DETERMINISTIC PLATFORM SOFTWARE FOR HARD REAL-TIME SYSTEMS USING MULTI-CORE COTS

Sylvain Girbal, Xavier Jean, Jimmy Le Rhun,
Daniel Gracia Pérez, Thales Research & Technology, Palaiseau, France
Marc Gatti, Thales Avionics, Vélizy-Villacoublay, France

Abstract

Future generations of avionic equipments are expected to embed multi-core processors. Using Components Off-The-Shelf (COTS) processors is considered both by the industrial and academic communities, as well as certification authorities. However, in the safety-critical domain, a common issue with COTS multi-core processors is their lack of predictability, directly linked to the difficulty to foresee and manage inter-core interferences due to shared hardware resources.

A possible solution consists in defining a Usage Domain that constrains the use of shared resources down to a level for which interference situations are known and their impact on software execution time is acceptable. Nevertheless, COTS processors have not been designed to see their behavior restricted by such usage domains, and do not provide dedicated mechanisms for that purpose. Hence the usage domains are enforced by more complex mechanisms implemented in dedicated pieces of software running below the applicative level. We call them Deterministic Platform Software (DPS).

The objective of this paper is to propose an overview of existing DPS solutions, and propose criteria leading to a uniform classification. Additionally, we propose a mapping of these solutions to a selection of avionic use cases.

1. Introduction

Multi-core processors usage in avionic equipments has been a topic of interest for several years [1]–[3]. That technology offers today the best computing performance for a reasonable size, weight and power. Moreover, the demand of cheaper equipments [4] makes the choice of Off-The-Shelf (COTS) processors seducing.

A common issue associated to COTS multi-core processor is their lack of predictability [5], [6]. This is mostly due to inter-core **interferences** [7]: Various pieces of software will be executed on different cores at the same time. Even when they do not rely on each other to execute correctly, they compete electronically to use shared hardware resources on the chip. Concurrent accesses are arbitrated by hardware mechanisms, which often entail extra jitters to individual accesses. This is especially the case for the main memory and high-speed busses, e.g. PCIe.

Time critical applications must fulfill their execution before statically defined deadlines. A usual way to provide such guarantees is to evaluate a Worst Case Execution Time (WCET) [8], [9]. However, a significant impact of interferences on execution time distribution has been observed [2]. That hardens the evaluation of interferences impact on WCET with a sufficient level of confidence, and without oversizing the whole avionic equipment.

On multi-core COTS processors, general approaches that target unrestricted configurations, e.g. initiator-target model [10], involve an exponential number of test cases according to the number of cores and the number of shared harwdware resources, and do not provide rationale to ensure that a test case has been correctly covered. Alternative methods require either a closed set of tasks to compute a global interferences penalty, for instance with the isWCET [11] (interferences sentitive WCET) method, or a Usage Domain [12], [13] that explicitly restricts the way shared resources can be used concurrently by cores. This paper deals with the second class of approaches. Enforcing a Usage Domain within shared resources introduces two high-level problems:

- How to define it to ensure interferences mastering?
- How to make sure it is correctly enforced?

The first question has answers in the literature. For instance, projects dealing with deterministic

Work for this paper was executed within the SAFURE project. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

processors, e.g. Merasa [14], PRET [15] or CompSoc [16] have proposed interference-free solutions. These projects have lead to predictable processors that may be synthesized. Their leading idea consists in enforcing timing isolation between each core's communications to shared resources. That is reached with specific hardware mechanisms implementing time-triggered arbitration policies, often named TDMA (Time Division Multiple Access). Similar concepts have been successfully applied on COTS processors [17] at applicative level. Alternatives based on budget allocation and interferences penalties based on these budgets have also been proposed [18].

The second question has been tackled on COTS processors with dedicated pieces of software, that we regroup under the name of "**Deterministic Platform Software**" (DPS). We define them more precisely as pieces of software that bring no functionality by themselves, but are in charge of enforcing Usage Domains over shared resources.

The objective of this paper is to propose criteria to compare DPS solutions relevance in an industrial context (Section 2), give an overview of DPS solutions available in the state-of-the-art and compare them according to the previous criteria (Section 3), and finally to consider these solutions on three case studies that cover different needs in avionic equipment (Section 4).

2. Stakes

Next generation avionic applications will face some performance requirement issues [19]–[21]. Multi-core COTS architectures possess the required inherent performance level, but bring some predictability issues [5], [6] that may endanger the certification processes in avionics.

In this section we will refine interesting **properties** that such DPS needs to provide to be able to exploit the required performance level of the multi-core COTS architecture, while minimizing the impact on the certification processes and costs.

For each property, we will define several levels of assessment from not fulfilled to completely fulfilled. A subset of existing DPS solutions, presented in section 3, will be evaluated against these properties.

A. Support for Legacy Applications

Supporting already certified single-core legacy code in multi-core COTS enable the avionic industry to reduce the overall certification costs.

Table I. Support for Legacy Assessment Levels

0	No legacy support, application needs to be redeveloped from scratch for the platform software.
1	The platform software requires some deep modifications of the source code of the application.
2	The platform software requires the recompilation of the source code, including very light modifications.
3	The platform software is able to run pre-existing unmodified certified binaries.

In Table I we quantified the different effort level associated with the solutions presented in Section 3. The more the legacy software will require to be modified, the more the certification costs will increase, as we will be forced to re-apply the certification process.

The lowest legacy support level (0) means no support for legacy at all, with a complete need to rewrite the software. This is for instance usually the case when you try to parallelize an application at loop nest level.

For a low legacy support level (1), a deep understanding of the application behavior is required to perform some significant modifications. This is for instance the case when altering task orders, identifying behavioral phases at the application level.

A better legacy support level (2), requires light modification of the source code, forcing the application to be recompiled, thus not allowing to reuse a pre-compiled certified binary.

The highest legacy support level (3) is characterized by the ability to run existing, already certified and already compiled binaries, allowing a black-box usage of existing applications.

B. Efficiency of the Proposed Platform Software Solutions

Compared to single-core architectures, multi-core architectures provide us the opportunity for more performance through more parallelism with several execution threads running in parallel on different cores on the system. As presented in Section 1, this property is critical for the next generation of avionic applications that will require much more performance than today.

