability in Memory Hierarchies

### A. Introduction

*a) Memory hierarchy contention:* The most prominent architectural weaknesses menacing predictable performance of the latest multi-core system-on-chip platforms, such as the HERCULES duo (NVIDIA TX2 and Xilinx ZCU102), are located in the memory hierarchy. Firstly, many components are shared — the number of cores and accelerators grows linearly with the degree of contention they generate on the memory controller and on the large Last Level Cache (LLC) subsystems. Secondly, hardware support to isolation is missing — last-level caches, although getting larger and deeper every year, lack support for explicit locking, or partitioning; DRAM controllers, on the other hand, are not equipped with any expensive programmability features. As a consequence, the system contention is left free to grow, and so is the detriment to the worst-case performance of real-time tasks.

*b) Hypervised memory management:* In HERCULES, we thus identified three specific issues, and solved them at a software level by implementing real-time extensions to the Jailhouse hypervisor. The solution enjoys simplicity, reduced overhead, programmability, and compatibility to legacy systems.

1) Contention to cluster-shared last-level cache is prevented by the *cache colouring support*, which partitions LLC for host OSs or bare metal applications by leveraging ARMv8 virtual extensions to properly setup address translation tables.
2) Self-eviction of useful lines in caches with pseudo-random replacement is avoided by *invalidation-driven allocation*, which consists, before fetching a number of new data, in simply marking as invalid a corresponding number of cache location, which are then deterministically selected for eviction.
3) Contention to shared RAM is tamed by enforcing the *Predictable Execution Model (PREM)*, which imposes to different hosted OSs or bare metal applications exclusive, thus high-speed and high-predictable, access to the central memory. When this is hardly applicable, we apply also *MemGuarding*, which limits the available bandwidth to a given hypervisor host.

The first two key techniques are the state-of-the-art [16], [17] technologies, whose industrial application has been already successfully tested on ZCU102 [18]. The comprehensive approach is validated against empirical evidence whose configuration is presented in the very next subsection. Such setup will be first to highlight the contention problem, and then to gradually introduce the different key ingredients that compose the HERCULES solution.

### B. Experimental Setup

We selected two task applications, a real-world example and an optimal corner-case.

- *GEMM.* Generalised Matrix Multiplication, as implemented in the PolyBench suite v4.2 [19].
- *Synth.* A naive implementation of the function which, given a 4 MiB array $A$ of 64 B data (i.e. a cache line — the atomic cache entity), it computes $a+|A|^2$, where $|A|$ is the length of $A$, for each $a \in A$, by iterated unitary incrementation over the whole length of $A$.

The applications are implemented as ERIKA tasks, to be independently run as a Jailhouse host. The host is allocated on the first (i.e. `0`) A57 core of the TX2 platform, whose core complex is equipped with a 2 MiB level 2 cache.

### C. Interference

To understand how dramatic the impact of memory interference can be, we have run the memory read latency micro-benchmark (lat_mem_rd) of the LMBench suite [20]. We have compared its isolated execution, both with sequential and random access pattern, with the *interference setup* where:

- 4 bare-metal applications stress the whole memory hierarchy by incessantly `memcpy`-ing a 4 MiB array each;
- 1 minimal Linux OS, hosting a similar application, which in addition also repeatedly spawns a CUDA `memSet` kernel, adding further load the SDRAM subsystem.

Results are plotted in the leftmost column of Figure 2 and are coherent with lat_mem_rd.

Figure 3 shows how the memory read access time grows with respect to the total depth. Under the level 1 data cache size (32 KiB on A57, 64 KiB on Denver cores), latency is stable under interference, since the cache is dedicated to the core. Over the level 2 size (2 MiB), instead, the slowdown is around 5-10x, because the whole hierarchy is under contention and every component is near to congestion. In between, the variation varies with the depth growth, although not always linearly.

