Taming Edge Computing for Hard Real-Time Advanced Control of Mechatronic Systems

Taming edge computing for hard real-time advanced control of mechatronic systems

Luca Orciari, Davide Raggini, and Andrea Tilli.

***Abstract*— In novel mechatronics enabled by smart structures and materials, servomechanisms are becoming increasingly complex, requiring computationally-intensive advanced control algorithms and diagnostic tools to fully exploit their potential. This calls for a significant increase in computational power while guaranteeing hard real-time features. In this work, we propose to address such an issue by adopting recently-emerged edge-computing solutions exploiting low-cost multicore that combine microcontrollers and microprocessors to boost the computational capability. However, such platforms are usually endowed with non-real-time software infrastructure, assigning a dominant role to microprocessors and leading to large overheads and unpredictability. Therefore, to tame them for hard real-time, we first lighten the infrastructure to enable one or more microprocessors to handle computations with minimal overhead and jitter. Then, we designate a microcontroller as the platform master of time and tasks, off-loading the heavy computations to the "relieved" microprocessor cores, acting now as computational slaves. We assess the potentials of this approach with a basic test using a demanding control algorithm as a benchmark, choosing the STM32MP157 as the reference platform and using the Jailhouse hypervisor to adapt one of its microprocessor cores for hard real-time tasks.**

***Index Terms*— Automatic control, Benchmark testing, Edge computing, Embedded software, Mechatronics, Microprocessors, Platform virtualization, Real-time systems, Servosystems.**

I. Introduction

In the field of mechatronics, servomechanisms are pivotal components. Within high-performance motion architectures, they function as mechatronic chains, combining electric drives (i.e. the union of control and power electronics with electric motors) with specialized mechanisms designed to move along periodic or asynchronous trajectories with one or multiple degrees of freedom.

In this domain, embedded microcontroller units (MCUs), tailored explicitly for operating electric drives, are commonly employed [1]. These MCUs feature high-resolution timers and ADC units, to provide fast control actions. In addition, they operate within hard real-time (HRT) constraints, related to periodic tasks, whose periods span from tens to hundreds of μs, with no tolerance on expected response instants to drive power electronics properly. However, such MCUs are, more often than not, single-core units with limited computational capabilities. Hence, they predominantly lead to simple control solutions complemented at most with basic feedforward, self-tuning, and loop-shaping functions.

Nowadays, the complexity of servomechanisms is increasing [2], which means that advanced control methods are necessary to turn their potential in true performance. Additionally, it is crucial to identify faults, wear, and inefficiencies in more intricate systems. As a result, diagnostic and prognostic tools have become almost mandatory in today's industry. Features like on-the-edge condition monitoring can lead to substantial cost savings, extending equipment lifespans and reducing downtimes by promptly identifying anomalies. However, the MCUs currently used in the mechatronics field lack the computational power required to implement these essential functionalities. As a result, to effectively integrate these features, bounded mainly by HRT constraints, there is a need for greater computational capabilities, "at the edge", or better "at the embedded edge", which requires to keep the overall cost and footprint as low as possible.

In the world of Internet of Things (IoT), innovative platforms have emerged bringing together different kinds of processor cores, such as Microprocessor Units (MPUs) and MCUs. These heterogeneous computing platforms offer an opportunity to fulfill the increasing need for computational power in advanced servomechanisms by combining the real-time capabilities of MCUs with the performance metrics akin to those of low-power PCs thanks to MPUs. Additionally, the MCU greatly facilitates interfacing with sensors and actuators. However, there is one significant drawback: the computational power of MPUs often cannot be fully utilized to meet the requirements of servomechanisms. This issue arises because MPUs on these platforms typically use Operating Systems (OS) like Linux, which are unsuitable for periodic tasks with HRT constraints like those requested in advanced mechatronics with significant computations and cycle times of few tens of μs. Additionally, in standard software setups, MCUs have a dominant role over the MPUs, making their collaboration in addressing such tasks very complex. As a result, effectively leveraging the power of these platforms for advanced servomechanisms remains a challenge.

Given all the above considerations, the main objectives of this work are:

- Introduce a novel paradigm for the management of IoT-oriented heterogeneous platforms that enables the MPU to handle HRT tasks efficiently. This approach aims to fully exploit the MPU's computational power to support the MCU in HRT advanced mechatronic applications

Paper submitted for review 11 October 2023.

The authors are with the Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, 40136 Bologna, Italy (e-mail: luca.orciari3@unibo.it; davide.raggini2@ unibo.it; andrea.tilli@unibo.it).

properly (i.e. with 10-100μs task periods)
- Maintaining cost-efficiency, focusing on platforms with low cost and low energy consumption, harnessing their full potential.

As for the first point, unlike the traditional software configuration, where the MCU is subordinate to the MPU, we suggest giving the MCU a more prominent role as the controller of time and tasks, directing one or more MPU cores. These MPU cores could efficiently handle computations for demaning HRT tasks with a proper low overhead software infrastructure, complementing the MCU's capabilities. In particular, we are targeting heterogeneous computing platforms with at least two MPU cores and one MCU. With multiple MPU cores, it becomes possible to maintain at least one core hosting a conventional operating system with all its functionalities. It is worth noting from the very beginning that one major challenge in our approach is minimizing the interference between the MPU cores to reduce the overhead and enhance predictability in the MPU assigned to HRT tasks.