The way a DPS solution actually enables parallelism exploitation on a multi-core processor is a dimensioning aspect of the efficiency property. Another aspect is the detail of hardware resources and execution time required by a DPS for its own execution. That may lead to individual slow-down of each task. In extreme cases, slow-down of tasks on each core could balance parallelism exploitation, so that the DPS becomes unrelevant. The efficiency property measures those two aspects.

The proposed solutions will also be evaluated against their ability to adapt to different contexts: criticality levels, computation bound vs communication bound applications, and so on.

Table II. Efficiency Assessment Levels

0	Multi-core is not exploited at all, or individual slow-down of applications is higher than gain offered by parallelisation.
1	Balance between parallelism exploitation and individual slow-down is uncertain or highly variable. Concurrent execution on the platform may also be conservatively limited by the platform software to strictly guarantee other required properties, thus reducing parallelism level and performance.
2	Balance between parallelism exploitation and individual slow-down is acceptable in most cases. Concurrent execution on the platform may also be limited by the platform software to guarantee other required properties, while still allowing to significantly exploit parallelism.
3	The platform software does not limit concurrent execution on the multi-core platform, providing optimal performance for the software. Additionally its impact on individual tasks is low.

In Table II we quantified the different effort level associated with the solutions presented in Section 3. The higher the ability of the proposed platform software to exploit the multi-core architecture, the higher the efficiency will be.

For lowest efficiency level (0), the proposed solution is not able to exploit parallelism at all, or introduce overhead within unreasonable proportion. This solution will not benefit from the extra performance provided by multi-core architectures. This is typically the case if you run single-core applications on a multi-core architecture, preventing any other application to run concurrently to avoid potential interferences between applications.

A low efficiency level (1) is characterized by some significant restrictions on parallelism by the platform software to guarantee other required properties, or high individual overheads for each tasks. DPS optimisations must be provided to leverage the risk of low performances in industrial context. Such restriction may only apply to some part of the application (such as only for most critical software), but it may significantly hamper the efficiency if only such kind of critical software is run on a particular multi-core COTS platform.

A better efficiency level (2) will restrict concurrent execution when it will actually endanger the other critical properties, usually using a deeper knowledge of the application. Such solutions will explicitly identify sections that will be allowed to exploit parallelism. Additionally, balance between parallelisation and individual overheads will be positive, so that DPS enables actual performance gains.

The highest efficiency level (3) will provide full access to the multi-core performance by fully exploiting all available parallelism with a negligible or very low individual slow-down. As this will be at the cost of other properties, it is unlikely for such a level of efficiency to be reached for safety critical systems.

C. Robust Partitioning Assessment Complexity

Robust partitioning is a requirement introduced for IMA [22], [23] systems in a single-core context, to guarantee both spatial and timing isolation between the application software running on the same hardware system. Ensuring such a property in a multi-core context is not easy [1], [24]–[26], as shared hardware resources come along with arbitration mechanisms that may break the time isolation property while concurrent accesses to the same resources are performed.

The platform software solutions presented in Section 3 will be evaluated against their ability to ensure robust partitioning in a multi-core COTS context.

In Table III we quantified the different effort level associated with the solutions presented in Section 3. The easiest it is for the proposed platform software to guarantee robust partitioning, the higher this metric will be.

For the lowest assessment level (0), the platform software does not help ensuring robust partitioning, and all the associated guarantees have to be made at application level. It is very unlikely for such a level

Table III. Robust Partitioning Assessment Levels

0	The platform software does not provide robust partitioning guarantees.
1	While the platform software is focusing on minimizing the inter-partition interference, no strict partitioning is guaranteed.
2	Robust partitioning is ensured for most critical part of the application, and relaxed for less critical ones.
3	The platform software provides robust partitioning by construction.

of assessment to be acceptable in an avionic context.

A low assessment level (1) will not strictly be able to guarantee timing isolation and thus robust partitioning, but will be able to bound the impact of inter-partition interferences, still allowing to compute a WCET.

A better assessment level (2) will ensure strict partitioning for most critical applications while relaxing this partitioning property for lower criticality software, with features such as degraded mode.

For the highest efficiency level (3) the platform software provides strict partitioning as currently required by the [22] standard. This usually comes at the cost of lower system performance.

D. Integration into an Industrial Process

The industrial process associated with avionic applications currently involve several actors. Especially *software components* are developed by different *solution providers* working under some computing resource budget constraints, and these solutions are later put together to build an avionic system by the *integrator*. Ideally, if the budget constraints are respected the integration should be seamless.

For single-core solutions this budget is usually expressed with timing constraint, like having 100ms every second to perform the required functionality. Expressing such a budget for multi-core architecture can be much more complex: As the major issue of multi-core is tied to concurrent usage of shared hardware resources [7], a useful budget information could be the share of each resource required by each application. But this is highly architecture-dependent, at a stage of the development process where the hardware target has not yet being finalized. It also forces *solution providers* to have a deep architectural knowledge, and to produce architecture-dependent solution less likely to be reused in future products.

For each platform software solutions presented in Section 3 we will present a guideline on how complex it will be to integrate them into such an industrial process, to avoid the risks defined above.

Table IV. Integration Assessment Levels

0	Cannot be integrated into an existing industrial process. A completely new process needs to be developed.
1	Significant changes to the existing industrial process.
2	Lighter changes to the existing industrial process.
3	Already supported by existing industrial process.

In Table IV we quantified the different effort levels associated with the solutions presented in Section 3. The easiest it will be for the proposed platform software to be integrated into a industrial process, the higher the metric will be.

The lowest integration level (0) means that the proposed solution cannot be integrated into an existing industrial process, and that a new one needs to be developed from scratch.

A low integration level (1) will require significant changes into the industrial process, such as the requirement for software providers to take into account the other applications that will run concurrently with their own, or to anticipate multi-core integration related issues, actually merging the software provider and the software integrator roles.

A better integration level (2) would allow us to keep the roles clearly separated. For instance, we can force the software providers to provide additional information to the integrator for him to take into account how co-running application will interfere, or we need to be able to define more precise computing budget constraints.