### D. Cache Colouring

Page colouring is a software technique that implements cache partitioning by exploiting disjointness traits of the cache mapping function, the one that associates a cache line to any given memory address. Whilst in general it requires manual implementation at application- or OS-level, we propose an almost transparent hypervisor support. The benefits are easily appreciable on the execution time for both GEMM and Synth — we configured a variation of the experiment configuration where a 512 KiB cache partition is dedicated to the core under test), and to each other A57 running interference. Results in the leftmost plot column of Figure 3 show that the cache limitation caused by colouring is usually negligible (as in GEMM), and that cache partitioning is effective when the application heavily exploits it (not as in Synth).

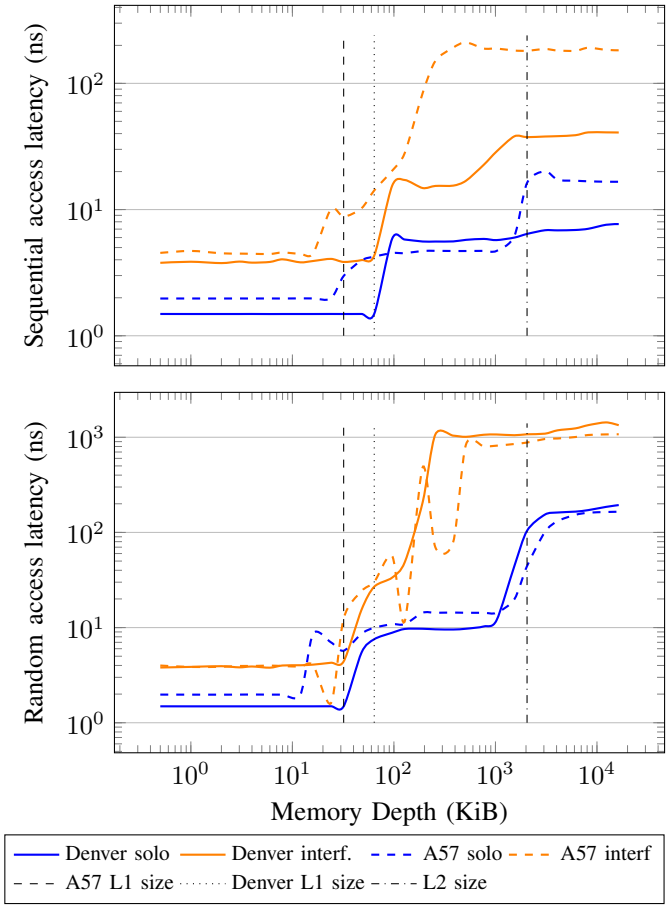


Fig. 2. Memory read latency micro-benchmark results (logscale).

### E. PRedictable Execution Model

The PRedictable Execution Model (PREM) [21] approach forces real-time tasks to be structured alternating two kind of operating modes named *intervals*:

- *Computation*: Execution performs no access to central DRAM, hence relying on dedicated caches (e.g. coloured).
- *Memory*: Cache writeback to DRAM is operated, followed by a cache prefetch for the next computation phase.

The key point is that memory phases of different tasks running on a multi-core system are required to never overlap, to eliminate any competition in DRAM access that would cause the bandwidth to be shared. The problem of contention on DRAM, is thus reduced to the synchronisation between tasks willing to start a memory interval.

We extended Jailhouse with a *memex* (memory mutex), so that such transition is subject to having acquired the memex, like a 'lock' operation. Conversely, the transition from the memory to the computation interval unlocks the memex for other tasks. Moreover, HERCULES project partners have built an optimiser plugin for the LLVM compiler [22] enabling reasonably simple C/C++ programs to be transformed so to achieve PREM-compliance, wherever possible (an alternative approach is discussed in next subsection). We exposed the Jailhouse PREM API through the OS up to application level, so that the compiler is able to insert the appropriate memex (un-)lock calls.

We repeated the previous experiments this time using: the automatically PREM-optimised version of GEMM, and a manually PREM-ised version of Synth. We observe (see central column of fig. 3) significant improvement (2x speedup on average) in the optimistic setup of Synth, and a negligible variation to GEMM.