Moving to the second point, oriented to cost-efficiency, we are considering heterogeneous computing platforms like STM32MP157 [3] from STMicroelectronics and I.MX 7 [4] by NXP Semiconductors. These platforms boast dual-core MPUs based on the ARMv7-A architecture, and they show cost-effectiveness and low energy requirements.

In this context, crucial preliminary steps to apply the above approach to a chosen platform include:
- Identifying the essential components to enable the MPU cores to handle HRT tasks with minimal system overhead.
- Assessing the extent of time saved when transitioning HRT tasks from the MCU to the MPU equipped with the identified software infrastructure.
- Evaluating the magnitude of jitter, as the worst-case time savings must be considered for HRT applications [5].

This work will address these essential preliminary benchmarking steps. Specifically, we will consider as a starting example:
- The STM32MP157, as the platform of choice, selected for its affordability, robust performance, user-friendly software ecosystem, and features relevant to our proposed paradigm.
- Jailhouse, a lightweight hypervisor chosen for its ability to enable the MPU core to handle HRT tasks by splitting the system resources.
- A benchmark rooted in an advanced control application to fully grasp the benefits of leveraging such platforms for advanced servomechanisms.

A. Related works

To the best of our knowledge, no existing work has addressed our specific objective regarding the proposed paradigm for heterogeneous computing platforms applied to modern mechatronics. Nonetheless, significant efforts have been made to achieve real-time functionalities on MPU cores within open-source commercial-off-the-shelf (COTS) platforms to concurrently execute both non-critical activities and real-time control tasks on the same platform. Without the ambition to be exhaustive, we present some threads carried out in this framework.

Several hypervisors have been thoroughly analyzed to enhance system predictability in an real-time context. Notable studies include those focusing on automotive applications, such as [6], and performance measurements specifically on embedded ARM processors, such as [7] and [8]. These studies have used hypervisors like Jailhouse, Xen, and BAO. [9] introduces the BlueVisor hypervisor, a scalable real-time hardware hypervisor for many-core embedded systems, showcasing its design and real-time capabilities. [10] presents Sypher an embedded hypervisor optimized for mixed-criticality systems, highlighting its real-time performance and hierarchical resource isolation, and its advancement over traditional hypervisors through improved VM scalability and efficiency using novel virtualization techniques for secure and efficient mixed-criticality management. In the context of Industry 4.0, [11] proposes RunPHI, an application that integrates partitioning hypervisors with OS-level orchestration systems for improved management of distributed resources. The potential of Xen and KVM as real-time hypervisors is evaluated in [8].

Regarding memory predictability and memory concerns for multi-core embedded systems. In [12], the authors address the predictability issues of multi-core embedded systems arising from shared memory access contention and propose a software framework to enhance memory access determinism. Meanwhile, [13] offers a comprehensive evaluation of shared memory's impact in multi-core mixed-criticality real-time applications on platforms with ARMv8-A and ARMv7-R processors.

Several studies have focused on enabling platforms to perform mixed-criticality tasks in the PC-based industrial automation domain, with hypervisors being the primary enabling technology. In [14], Cinque et al. offer an in-depth review of virtualization trends and challenges in the industry. In [15], Queiroz et al. assess general-purpose hypervisors in real-time control systems, while [16] discusses the application of embedded virtualization for industrial edge computing. Additionally, [17] and [18] explore the impact of virtualization, focusing on flexibility, scalability, and the development of modular platforms.

To conclude this subsection, it is worth recalling the distance between our proposal and the above literature. This work addresses a specific strategy to tackle the enhanced computational requirements of modern mechatronic servomechanisms by employing low-cost MCU-MPU heterogeneous platforms. It aims to boost the MCU's performance by exploiting the MPU, without completely replacing the MCU. Thus, our target is very specific, focusing on a streamlined architecture. In contrast, the above-mentioned works consider a different scenario. On one hand, they consider either no control applications or industrial control applications (like PLC-based machine/plant control) having a scope and a time scale far from the one characterizing our mechatronic servomechanism domain. On the other hand, the above works revolve around platforms exploiting only MPUs (usually high-performance MPUs in the industrial PC domain) without introducing MCUs. Owing to the above characteristics, direct comparison of our proposal

with the above solutions looks inappropriate and potentially misleading.

B. Paper Organization

To provide clarity on the structure of this work, we offer a brief overview of its various sections:

- Section II: This section delves into the key features of the STM32MP157 platform, emphasizing the pivotal aspects concerning the proposed software infrastructure.
- Section III: We discuss the different methods to enable the proposed paradigm, exploring various methodologies enabling the MPU core to handle HRT tasks.
- Section IV: We provide an overview of the software architecture, outlining the functions and features that can be implemented on the platform.
- Section V: Representing the core of our work, we evaluate the viability of the proposed method. We focus on assessing its potential in processing complex control algorithms, optimizing computation time, and maintaining minimal jitter mandatory for HRT tasks.
- Section VI: It remarks the results of our work and highlight possible future direction.