The highest integration level (3) is characterized by the fact that current industrial processes already support the proposed solution, indicating a seamless adoption of the solution.

E. Easiness to Adapt Existing Application to the Platform Software

When proposing a new platform software, especially when it comes to certification, the industry only

has to pay the complexity of the platform software once. However, the cost of the adaptation of every application is recurring. Therefore an interesting property is to keep the adaptation costs low.

This adaptation cost is twofold: First, how easy it is to perform the modification in the application software, and second which share of the existing software require to be modified.

The platform software solutions presented in Section 3 will be evaluated against their requirements in term of adaptation of the existing software.

Even though this property is correlated to the Legacy Support, it is still quite different when it comes to the impact. A solution can force to systematically reorganize each task code in a particular well defined and easily applicable way, while it is a major concern for legacy as all parts are modified and need to be certified again. Another solution can involve very complex changes, but very localized in the application, leaving most of it untouched which is better for legacy.

Table V. Easy Adaptation Assessment Levels

0	All applications need to be completely redeveloped.
1	Reorganization of the software architecture is required.
2	Minimal source code alterations are sufficient to support the platform software.
3	No adaptation of existing software required.

In Table V we quantified the different effort level associated with the adaptation of existing application to the platform software solutions presented in Section 3. The less the existing application will require to be modified, the higher this adaptation metric will be.

For the lowest assessment level (0), the applications requires to be rewritten from scratch for the paradigm / platform software to be applied. Such a solution is very unlikely to be considered.

A low assessment level (1) requires large parts of the applications to be re-written. It is typically the case when parallelizing an application at loop-nest level, as the most suited parallel algorithm is usually not the same as the most-suited sequential one.

A better assessment level (2), while not requiring some drastic changes in the structure of the application keeping the same software architecture, may force us to perform slight modifications such as reordering the scheduling, or inserting budget-related information.

The highest efficiency level (3) will require no modification of the existing software, not involving at all the solution providers.

F. Complexity and Certifiability of the Platform Software

Even if it is of lesser importance than the adaptation cost of existing software, the complexity and the certification cost of the platform software itself also needs to be considered when comparing different solutions: A platform software performing dynamic online load balancing may require no modification of the application, while possibly making the platform software not predictable enough to be certified.

The platform software solutions presented in Section 3 will be evaluated against its complexity and certification requirements.

Table VI. Certifiability Assessment Levels

0	Very complex and unpredictable platform software behavior.
1	Complexity of dynamic mechanism not easy to statically bound.
2	New deterministic mechanisms with identified static upper bounds.
3	Minimalist platform software with negligible certification costs.

In Table VI we quantified the complexity and certifiability levels for the platform software solutions presented in Section 3. The lowest the complexity of the platform software will be, the higher the certifiability metric will be.

For the lowest certifiability level (0), the proposed solution is not likely to be certifiable, e.g. not providing the guarantees required by the standards.

A low certifiability level (1) of the platform software involves mechanisms that are not easy to certify such as mechanism based on statistic distributions, or adaptive online mechanisms that can not be statically proved.

A better certifiability level (2) of the platform software will propose deterministic mechanisms to be added. While it requires to certify new mechanisms (such as additional scheduling mechanisms) this certification can rely on existing techniques with static properties.

The highest certifiability level (3) of the platform software is achieved when this platform software remains minimalist enough to not alter existing certification processes. For example control software that only rely on existing already-certified components.

G. Portability to other Multi-core Platforms

Finally, a platform software developed for a particular multi-core architecture needs to be evaluated against its ability to be easily ported to the next generation of the same multi-core architecture, and to a lesser extent to completely different multi-core architectures, indicating the adherence of the proposed solution to the considered hardware.

The platform software solutions presented in Section 3 will be evaluated against its portability to different multi-core architectures.

Table VII. Portability Assessment Levels

0	The platform software is specific to a particular multi-core architecture.
1	The platform software is specific to a particular family of multi-core architecture.
2	Part of the platform software needs to be adapted to each multi-core family.
3	The platform software has no adherence to the architecture.

In Table VII we quantified the portability levels for the platform software presented in Section 3. The easier it will be to port the platform software to different multi-core processors, the higher the portability metric will be.

For the lowest portability level (0) the platform software needs to be re-written from scratch for any multi-core system, and even for an evolution of a particular multi-core system.

A low portability level (1) exhibits an adherence to a particular family of multi-core, requiring some specific features such as the cache locking mechanism of PowerPC. A similar DPS exhibited on other multi-core architectures would require the introduction of new concepts to meet platform software objectives. Result of porting a platform software from one architecture family to another is uncertain.

For a better portability level (2), the platform software needs to be modified to embed new mechanisms that represent minor analysis and development effort. However, the correct behavior of the ported platform software does not represent a significant risk.

The highest portability level (3) does not depend on architecture specificities, and can be easily ported from one multi-core processor to another.

H. Evaluating Properties

In this section, we presented a set of properties that the proposed solutions presented in Section 3 can be evaluated against. Each solution will lead to a compromise between these properties. In the following sections we will try to identify which kind of compromises are relevant, especially for an avionic usage.

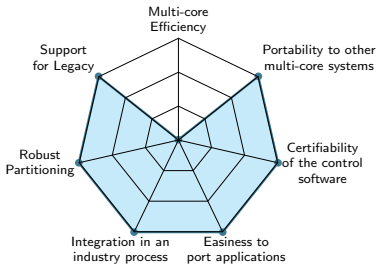

Figure 1. Current Single-Core Solutions Serving as Baseline

These qualitative results will be presented as radar charts similar to Figure 1. Each axis of the radar chart correspond to one of the properties defined in this section. This figure corresponds to the existing single-core solutions and will serve as a baseline.

The figure exhibits both the current lack of efficiency of single-core deployments for future avionic applications, and their ability to be fully mastered by the industry from a certifiability point of view.

3. Deterministic Platform Software Solutions

There are already multiple deterministic software solutions proposed in the literature, the following is a selection of the solutions that we consider the most appropriate for the avionic domain. We classify them in two categories: (A) **control solutions** where determinism is ensured by applying an active control/sequencing mechanism, and (B) **regulation solutions** that provide a reactive mechanism to recover from usage domain violations by applicative software. The former is further differentiated into solutions that do and do not require the applications to be aware of the control mechanism existence.