Also the interference applications are replaced by a PREM-compliant variant, where the length of the `memcpy` invocations is limited to 512 KiB, i.e. the available cache partition. We see that this significantly reduces the memory interference impact on Synth, but it degrades GEMM performance because the lock acquisition delay may be greater than the unrestricted effect of memory stress (fig. 3).

### F. MemGuard

PREM intervals require non-preemptible execution and forbid foreign calls, e.g. syscalls. Moreover, memory operations within a given program may not be reordered and grouped in a sufficiently large sequence that would justify the overhead caused by synchronisation. For the sake of flexibility, we propose a PREM relaxation by allowing a third kind of interval without the strict dichotomy between memory and computation. We call *compatible* such an interval.

A compatible interval is either memexed, as memory intervals are, or is protected by MemGuard [23], a software technique developed within HERCULES which limits the memory bandwidth attainable by a core in compatibility mode. A periodic interrupt service routine is executed to grant memory usage budget, i.e. an allowance of L2 cache miss, to hypervisor hosts. If an host exhausts its budget, then its CPU cores are throttled until the next replenishment.

We implemented a variation of the previously described memory interference setup where A57 cores' RAM bandwidth is quite strictly limited to allow only 10 KiB/s with a 1 ms refill period. Baseline-relative measured latency drops down slightly over 1.5x slowdown fig. 3, considerably better than the unrestricted version.

### G. Integrated Results

We are now ready to combine the separately-introduced ingredients, and show the effectiveness of the integrated HERCULES platform. The rightmost boxplots column of Figure 3 collects results intermediate and aggregate configurations combine colouring with PREM, MemGuard and/or the different kind of interference. The HERCULES framework enable system isolation — memory interference overhead is tamed so to produce less than 0.3x overhead in almost all the cases.

## IV. CONCLUSIONS

We have presented the HERCULES software architecture, that allows to meet industry's mixed-criticality needs using novel ARM multi-core platforms. The architecture particularly targets automotive applications, running the ERIKA