II. System Architecture

A. Hardware description

As previously noted, the STM32MP157 was selected as the heterogeneous computing platform of choice to implement and assess the validity of our proposed paradigm. This platform is ideal for our purposes since it features all the necessary hardware components while preserving cost-effectiveness. Moreover, it comes with a rich, well-documented, and easy-to-use software ecosystem.
Key features of the STM32MP157 platform are reported, highlighting their exploitation in our new architecture:

- **MPU: Dual-Core Cortex-A7:** Operating at a clock speed of 650MHz and based on the ARMv7-A architecture, the inclusion of at least two cores satisfies one of the minimal requirements for our approach. While one core will be tasked with high-level, not time-sensitive operations via Linux, the other, given the proper software infrastructure, will serve as a computational slave to the MCU for HRT tasks.
- **MCU: Single-Core Cortex-M4:** This core, operating at a speed of 200MHz, is part of the ARMv7E-M family. In our architecture, it will manage all HRT processes, establishing its authority as the master over the Cortex-A7.
- **IPCC (Inter-Processor Communication Controller):** The IPCC menages data transfers between all processors cores. Employing a non-blocking signaling technique ensures prompt data transfers, exploiting shared memory buffers. Both Cortex-M4 and Cortex-A7 cores can access these memory buffers located in the MCU SRAM.
- **Timer:** This peripheral, synchronized with the Cortex-M4's frequency and featuring a 32-bit resolution will be essential to manage HRT tasks in our architecture. Besides, it ensures precise measurement of execution time for our evaluation benchmarks.
- **Additional Peripherals:** These devices include Ethernet interface, a USB hub, a serial interface, and other components. Beside providing additional features, they can also create disruptions when conducting benchmark evaluations, which helps determine appropriate core isolation.

Considering these characteristics, the STM32MP157 emerges as a viable choice for the abovementioned purposes. In addition, this platform is equipped with a viable and user-friendly ecosystem, making development smoother. Nevertheless, while we selected the STM32MP157 to validate our proposed paradigm, it is important to highlight that the platform serves only as an example. The architectural framework and operational insights provided here, although demonstrated through the STM32MP157, are intended to be adaptable to other heterogeneous computing platforms that meet the essential criteria for implementing efficient edge computing solutions.

III. SW modules for HRT and efficient computations

A primary target for the proposed orchestration is to optimize the computational capabilities of one of the available A7 cores, making it:

- Highly efficient, ensuring minimal system overhead.
- Predictable, hence maintaining a low jitter for HRT purposes.

To this end, we explore available software modules and strategies, discussing their advantages and drawbacks:

- **Real-time Linux:** This version of Linux has been improved with patches to make the kernel more predictable. These modifications make the kernel fully preemptible and include a real-time CPU scheduler, which was introduced in release 3.14 [19]. However, despite these enhancements, real-time Linux may only sometimes meet performance expectations, especially when dealing with computationally demanding control tasks commonly found in modern automation. In particular with periodic HRT tasks with cycle times of few μs [19].
- **Bare-metal Programming:** This approach entails creating software that directly interacts with the hardware, bypassing the requirement for an operating system. While it provides unparalleled control over system resources, implementing this method can be complex and could require a significant time investment to fully utilize all available system resources.
- **Hypervisor:** Partitioning hypervisors have emerged as particularly promising solutions, enjoying considerable favor in the automotive sector. These hypervisors facilitate the creation of isolated partitions where resources are statically allocated to each virtual machine (VM). Consequently, every VM is empowered to operate its own distinct operating system or a bespoke real-time environment. With robust isolation, interference from other cores is minimized, promoting improved predictability. Hence providing the ability to meet HRT constraints.

After considering the above options, we decided to go for the hypervisor approach. This choice strikes a balance by offering the efficiency of bare-metal programming while providing an organizational framework that simplifies the development process.

To find a suitable hypervisor solution, we evaluated several type-1 hypervisors, also known as "bare metal" hypervisors. These hypervisors operate directly on the host hardware without relying on an operating system. Our evaluations focused on three options: Xen (its dom0less variant), Jailhouse, and Bao. We chose these options because they are widely adopted with an active community and open source, making them promising candidates for our needs.

1) **Xen:** Xen is a hypervisor that can manage multiple VMs on a single physical host, ensuring strong isolation and optimal performance. In its traditional setup, Xen employs a VM called "Dom0" to handle administrative tasks. This special domain has exclusive access to the hardware. Takes charge of activities such as launching and managing unprivileged VMs (DomU), interfacing with hardware components, and managing specific drivers. However, there's now a "dom0less" mode in Xen where it can operate without relying on this domain [20]. This mode is particularly useful for embedded systems and automotive applications where minimizing size and complexity is crucial.
2) **Bao:** Bao is a hypervisor specifically designed for embedded systems [21]. Its main goal is to ensure isolation between partitions while keeping the performance impact as low as possible. It adopts a static configuration approach, where resources are allocated to partitions at compile time. Bao's emphasis on efficiency and security makes it ideal for constrained environments where performance and safety are paramount.
3) **Jailhouse:** Jailhouse is a partitioning hypervisor designed for static partitioning of a system's resources [22]. It aims to create isolated environments, known as cells, on multicore systems without the need for a full-fledged traditional hypervisor stack. Instead of virtualizing hardware, Jailhouse focuses on separating workloads running on the same physical machine. One distinguishing feature is its simplicity: Jailhouse eschews complex functionalities like overcommitment, aiming to reduce overhead.

We chose the Jailhouse hypervisor because it is simple and efficient. We ruled out two other options for the following reasons:

- **Bao**: It has similarities to Jailhouse, such as its lightweight nature and focus on resource partitioning with minimal overhead. Additionally, Bao offers flexibility by not requiring a Linux-based root cell, which can be beneficial for platforms with fewer CPU cores. Anyhow, it was excluded due to its ongoing development for ARMv7 architecture support.
- **Xen**: Despite being compatible with the ARMv7 architecture, Xen did not have support for our specific board. Integrating Xen into the ST Microelectronics ecosystem would have required considerable effort.