A. *Control Software Paradigm*

1) Application Aware Solutions: The goal of **Deterministic Execution Model** approaches is to maintain the processor in a predictable state, by applying a strict execution model to the software. More specifically, the deterministic execution models ensure that concurrent accesses to shared resources in a multi-core processor, which are prone to interference, are avoided. The main concept involved in [17], [27]–[29] is to identify several execution phases of application tasks, and to distinguish those which involve accesses to shared resources (including memory hierarchy and internal communication infrastructure) and those which do not.

The AER execution model, described in [17], distinguishes three execution phases :

- Acquisition, during which fresh data are copied from central memory or IO into private memory;
- Execution, performing calculation on the data in isolation within the private memory;
- Restitution, when results are copied back into central memory or IO.

By applying a global scheduling policy to all acquisition and restitution phases, one can ensure that no interference occurs, while still allowing execution phases to happen in parallel, therefore exploiting the performance advantage of multi-core processors. This scheduling is illustrated in Figure 2, exhibiting the exclusion of concurrent shared resource accesses while allowing the parallel execution of isolated computations. As described in [7], this technique works best with a distributed-memory architecture, where execution phases benefit from strict spatial isolation.

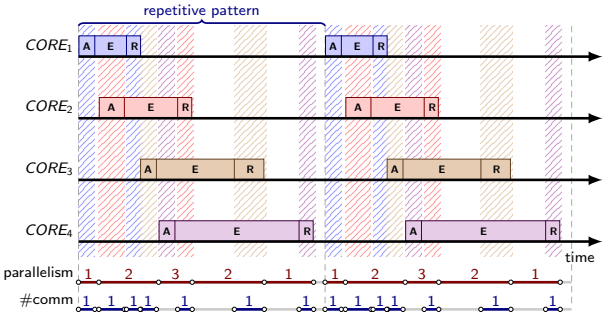

Figure 2. AER Execution Model

As shown in Figure 3 deterministic execution models can be very efficient as execution phases could be run in parallel. However, this efficiency depends on the communication versus computation ratio in the application, as IO-intensive application would run mostly sequentially.

As the control software consists of a static schedule, a proven technology, its certification should be straightforward. The support of legacy applications is limited, but most avionics applications already rely on explicit communications (such as APEX calls [23] for inter-partition communications) and identified phases of central memory accesses (e.g. after a cache flush) to ensure robust partitioning. The adaptation cost is therefore minimized, but source codes still need to be updated.

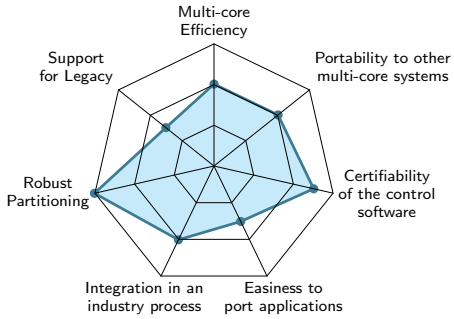

Figure 3. Evaluation of Deterministic Execution Models

Depending on the hardware architecture, further optimizations are possible. For example, if a network-on-chip allows several independent transaction at any given time, that amount of simultaneous communication phases is allowed while maintaining strict isolation. Conversely, allowing a certain amount of simultaneous acquisition or restitution phases on a shared communication medium, such as a bus, still enables the computation of an upper bound on interference. This technique has been implemented in bare-metal environment, the integration into a RTOS would also requires to manage code fetch in acquisition phases.

2) Application Unaware Solutions: With **Deterministic Adaptive Scheduling**, Fischer et al propose in [30] one of the first commercial and certified implementation of a run-time system for critical systems that can be used on multi-cores, more concretely up to SIL 4 on the railway domain. The proposed run-time allows the designer to securely define which applications (or partitions) should not suffer interferences, so they can be executed in isolation on their assigned

time slots. This way, only the core with the critical application runs while the other cores are forced idle. Besides, non-critical partitions (or interference-aware partitions) can be executed concurrently, during their assigned time slots as shown in Figure 4.

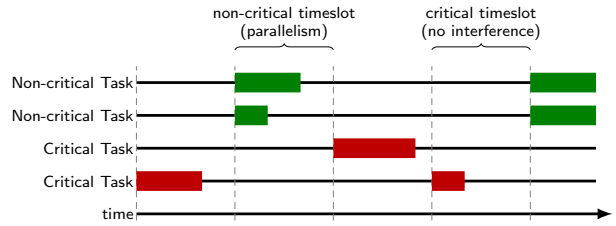

Figure 4. Deterministic Adaptive Scheduling

While suffering of a lack of efficiency when the system is mainly composed of critical applications, this technique enables the usage of multi-core processors and it is easy to certify. Effectively, when executing the critical application only one core is used, making existing single-core analysis and methods applicable. Another advantage is that this proposal does not require the modification of legacy software.

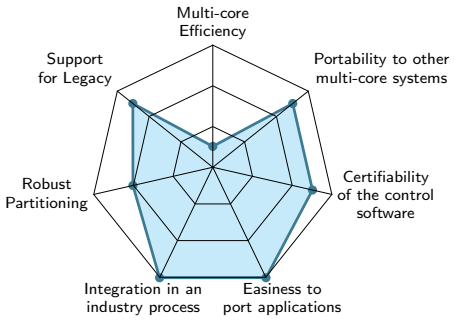

Figure 5. Evaluation of Deterministic Adaptive Scheduling Solution

Jean [25], [31] proposes **Marthy**, a solution to potentially make better use of the multi-core capabilities when running concurrently both critical and non-critical applications in the system. Marthy enables all the cores to run critical applications, but only one of them is allowed to access the shared resources (including main memory) at a given time through a TDMA scheduling of these resources. Sequential resource access eliminates interference, thus providing determinism. To control that a core does not access the shared resources when it is not in its assigned TDMA slot, Marthy relies on smart MMU management and cache locking.