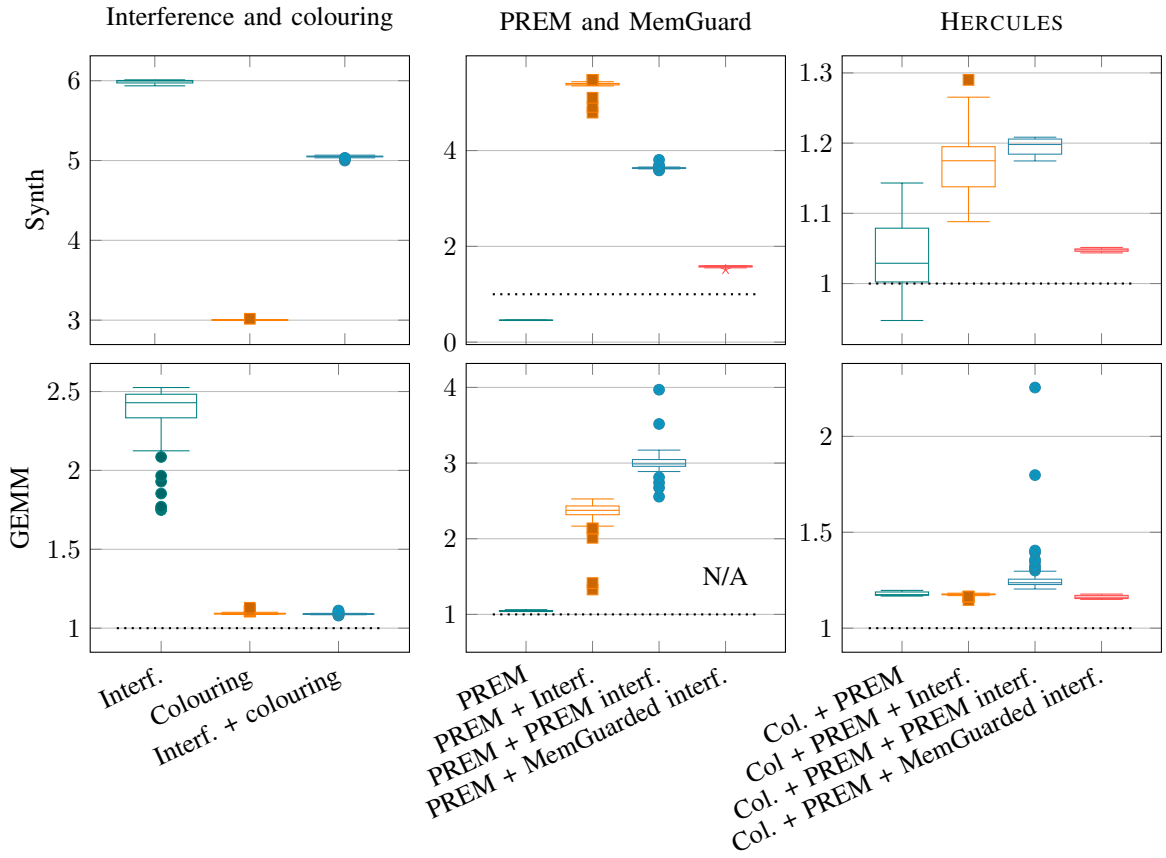


Fig. 3. Benchmark results. Execution time with ns precision, normalised to the average baseline (dotted).

AUTOSAR RTOS alongside the Linux OS. An RTE generator allows real-time communications between the two operating systems. Finally, a set of tests have shown the effectiveness of the proposed approaches for reducing (or avoiding) the interference on shared hardware resources.

## REFERENCES


[1] RTLinux. [Online]. Available: https://en.wikipedia.org/wiki/RTLinux

[2] RTAI. [Online]. Available: https://www.rtai.org

[3] Xenomai. [Online]. Available: https://xenomai.org

[4] PREEMPT_RT. [Online]. Available: https://wiki.linuxfoundation.org/realtime

[5] Siemens. Jailhouse hypervisor. [Online]. Available: https://github.com/siemens/jailhouse

[6] NVIDIA. Jetson TX2. [Online]. Available: https://github.com/siemens/jailhouse

[7] Xilinx. ZCU102 Evaluation kit. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

[8] Evidence Srl. ERIKA Enterprise. [Online]. Available: http://www.erika-enterprise.com

[9] Qemu processor emulator. [Online]. Available: https://www.qemu.org

[10] RETINA ECSEL project. [Online]. Available: http://retinaproject.eu

[11] Autosar standard. [Online]. Available: http://www.autosar.org

[12] Acceleo. [Online]. Available: https://www.eclipse.org/acceleo

[13] Artop. [Online]. Available: https://www.artop.org

[14] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2335

[15] C. Scordino, L. Abeni, and J. Lelli, "Energy-aware real-time scheduling in the linux kernel," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: ACM, 2018, pp. 601–608. [Online]. Available: http://doi.acm.org/10.1145/3167132.3167198

[16] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A Real-Time Scratchpad-Centric OS for Multi-Core Embedded Systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.

[17] 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 *25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS, 2019)*.

[18] G. Corradi, B. Klingler, E. Puillet, P. Schillinger, M. Bertogna, M. Solieri, and L. Miccio, "Delivering real time and determinism of ZYNQ Ultrascale+ A53 clusters with Coloured Lockdown and Jailhouse hypervisor," February 2019.

[19] L.-N. Pouchet. PolyBench/C – the Polyhedral Benchmark suite. [Online]. Available: http://polybench.sourceforge.net

[20] L. McVoy and C. Staelin, "Lmbench - tools for performance analysis," 2005, version 2 http://www.bitmover.com/lmbench/.

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

[22] J. Matejka, B. Forsberg, M. Sojka, P. Sucha, L. Benini, A. Marongiu, and Z. Hanzalek, "Combining prem compilation and static scheduling for high-performance and predictable mpsoc execution," *Parallel Computing*, 2018.

[23] 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 *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*. IEEE, 2013, pp. 55–64.