The motivations mentioned above are closely tied to the heterogenous platform chosen to demonstrate our proposed paradigm. For heterogeneous platforms featuring different architectures, alternative hypervisors can be a viable solution.

IV. The adopted SW architecture

Additional details about the proposed software architecture and its deployment on the STM32MP157 platform are outlined here. As previously mentioned, according to our proposed paradigm:

- The Cortex-M4 serves as the central unit, overseeing all real-time operations as the master.
- One Cortex-A7 core, which hosts the Jailhouse root cell running Linux, manages tasks that aren't time-sensitive.
- The other Cortex-A7 core, operating the Jailhouse secondary cell, assists the MCU with HRT tasks. It runs pseudo-bare-metal code with minimal infrastructure given by the hypervisor. Hence, its primary role is to reduce the computational burden on the Cortex-M4, aiding it, when triggered, to deal with computationally heavy tasks. It is worth noting that in general is not possible to shift HRT Control and Diagnosis tasks to MPU completely, since sensor and actuator interfacing is carried out by the master MCU which fits such task best.

Moreover, the platform under consideration is conceived to be integrated into an advanced electric drive for innovative mechatronic chains. Therefore, the following modules are foreseen for both the MCU and the Linux-mastered MPU to harness the system's full capabilities:

- MCU: Besides HRT control tasks based on interfacing with secondary cell MPU. It will oversee fieldbus communications such as EtherCAT [23] and manage sensor acquisition and power actuation.
- MPU featuring Linux: This will handle data management, including remote data exchange, a human-machine interface, and a command-line interface.

It is also important to highlight the important role of the IPCC introduced in Section II. This controller oversees communications between all the processors. By leveraging shared memory resources, the IPCC ensures rapid and precise data transfers between cores. Hence, allows for the exchange of data, such as control inputs and outputs, as well as sensor data, between cores, thus enabling the implementation of the proposed methodology.

In summary, the proposed architecture is depicted in Fig.1.

V. Test definition and performance evaluation

Considering the proposed paradigm, we aim to evaluate the overall computational time we can preserve by transitioning resource-intensive control or diagnostic tasks from the MCU to the MPU. Since these tasks are subjected to HRT constraints, paying particular attention to the maximum elaboration time is crucial, given that when dealing with this kind of application, it is mandatory to meet deadlines even in the worst-case scenario. Moreover, this assessment should also address data management time as the control algorithms elaboration is split

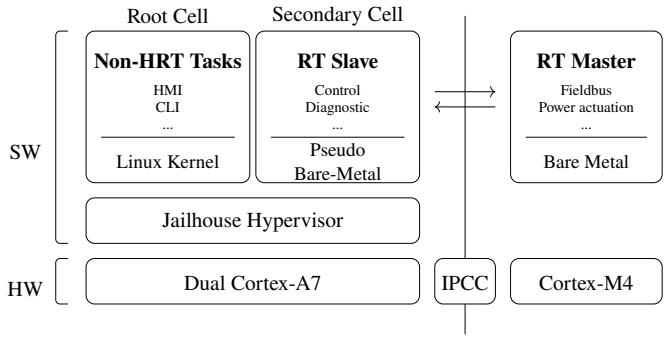

Fig. 1: Overall software architecture.

among multiple cores, and diagnostic tools require a large amount of data to be transferred.

It is worth underlining that, according to the above targets, we do not need to rely on a standard testing suite, as we are not aiming to rate the performance of our methodology with the selected platform as a whole in comparison to other multicore and SW platforms. Moreover, we intend to compare the performance between the MCU and MPU with a benchmark that is deeply rooted in control applications of advanced servomechanisms to evaluate control computation time and control and diagnostic data transfer time. Therefore, standard test suites like MiBench [24], which cover a wide set of general applications, are not suited to our specific domain. Furthermore, other suites like Cilictest[1], which offer insights into real-time scheduler latency and responsiveness, do not test what is of our interest at this stage of development.

Consequently, we propose two kinds of specific tests:

- The first test focuses on determining the time required to transfer data between the MCU and isolated MPU back and forth. It is crucial to assess the speed of data transmission for batches of different sizes as smaller chunks are significant for control application purposes, while larger ones are more relevant for condition monitoring and diagnostic tasks.
- The second test considers a benchmark to assess the overall computational time needed to perform complex elaborations. We anticipate that moving the calculation from the MCU to the MPU will result in a reduction in average computational time. Nevertheless, it is crucial to assess the jitter. MPUs embodied in a multicore structure show a complex and dynamic computational architecture, which might lead to substantial variance in the execution time of a given code chunk. As reported above, we use a servomechanism control-specific code (described later) to run such a test. In order to stimulate the adoption of this kind of benchmarking for similar configurations on other platforms, we provide our code in the supplementary material.

In the following, we report the detailed arrangement of the proposed tests on the considered platform and the related results, along with comments and considerations.

[1]https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest

A. Test descriptions

As anticipated, the first test consists of a loopback test. Its aim was to properly assess the overhead introduced by transmitting and retrieving data to the isolated Cortex-A7 core.

More specifically, the test entails the following steps:

1) The *Cortex-M4* generates a random data package of fixed size.
2) This data package is copied to the shared memory with the *Cortex-A7*.
3) A notification is sent via the *IPCC* to indicate the presence of the data package.
4) The isolated bare-metal *Cortex-A7* core then copies this data package to its local memory.
5) Subsequently, the data package is sent back through the shared memory.
6) A notification is sent again via the *IPCC* to confirm the return of the data package.
7) Go back to step 1).