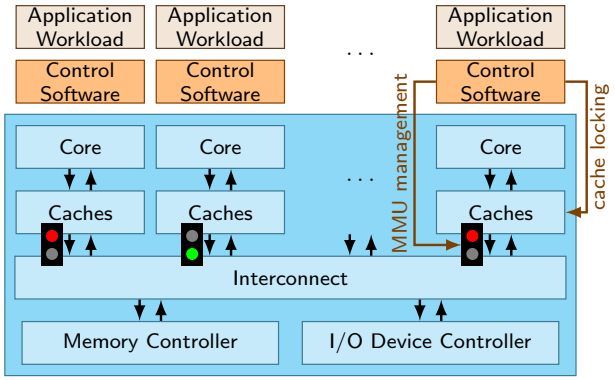

Figure 6. Marthy Deterministic Control Software

As illustrated by Figure 6, when a core tries to access a memory location that is not mapped in the cache, Marthy catches the associated interrupt and blocks the core execution until its assigned TDMA slot.

Experiments on a 4 core processor [31] have shown that the performance of a legacy application can suffer from a $\times 1.3$ to $\times 10$ slowdown due to the control overhead added by Marthy and the applications cache locality. Despite these slowdowns, it is important to consider that:

- The slowdown is deterministic, i.e. the slowdown does not depend on the other applications behavior.
- Marthy allows to run multiple critical applications in parallel.

For example, consider an application that in isolation takes 1 second to execute and suffers a $\times 1.3$ slowdown when using Marthy. When executing in parallel 4 instances of this same critical application, one on each core of a 4-core processor, the cumulative execution time would only be 1.3 seconds. The same scenario when using the Deterministic Adaptive Scheduling approach would require 4 seconds to complete the execution of the four application instances, as they would have to be executed sequentially. Likewise, if the application slowdown when using Marthy was $\times 10$, the cumulative execution time with Marthy would be 10 seconds, while the previous approach would require only 4 seconds to execute them.

Strictly speaking, Marthy does not require legacy applications to be modified. However, in case of a significant slowdown, it might me beneficial to

reorganize data access in the application to improve cache locality.

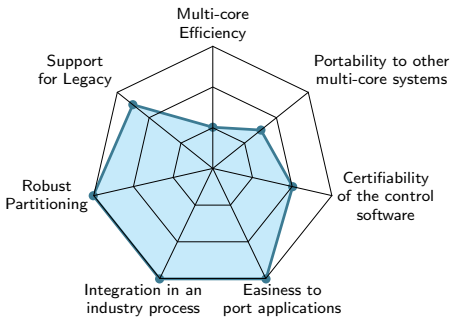

Figure 7. Evaluation of Marthy

B. Regulation Software Paradigm

Regulation software solutions do not aim at avoiding interferences, but at managing them so that critical applications do not miss their critical requirements, which typically consist in respecting their deadlines.

The *Single-core Equivalent Virtual Machines* project [32] provides a regulation software solution with the **Memguard** mechanism [18]. Memguard relies in a memory system with caches that are private or that can be partitioned at the initialization of the system, to avoid functional interferences at this level and only managing resource contention at the interconnect and the memory level. Memguard also relies on the cores being synchronized. With these requirements satisfied, Memguard provides regular time slots of configurable length.

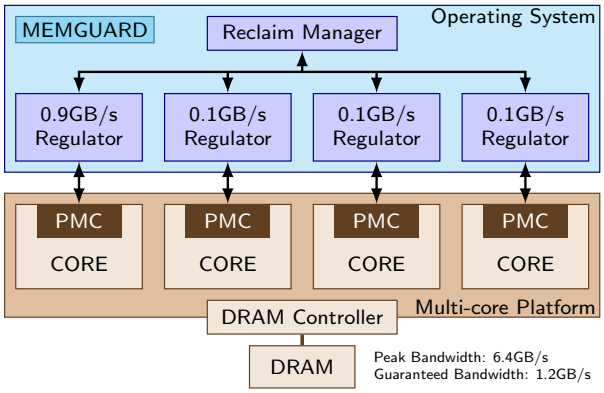

Figure 8. Memguard Reservation and Reclaiming System

Memguard uses the bus access performance counter, with interrupt generation support, to allocate to each core timeslot a maximum bandwidth usage, i.e. maximum number of bus accesses the core can make in a given timeslot. In order to avoid interferences, the sum of allocated bandwidths should be less than the system memory sustainable bandwidth. At the beginning of a timeslot the performance counter is reinitialized before the core executes its assigned application. When the maximum number of accesses is reached, Memguard stops the core execution until the beginning of the next slot. For each timeslot, the system integrator is responsible for allocating bandwidth to each core, making sure that the total allocated bandwidth should not exceed the maximum sustainable bandwidth. Memguard allows to further enhance the system performance by sharing unused bandwidth over several timeslots. However, it involves speculation, reducing partitioning guarantees, as summarized in Figure 9.

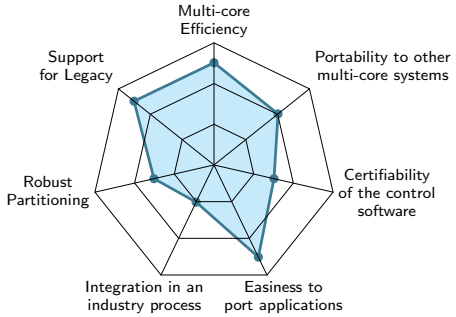

Figure 9. Evaluation of Memguard

In [33] Kritikakou et al. propose a **distributed run-time WCET controller**, another regulation software solution enabling the user to run a critical application at the same time as non-critical applications.

With this approach, critical tasks regularly check if the interferences due to other tasks can be tolerated. Otherwise, critical tasks request the WCET controller to temporarily suspend low criticality tasks as shown in the scenario presented in Figure 10.

Therefore, support for legacy is not fully supported as the critical tasks need to be instrumented to perform the regular checks. Robust partitioning is not fully ensured, but when detecting a risk of deadline miss for a critical task the system tries to still match the deadline by delaying non critical tasks.

Unlike control software solutions, regulation software solutions allow the execution in multiple cores that might result in interferences. However,

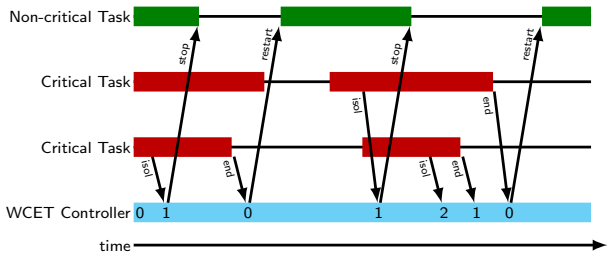

Figure 10. Distributed Run-time WCET Controller

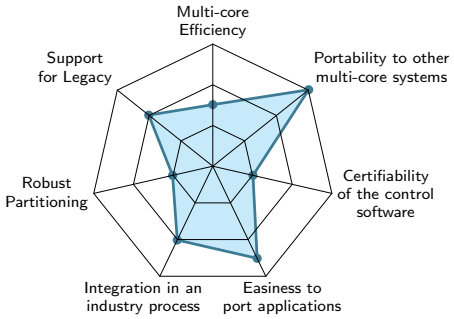

Figure 11. Evaluation of Distributed Run-time WCET Controller

these techniques often rely on budgeting. Extra analysis steps are required to accurately compute these budgets without pessimistic over-provisioning. The accurate quantification of the interferences effect remains an open problem in both regulation techniques.

4. Avionic Case Studies

In this section we present four case studies composed of avionic equipments and detail their needs regarding the properties that were introduced in section 2. As it may impact the priority of these properties, we selected case studies with different levels of safety requirements.

For avionic equipments, safety requirements are expressed as Design Assurance Levels defined relatively to what the effect of a failure might be. DAL are ranging from DAL-A for most critical subsystems for which a failure is likely to imply some catastrophic consequences, down to DAL-E for non-critical application for which a failure will have no impact on the flight.

We are covering the following classes of avionic systems, their criticality level appearing in Table VIII:

- a DAL-A Full Authority Digital Engine Control (FADEC) subsystem,
- a DAL-A to DAL-D Integrated Modular Avionics (IMA) subsystem,
- a DAL-C to DAL-D Data Server subsystem, and
- a DAL-E In-Flight Entertainment (IFE) subsystem.

Table VIII. Avionics Equipments Needs

FADEC	IMA	Data Server	IFE
DAL-A	DAL-A		
	DAL-B		
	DAL-C	DAL-C	
	DAL-D	DAL-D	
			DAL-E

A. *Characterizing Subsystems / Applications*

As for properties defined in Section 2 we evaluated these case studies against 7 evaluation axis organized in radar charts:

- The **Performance Requirement** axis reflects the expected requirements in term of performance for next generations of the case study. It has to be compared to the *multi-core efficiency* axis of each DPS solution.
- The **Legacy Requirement** axis illustrates the necessity for *legacy support*. This is usually tied to the certification costs and thus the DAL level of the considered case study.
- The **Partitioning Requirement** axis indicates how much the case study relies on *robust partitioning*.
- The **Integration Requirement** axis indicates how much the case study fits into a multi-actor integration process. It has to be evaluated against the *integration into an industry process* property.
- The **Application Complexity** axis reflects the complexity of the case study source code. It has be be checked against the *easiness to port application* property associated with each DPS solution.
- The **Certification Costs** axis estimates the overall cost when certifying the case study. This is tied both with the DAL level of the application and its complexity. It has to be compared against the *Certifiability* of the DPS.
- The **Diversification Requirement** reflects the requirements for replication and diversification. This criterion as to be checked against the *Portability* property.

In the following sections, we will evaluate these properties for 4 different subsystem case studies, considering future generation of these subsystems, likely to run on multi-core processors.

B. FADEC Subsystem

The Full Authority Digital Engine Control (FADEC) is a critical subsystem in charge of controlling all aspects of the aircraft engine performance. Its purpose is to provide optimum engine efficiency for a given flight condition.

The flight crew has usually no means of manually overriding the FADEC engine control, and in case of a total FADEC failure occurs, the engine fails. Safety is therefore of prime concern, and redundancy with diversification is a common practice.

For such a high-critical control-command system, the equipment provider manages simultaneously the development of the hardware platform and the software applications. In this condition the equipment designer can introduce mechanisms managing all the shared resources accesses at the application level. Therefore proof of determinism for certifiability is performed at design time.

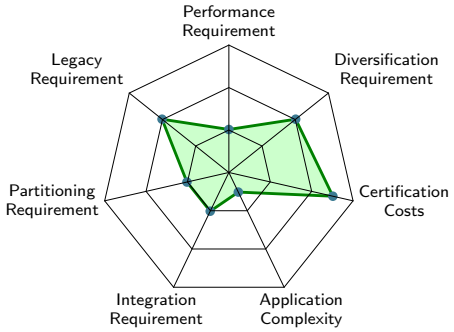

Figure 12. FADEC Application Requirements

Figure 12 summarizes the requirements of the FADEC subsystem with regards to the properties defined in Section 2. Being a control-command application, the application complexity remains low with average future requirements in performance. However, the certification costs and the ability to diversify is of prime concern for a DAL-A subsystem.

The best suited DPS for this case study are Execution Models, Deterministic Adaptive Scheduling and Marthy. Memguard and Distributed run-time WCET controller approaches fails to fit the certification requirements of a DAL-A subsystem.

C. IMA Subsystem

Integrated Modular Avionics systems were introduced to run several high performance mission computing software on the same hardware component. IMA subsystems safety requirements can range from DAL-A down to DAL-D.

Their purpose is to: 1) reduce weight, space and energy requirements by sharing the same hardware; 2) reduce conception and certification costs with an incremental certification process; 3) reduce maintenance and upgrade costs during the aircraft lifespan.

The DO-197 standard [22] organizes IMA development though three different actors: The *platform supplier* developing the hardware platform and kernel software services, the *system integrator* performing shared resource allocation for the different software functions to integrate, and several *application suppliers* developing the avionic functions,

IMA modularity simplifies the development process of avionics software, enabling concurrent and independent conception and certification of different avionic functions.

To deal with resource sharing, IMA subsystems are strongly relying on robust partitioning, as shown in Figure 13. Running several software components on the same hardware also makes it very sensitive to legacy support.