Our performance metric evaluates the duration taken for the data package to make this round trip. We performed the test with various data package sizes, starting from 0 bytes, where we only assessed the overhead introduced by the IPCC notification mechanism up to 1024 bytes. Data packages on the lower-size end are essential to grasp how much time is lost when offloading part of the Cortex-M4's computations to the Cortex-A7. Larger data packages are meaningful to know how much time is required to transmit larger data chunks present in diagnostic and prognostic applications. To establish a meaningful benchmark for comparison, we also conducted the same test without Jailhouse, opting instead for the real-time Linux patches discussed in Section II.

We augmented the loopback test by adding an interference application to evaluate the core's isolation better. Throughout the entire duration of the test, the Cortex-A7 hosting Linux was kept occupied. Specifically, it continuously transmitted a file located in the SD card memory of the board to a pen drive connected to a PC via SSH protocol. By doing this, we thoroughly tested the core isolation and how the jitter behaves in different operating conditions, particularly considering the utilization of various peripherals and continual access to the system memory.

Following the loopback test, we next focused on evaluating the computational speed-up potential obtained by offloading calculations to the MPU. In this effort, we focused on a computationally intensive controller as our benchmark. This choice was motivated by the assumption that a resource-heavy controller would more clearly highlight the benefits of leveraging the Cortex-A7. Specifically, we favored a controller based on the *internal model principle* [25] [26]. In a nutshell, the internal model principle posits that to achieve perfect tracking or disturbance rejection in a feedback control system (without effective feed forward actions), the controller must incorporate an internal representation (or model) of the external signal to be tracked or rejected and a proper asymptotically stabilizing unit. By exploiting such "internal model", after a suitable convergence transient, the system can generate a proper control action for tracking and disturbance rejection

without any non-null tracking error input (see [25], [26] for details). Internal models usually adopted in mechatronic applications are a composition of integrators and oscillators to compensate for constant and harmonic signals. In our test, we considered a control algorithm endowed with 20 harmonics and an integrator (i.e the controller state space model has 41 state components). The high number of harmonics results in significant computational requirements, thereby qualifying it as an apt choice for our assessment.

The above mentioned control algorithm is part of a larger control architecture for an electric drive moving a nonlinear complex mechanism, and it was designed with the assistance of MATLAB and simulated through MATLAB Simulink [27]. We used Simulink's Embedded Coder toolbox to generate the C code required to run the program on our platform, ensuring accuracy and optimization. The controller was implemented in both fixed-point and floating-point precision. The fixed-point implementation was used to have a fair comparison with the Cortex-M4 since the latter features an unoptimized Floating-Point Unit (FPU). In contrast, the floating-point format was used to highlight the benefits of leveraging the FPU of the Cortex-A7. We used a sample set of inputs (motor position reference and measurement) and corresponding outputs obtained from a Simulink simulation to benchmark the system and provide realistic inputs and outputs for the controller. As anticipated, the overall code and data set will be released in the supplementary material. The test comprises the following steps:

1) The *Cortex-M4* sends the reference position and motor position to the isolated bare-metal *Cortex-A7* through the IPCC and shared memory regions.
2) The *Cortex-A7* uses these inputs to compute the controller output.
3) The computed output is then sent back through the shared memory and notified via the IPCC.
4) The resulting output is checked to ensure it matches the simulated values obtained from the Simulink simulation.
5) Return to step 1).

The interference application, introduced during the loopback test and running on the A7 core with Linux, was maintained active for the entire duration of this test to assess the computational speed-up obtained by exploiting the MPU under unfavorable operating conditions.

B. Test results and comments

We begin by presenting the results of the loopback test. We conducted 1 million iterations of the test for each data packet size. In Table I, we provide key statistics comparing the performance of the core isolated via the Jailhouse hypervisor, both with and without the interference application active, to that of a Linux system employing real-time patches without active interference.

From the results, we immediately note that the maximum execution time for the loopback aligns closely with the average execution time when using the core isolated through the *Jailhouse* hypervisor. This holds true whether the interference application is active or not, suggesting a reliable core isolation.

TABLE I: Loopback Test Results in μs

Size(Bytes)	Jailhouse			Jailhouse with interferance ON			Linux RT		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
0	0.91	1.07	1.01	0.92	1.18	1.01	0.91	2.48	1.01
1	1.20	2.41	1.25	1.21	1.76	1.25	1.20	2.71	1.26
8	2.83	4.28	2.90	2.84	4.23	2.90	2.83	4.5	2.91
16	2.91	5.09	3.05	2.91	5.09	3.05	2.91	5.09	3.05
32	4.81	7.21	4.97	4.81	7.08	4.96	4.81	7.30	4.96
64	8.51	10.47	8.76	8.52	10.73	8.75	8.50	10.90	8.75
128	16.01	18.52	16.44	16.01	18.87	16.39	15.99	69.42	16.39
256	30.87	33.26	31.71	30.97	36.98	31.59	30.91	119.56	31.59
512	60.95	63.73	62.28	60.95	73.40	62.02	60.81	132.20	61.86
1024	120.87	125.10	123.42	120.87	134.48	122.29	120.63	296.54	122.258