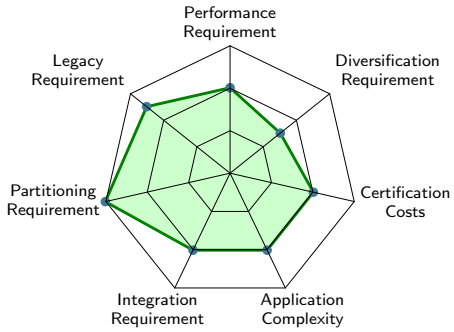

Figure 13. IMA Application Requirements

As the robust partitioning property is part of the definition of IMA standard [22], the best suited DPS for this case study are Marthy and Execution Models. The first one will lack some of the performance requirement to ensure legacy. The second will deliver a sufficient level of performance at the cost of lower legacy support. The Deterministic Adaptive Scheduling approach will provide a sufficient level for

these two properties, at the cost of a smaller support for robust partitioning. Finally, the Memguard and the Distributed run-time WCET controller DPS will hardly fit this case study due to their poor support of robust partitioning.

D. Data Server Subsystem

The data server subsystem is in in charge of the management of the communication with satellites using SATCOM, of the crew communication and of the maintenance interface. The associated computing platform should be capable of hosting communications management services, performing some network management, and acting as a network server or file server. Having less stringent requirements in term of real-time constraints this subsystem is typically a DAL-C or a DAL-D application.

For such a case study, throughput performance is more important than pure computation performance or memory bandwidth. As a consequence, most of the multi-core related interferences will occur while accessing I/Os. Otherwise, all the requirements along the other identified axises are low to average as depicted in Figure 14.

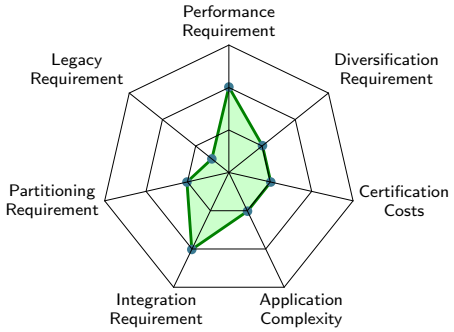

Figure 14. Data Server Application Requirements

The best suited DPS for this case study is Distributed run-time WCET controller. The Execution Models approach is also an excellent candidate, however it will suffer from communication over computation ratio. Marthy and Memguard approaches are farther away, the first lacking some performance, and the later having issues with the integration process. Finally, Deterministic Adaptive Scheduling will hardly match the case study performance requirements, as this approach is as conservative for low-criticality tasks as it is for high-criticality tasks. Extra performance is only available for non-critical tasks.

E. In-Flight Entertainment Subsystem

The In-Flight Entertainment subsystem, running at DAL-E, is dedicated to the passengers' entertainment with no safety-critical requirements but an important demand for processing and communication efficiency.

The IFE subsystem is hosting multimedia applications that are very demanding in terms of performance and that can achieve a high level of complexity.

While not being purely safety-critical, IFE systems are frequently managed by external content service providers. Beyond cost efficiency, system safety and reliability remains a design issue for these systems: To contain any possible issues, IFE systems are typically isolated from the other systems of the aircraft. However such systems usually involves miles of wiring with added weight and associated risks of voltage arcing or current leaks.

In recent years, IFE has been expanded to include Wi-Fi connectivity services through satellite networking or an air-to-ground networking, encompassing new safety / security concerns.

Figure 15 summarizes the requirements of the IFE subsystem with regards to the properties defined in Section 2. Involving mostly best-effort applications similarly to the consumer electronic market, performance is a key property. Application complexity can also be quite high, while the other properties could be relaxed.

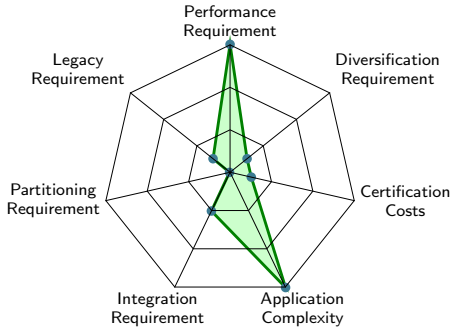

Figure 15. IFE Application Requirements

The best suited DPS for this case study is Memguard. As their efficiency is increased when dealing with non-critical tasks, the Deterministic Adaptive Scheduling and the Distributed run-time WCET controller approaches will also be efficient. Marthy and Execution Models DPS will hardly fit this case study,

as the first will not match the performance requirements, and the complexity of the applications will make the later requirement to update the application very costly.

F. Evaluation Summary

We evaluated the presented Deterministic Platform Software solutions versus the four avionic subsystems with different level of safety requirements.

Table IX. DPS Behavior Evaluation Summary

	good	mid	bad
Marthy	1	2	1
Execution Models	2	1	1
Memguard	1	1	2
Distributed run-time WCET controller	2	0	2
Deterministic Adaptive Scheduling	2	1	1

Table IX enumerates for each DPS how many times it was suited for the selected case studies, how many times it was not completely suited, and how many times it was failing to handle the case study.

Each of the proposed approaches both succeeds for at least one of the case study and fails for another one. So none of the approach clearly outperforms the others, and the most appropriate solution seems to be safety-level dependent.

As a consequence, to select the optimal solution, it is critical to identify the critical properties associated with the considered applications. Each of the proposed case studies has a specific property that cannot be relaxed: certification costs for FADEC, robust partitioning for IMA, ... Identifying these properties allows us to select the most suitable approach.

Finally, in this paper we have not considered mixed critical systems that could run concurrently applications of different criticality levels. Some of the proposed DPS, especially the Deterministic Adaptive Scheduling and Distributed run-time WCET controller approaches were developed for such a mixed critical context.

5. Conclusion

In this article, we proposed a survey of a representative set of Deterministic Platform Software solutions. These solutions constrain a multi-core COTS processor within a usage domain that restricts the use of shared hardware resources, so that inter-core interferences are either eliminated, or bounded with a sufficient level of confidence.

We have proposed high-level criteria to compare these DPS solutions, and to establish a uniform classification. Those criteria cover several aspects of DPS relevance for industrial usage, including:

- Efficiency of applicative software
- Support of legacy code
- Extra certification costs for DPS
- Complexity of Robust Partitioning analysis
- Compliance with industrial processes, e.g. IMA
- Portability of DPS to various COTS processors

The comparison of DPS has shown that all solutions reach a diffrent compromise on these criteria. Particularly, efficiency of applicative software and support of legacy code seem to be antagonistic. Hence a common trend for each DPS is to improve the bad criterion while keeping the good one unchanged. Opportunities to ease legacy code migration or to perform black-box optimisations seem promising.

Maturity of DPS solutions is also variable. Some of them have been developed for well identified processors, or processor series, and would require more representative test cases to gain maturity and robustness for potential industrialization. Other solutions, such as Deterministic Adaptative Scheduling or Memguard, have already industrial support and seem to have reached the point where they could be used in certified products.

Finally, that ecosystem would benefit from a global support from the industrial community. Hence future avionic equipments could embed COTS multicore processors with confidence in their final certification, by using a DPS solution that fits their needs optimally.

References

[1] L. M. Kinnan, "Use of multicore processors in avionics systems and its potential impact on implementation and certification," in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, IEEE, 2009, 1–E.

[2] J. Nowotsch and M. Paulitsch, "Leveraging multicore computing architectures in avionics," *European Dependable Computing Conference*, pp. 42–52, 2012.

[3] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset, and A. Merigot, "Studying co-running avionic real-time applications on multi-core cots architectures," in *Embedded Real Time Software and Systems conference*, 2014.
[4] T. G. Baker, "Lessons learned integrating COTS into systems," in *Proceedings of the First International Conference on COTS-Based Software Systems*, ser. ICCBSS '02, 2002, pp. 21–30.
[5] R. Kirner and P. Puschner, "Obstacles in worst-case execution time analysis," in *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008, pp. 333–339.
[6] E. Mezzetti and T. Vardanega, "On the industrial fitness of wcet analysis," in *Proceedings of the 11th International Workshop on Worst Case Execution Time Analysis (WCET2011)*, 2011.
[7] S. Girbal, D. G. Pérez, J. L. Rhun, M. Faugère, C. Pagetti, and G. Durrieu, "A complete toolchain for an interference-free deployment of avionic applications on multi-core systems," in *Proceedings of the 34th Digital Avionics Systems Conference*, ser. DASC'2015, Prague, Czech Republic, 2015.
[8] P. Puschner and A. Burns, "Guest editorial: A review of worst-case execution-time analysis," *Real-Time Systems*, vol. 18, no. 2/3, pp. 115–128, 2000.
[9] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, 36:1–36:53, 3 2008.
[10] A. Roger and V. Brindejonc, "Avoidance of dysfunctional behaviour of complex cots used in an aeronautical context," in *Lambda-Mu*, 2014.
[11] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, 2014, pp. 109–118.
[12] *CAST-32 Position Paper, Multi-core Processors*, Federal Aviation Administration, European Aviation Safety Agency, 2014.
[13] X. Jean, M. Gatti, G. Berthon, and M. Fumey, "MULCORs, The Use of MULticore proCessORS in Airborne Systems," European Aviation Safety Agency, Tech. Rep. EASA 2011.OP.30, 2012.
[14] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, pp. 66–75, 2010.
[15] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, ser. CASES '08, Atlanta, GA, USA: ACM, 2008, pp. 137–146.
[16] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "Compsoc: A template for composable and predictable multi-processor system on chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, 2:1–2:24, Jan. 2009.
[17] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Embedded Real Time Software and Systems*, ser. ERTS '14, Toulouse, France, 2014.
[18] 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.
[19] E. Bailey, "Study report on avionics systems for the time frame 2007, 2011 and 2020," *European Organisation for the Safety of Air Navigation (EOSA)*, vol. EUROCONTROL, 2004.
[20] C. Ebert and C. Jones, "Embedded software: Facts, figures and future," *Computer*, vol. 42, no. 4, pp. 42–52, 2009.
[21] D. Dvorak and M. Lyu, "NASA study on flight software complexity," *Jet Propulsion*, p. 264, 2009.
[22] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE), *Do-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations*.

[23] ARINC, *ARINC Specification 653: Avionics Application Software Standard Interface*, Aeronautical Radio INC, 2005.
[24] J. Littlefield-Lawwill and L. Kinnan, "System considerations for robust time and space partitioning in integrated modular avionics," in *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, IEEE, 2008, 1–B.
[25] X. Jean, D. Faura, M. Gatti, L. Pautet, and T. Robert, "Ensuring robust partitioning in multicore platforms for ima systems," in *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, IEEE, 2012, 7A4–1.
[26] M. Faugere and al, *Certainty: Certification of real time applications designed for mixed criticality, appropriate multicore usage for mixed-critical system certification*, Press Release, 2014.
[27] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing predictability on multi-processor systems with shared resources," in *Workshop on Reconciling Performance with Predictability (RePP), 2010*, 2009.
[28] 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 *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, 2011, pp. 269–279.
[29] V. Jegu, B. Triquet, F. Aspro, C. Pagetti, and F. Boniol, "Method and device for loading and executing instructions with deterministic cycles in a multicore avionics system having a bus, the access time of which is unpredictable," English, pat. EP 2438528 A1, 2012.
[30] S. Fisher, *Certifying Applications in a Multi-Core Environment: The World's First Multi-Core Certification to SIL 4*, 2013.
[31] X. Jean, "Hypervisor control of COTS multicores processors in order to enforce determinism for future avionics equipment," PhD Thesis, Telecom ParisTech, 2015.
[32] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale, *et al.*, "Single Core Equivalent Virtual Machines for Hard Real-Time Computing on Multicore Processors," University of Illinois at Urbana-Champaign, Tech. Rep., 2014.
[33] A. Kritikakou, C. Pagetti, C. Rochange, M. Roy, M. Faugère, S. Girbal, and D. Gracia Pérez, "Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems," in *Proceedings of the 22th International Conference on Real-Time and Network Systems (RTNS'14)*, 2014, pp. 139–148.