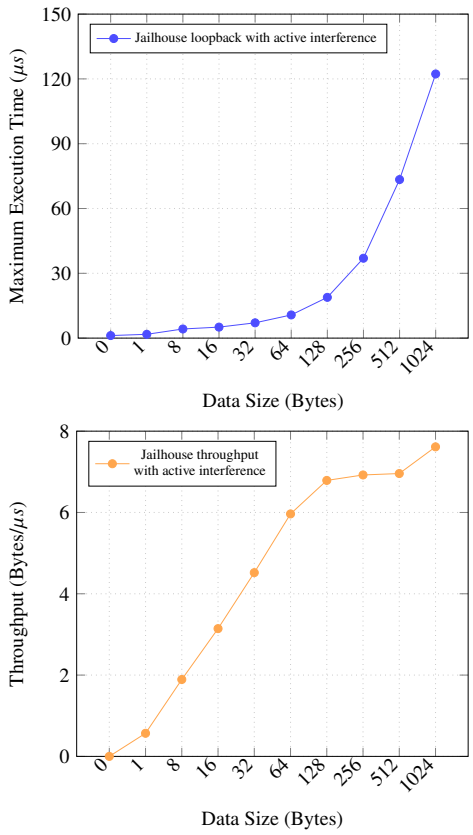

Fig. 2: Performance analysis when transferring data using Jailhouse

In contrast, while the mean execution time remains relatively stable when running the loopback test on the core with Linux using real-time patches, the maximum execution time is significantly higher. Considering we are addressing HRT tasks where deadlines must be met at each iteration, the maximum execution time becomes our primary concern. The test using real-time Linux was not repeated with the active interference application, as the results were already unpromising without it.

As observable from Fig.2, transmitting larger data packages yields improved performances in terms of throughput. This can be attributed to the constant overhead time associated with the notification mechanism, which remains unchanged regardless of the size of the data packet being transmitted. While this overhead plays a significant role when transmitting smaller data packets, it becomes increasingly negligible with larger packets.

After having validated the communication mechanism be-

tween cores and confirming that it introduces an acceptable overhead, we turned our attention to the second type of benchmark introduced. In this evaluation, we determine the potential computational speed-up obtained by offloading the calculation to the core isolated with Jailhouse. Hence, we compared the execution time of the previously described controller on the isolated core to that of the Cortex-M4. We performed the test using 2000 controller inputs obtained from the Simulink simulation looped for a total of 2 million iteration of the controller. We measured the execution time on the isolated core, including and excluding the communication time. While assessing the potential speed-up, it is essential to account for the communication time. However, we also wanted to benchmark the standalone performance of the application. We showcase the results in Fig.3, where we detail the minimum, maximum, mean, and median execution times.

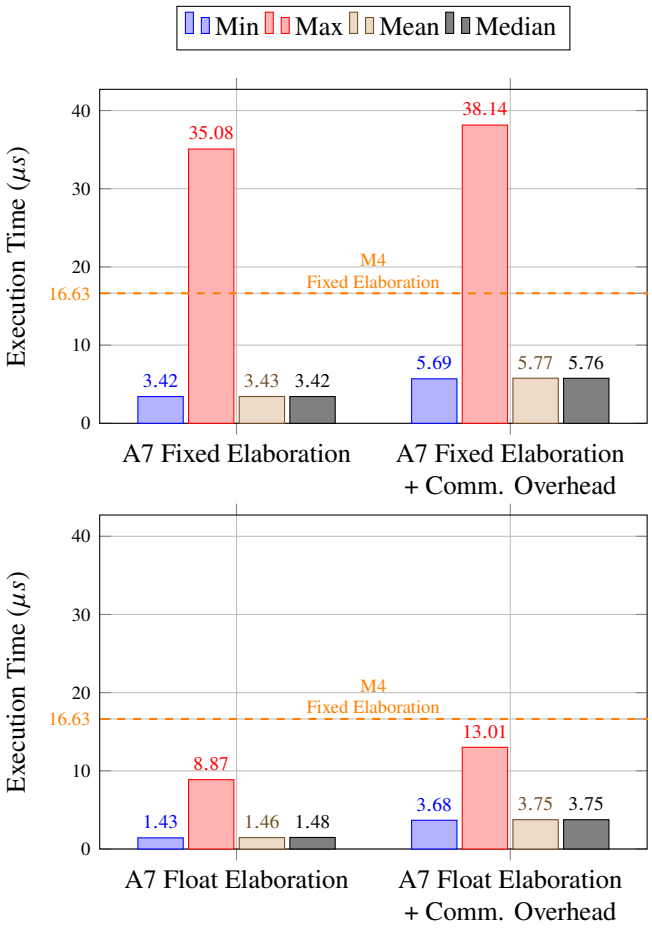

Fig. 3: Histograms representing execution times of the control algorithm

As expected, concerning the fixed-point precision implementation of the controller, we found promising results on the isolated core. It outperforms the M4, even when accounting for communication time overhead. However, we observed significant spikes in execution time. These are outliers, as evidenced by the low mean and median. However, despite their infrequency, such spikes are unacceptable in real-time applications where missing a deadline is unacceptable. Therefore, implementing this controller on the A7 would not provide a computational speed-up compared to the M4. Nevertheless, when looking at the execution time of the code in floating point precision on the A7, this offers a speed-up of approximately 21%, accounting for the worst case. We measure the speed-up against the M4 executing the controller code again in fixed-point precision, as the latter features only a basic FPU unit and, as anticipated, execution times would be at least 4 times higher. It is worth stressing that, the capability of the A7 to efficiently run floating point code will greatly impact design time for control applications since converting the latter in fixed-time precision with good precision requires a significant amount of time.

The spikes in the execution time are almost certainly memory-related. In particular, the Last Level Cache (LLC) is shared in this software configuration between the two A7 cores, leading to mutual eviction events. Next, we will explore two potential solutions to address this issue:

1) Partition the cache using cache partitioning techniques, such as cache coloring [28].
2) Disable the data cache and consequently the unified cache in the Linux core

Cache coloring is a technique that divides memory pages into sections called ”colors.” Each color represents a group of cache sets, and by assigning them to designated cores, we can ensure that the data used by these cores is stored in cache sets that don’t overlap. This helps reduce competition for cache resources and makes the behavior of the system more consistent and predictable.

Unfortunately, the cache partitioning mechanism provided by cache coloring is not available in Jailhouse for ARMv7 architectures. Given that cache partitioning occurs dynamically in the current implementation of Jailhouse, its execution is challenging without an input–output memory management unit (IOMMU) [29], which is absent in ARMv7 architectures. While static cache partitioning is feasible and has been accomplished by other hypervisors on ARMv7 architectures, we have deemed its implementation unnecessary for our study. This decision stems from the fact that the STM32MP157 platform is employed as an illustrative case rather than a definitive answer. Our future endeavors will shift focus to newer, more advanced architectures, where dynamic cache partitioning can be more easily implemented.

Therefore, we decided to turn off the cache on the A7 core hosting Linux for this benchmark.

We present the results of the same test with cache disabled in Fig.4.

The spikes observed are significantly reduced in height but still present, likely due to some additional interactions related to the memory. While memory is partitioned between cores, it’s worth noting that both processors access the same memory bank. Nonetheless, even when considering the maximum execution time, we achieve a speed-up of about 32% with the fixed-point implementation and of 66% with floating-point implementation (again, against Fixed Point for MCU), indicating the viability of the isolated core to act as a computational slave.

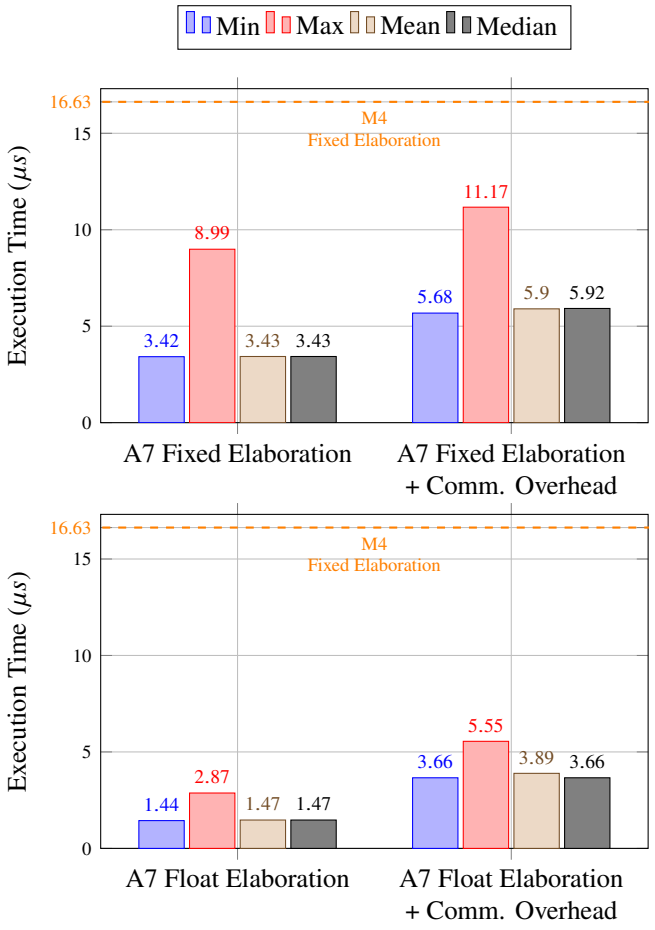

Fig. 4: Histograms representing execution times of the control algorithm with LLC cache disabled on linux A7 core

VI. Conclusion

We have introduced a benchmarking procedure to evaluate the advantages of moving complex computations from the MCU to the MPU. This is a crucial step toward the software architecture we propose to sustain HRT control and diagnostic applications for advanced mechatronics exploiting edge computing heterogeneous platforms.

We applied the proposed benchmark to the STM32MP157 platform, highlighting the advantages and disadvantages of exploiting it. In particular, it is worth noting that the current solution to turn off the cache in the non-HRT dedicated core in such a platform is a suboptimal fix since it considerably slows down such core. However, alternatives like dynamic cache coloring are not readily applicable to the STM32MP157's architecture when using Jailhouse. Since the system lacks an IOMMU and the coloring procedure occurs while activating the hypervisor, ongoing memory translators are unaware of the new memory mapping due to cache coloring. By resetting the system MMU after the dynamic cache coloring procedure, the problem could be easily solved, making the overall system aware of the new memory mapping.

Looking forward, we aim to transition to platforms with ARMv8-A architecture, which includes an IOMMU, like the upcoming STM32MP2, maintaining a good balance between computational power and cost-effectiveness. This shift could effectively address the limitations encountered with the STM32MP157, especially concerning cache management issues.

Additionally, for more demanding applications, exploring higher-performing platforms featuring alternative architectures becomes pertinent. The HULK-V [30] platform, for example, is distinguished by its utilization of RISC-V architecture. RISC-V, known for its open-source nature, offers the advantage of keeping costs down. Such exploration aligns with our aim to enhance resource usage across various computational environments for industrial automation applications.

References

[1] *STM32G474xB STM32G474xC STM32G474xE Datasheet*, STMicroelectronics, 09 2021, rev. 6.
[2] (2021) ACMEC "Additive manufacturing and Cyber-physical technologies for the MEChatronics of the future" Project. [Online]. Available: https://acmec.it/project/
[3] *STM32MP157A/D*, STMicroelectronics, 11 2022, rev. 8.
[4] *i.MX 7Dual Family of Applications Processors*, NXP Semiconductors, 09 2023, rev. 7.
[5] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.
[6] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive e/e-systems," in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*. IEEE, 2014, pp. 189–198.
[7] S. Toumassian, R. Werner, and A. Sikora, "Performance measurements for hypervisors on embedded arm processors," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 851–855.
[8] L. Abeni and D. Faggioli, "Using xen and kvm as real-time hypervisors," *Journal of Systems Architecture*, vol. 106, p. 101709, 2020.
[9] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 75–84.
[10] Y. Shen, L. Wang, Y. Liang, S. Li, and B. Jiang, "Shyper: An embedded hypervisor applying hierarchical resource isolation strategies for mixed-criticality systems," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1287–1292.
[11] M. Barletta, M. Cinque, L. De Simone, R. Della Corte, G. Farina, and D. Ottaviano, "Runphi: Enabling mixed-criticality containers via partitioning hypervisors in industry 4.0," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2022, pp. 134–135.
[12] 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.
[13] A. Bansal, R. Tabish, G. Gracioli, R. Mancuso, R. Pellizzoni, and M. Caccamo, "Evaluating the memory subsystem of a configurable heterogeneous mpsoc," in *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, vol. 7, 2018, p. 55.
[14] M. Cinque, D. Cotroneo, L. De Simone, and S. Rosiello, "Virtualizing mixed-criticality systems: A survey on industrial trends and issues," *Future Generation Computer Systems*, vol. 129, pp. 315–330, 2022.
[15] R. Queiroz, T. Cruz, and P. Simões, "Testing the limits of general-purpose hypervisors for real-time control systems," *Microprocessors and Microsystems*, vol. 99, p. 104848, 2023.
[16] X. Zhang, D. Yang, S. Gao, and W. Dai, "Applying embedded virtualization technologies for the next generation industrial edge applications," in *IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2023, pp. 1–6.
[17] M. Gundall, D. Reti, and H. D. Schotten, "Application of virtualization technologies in novel industrial automation: Catalyst or show-stopper?" in *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, vol. 1. IEEE, 2020, pp. 283–290.
[18] C. Scordino, I. M. Savino, L. Cuomo, L. Miccio, A. Tagliavini, M. Bertogna, and M. Solieri, "Real-time virtualization for industrial automation," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 353–360.
[19] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on preempt_rt," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–36, 2019.
[20] Jgross. (2019) Xen dom0less update. [Online]. Available: https://xenproject.org/2019/04/02/whats-new-in-xen-4-12/
[21] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Workshop on next generation real-time embedded systems (NG-RES 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[22] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no vm exits!(almost)," *arXiv preprint arXiv:1705.06932*, 2017.
[23] EtherCAT Technology Group. [Online]. Available: https://www.ethercat.org/
[24] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.
[25] L. Marconi, F. Ronchi, and A. Tilli, "Robust nonlinear control of shunt active filters for harmonic current compensation," *Automatica*, vol. 43, no. 2, pp. 252–263, 2007.
[26] A. Isidori, L. Marconi, and A. Serrani, *Robust autonomous guidance: an internal model approach*. Springer Science & Business Media, 2003.
[27] D. K. Chaturvedi, *Modeling and simulation of systems using MATLAB and Simulink*. CRC press, 2017.
[28] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 89–102.
[29] E. Bugnion, J. Nieh, D. Tsafrir, and M. Martonosi, *Hardware and software support for virtualization*. Springer, 2017.
[30] L. Valente, Y. Tortorella, M. Sinigaglia, G. Tagliavini, A. Capotondi, L. Benini, and D. Rossi, "Hulk-v: a heterogeneous ultra-low-power linux capable risc-v soc," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

Luca Orciari earned his master's degree in Automation Engineering in 2019 and is currently pursuing a Ph.D. in Biomedical, Electrical, and System Engineering at the University of Bologna, Italy. Before his Ph.D., he was a Research Fellow with the Department of Electrical, Electronic, and Information Engineering (DEI) at the same university for two years. His research spans sensorless observers for electric machines, advanced control applied to compliant mechanisms, embedded real-time control systems, and predictive maintenance for industrial machinery.

Davide Raggini received the Master's Degree in Automation Engineering from the University of Bologna, Italy, in 2017. He is currently a research collaborator at the Department of Electrical, Electronic and Information Engineering (DEI), at the University of Bologna, focusing on the development of embedded control systems for advanced mechatronics. His research interests include embedded real-time control systems and real-time kernels, wireless sensor network (WSN), power and digital electrical and electronics design.

Andrea Tilli received the Ph.D. degree in system science and engineering from the University of Bologna, Italy, in 2000. He is currently an Associate Professor with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI) with the University of Bologna. His research interests include applied nonlinear, adaptive and constrained control techniques, electric drives for motion control and energy generation, advanced mechatronic systems, diagnosis and prognosis of automatic machines, active power filters, and thermal control of many-core systems-on-chip and supercomputers.