A COMPLETE TOOLCHAIN FOR AN INTERFERENCE-FREE DEPLOYMENT OF AVIONIC APPLICATIONS ON MULTI-CORE SYSTEMS

Sylvain Girbal, Daniel Gracia Pérez, Jimmy Le Rhun, and Madeleine Faugère,
Thales TRT, Palaiseau, France.
Claire Pagetti, and Guy Durrieu,
ONERA, Toulouse, France.

Abstract

In the safety critical domain such as in avionics, existing embedded solutions based on single-core COTS processors are very unlikely to handle the new level of performance requirement of next generation safety-critical applications.

One alternative would be to use multi-core COTS computers, but the predictability versus performance trade-off remains an obstacle for their use in a safety critical context: concurrent accesses to shared hardware resources are generating inter-task or inter-application interferences, breaking the isolation principles required by such critical software.

To enable the usage of multi-core processors on safety critical systems, interferences need to be controlled and techniques need to be developed to exploit multi-core performance benefits.

In this paper, we have developed an approach and an associated tool suite able to enforce an interference-free system execution while emphasizing task parallelization to benefit from multi-core systems inherent performance.

Providing strong certification guarantees of interference-free multi-core systems would require us to identify all potential sources of interference. This is beyond the scope of this paper. While restricting ourselves to the memory subsystems and the I/Os, our goal is to ensure an interference-free execution of a safety critical application deployed on a multi-core architecture, by proposing an approach avoiding interference scenarios.

Our proposed approach couples hardware configurations minimizing interferences with a software execution model decoupling communication phases from execution phases. We are relying on a constraint problem solving (CPS) approach to build an interference-free multi-core deployment.

This approach has been fully automated and is supported by a toolchain from the problem formulation to the code generation. It has been experimented on an avionic application, and both the absence of interference and the performance benefits have been evaluated. With this approach, large safety-critical applications can be ported to multi-core COTS processors while preserving single-core based analysis methodologies.

1. Introduction

Safety-critical domain industries such as the avionics, automotive, space, healthcare or robotic industries are facing an exponential growth of performance requirements [1]–[3]. In the avionic industry, international initiatives such as SESAR, NextGen, and AIRES aiming at modernizing the air traffic management while lowering the impact on the environment, are pushing forward the need of more efficient embedded systems to integrate new performance demanding functions such as 4D trajectory.

Safety-critical software is usually characterized by stringent hard real-time constraints, and missing a single deadline could have catastrophic consequences on either the user or the environment. A common practice to guarantee the deadlines of a safety-critical application with previous single-core COTS architectures was to determine the application Worst-Case Execution Time (WCET). This WCET computation usually relies on analysis tools based on static program analysis, detailed hardware model, as well as measurement techniques [4]–[8].

A. The Challenge of Multi-core Architectures

For several decades, frequency scaling has provided sufficient performance growth to cover the avionic requirements. The processor core was the main shared resource, and the implicit time-sharing

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.

of all other resources of the system was ensured by adequate partitioning at the core level. With the end of Moore's law, chip providers have shifted to the multi-core paradigm to provide effective performance growth. However, this recent shift to multi-core architectures in the embedded COTS market worsened the runtime variability problem [9], [10] as contention on shared hardware resources brought new variability sources. For critical software, providing new solutions to mitigate the impact on the worst case performance is required.

Timing analysis tools are currently unable to properly model such interferences between co-running tasks, due to the lack of an accurate understanding of the contention mechanisms which are frequently undocumented.

Several papers in the literature have quantified the impact of such interferences on the observed runtime of co-running tasks [11], [12]. A significant degradation on the measured WCET can be observed compared to a standalone single-core execution, up to an order of magnitude equal to the number of cores. This is clearly the opposite of the expected speed-up that a multi-core architecture could provide.

To effectively use multi-core architectures for avionic applications, the challenge is therefore to control those interferences to avoid these unsustainable worst case scenarios, and provide ways to ensure time and space isolation properties required by the standards [13]–[15].

A straightforward way to eliminate interferences is to forbid the concurrent execution of tasks that may target the same shared hardware resources. However, preventing parallel execution of tasks accessing the main memory through the interconnect would lead to no parallel execution at all and performances similar single-core architectures, thus providing no advantage from the performance point of view.

B. Contribution

In this paper, our goal is to build an **interference-free** system based on multi-core COTS. It will allow us to compute the WCET when distributing an avionic application to a such a multi-core system relying on existing techniques defined for single-core architectures.

We are relying on an execution model decoupling execution phases from interference-prone communication phases used jointly with the private memories of a distributed memory architecture. It allows us to benefit from some parallel execution while restricting communications on a unique core at a time, thus eliminating interference scenarios on interference-prone channels.

This execution model and the associated architecture are evaluated against a representative test application. A complete toolflow is provided to generate the application deployment (mapping and scheduling) on the target multi-core. The ability of the framework to generate deployments ensuring the absence of interference on the shared resources is evaluated, as well as its ability to scale while increasing the number of tasks or the discrepancy of task periods.

We are comparing our solution against a single-core deployment and a straightforward parallel deployment not considering interferences, and evaluate the benefits of the approach in terms of: 1) the ability to ensure all tasks deadlines, 2) the ability to effectively eliminate all interferences, 3) the capacity of the system to be incrementally extended, and 4) the ability to exploit the multi-core inherent performance based on parallelism.

The paper is organized as follows: In Section 2 we are defining interferences and are providing some guidelines to build interference-free systems. In Section 3 we are presenting the automatic tool-flow we implemented to generate interference-free deployments. Section 4 and 5 respectively correspond to the experimental setup and corresponding evaluations.

2. Multi-core & Interferences

Compared to single-core architecture, multi-core architectures potentially provide more performance by allowing the parallel execution of several applications or tasks. With the increase of performance-demanding functions in avionics, it is becoming critical to be able to exploit such level of parallelism, effectively running different tasks in parallel, while still ensuring **spatial isolation** and **timing isolation** properties. However, running tasks in parallel introduces some **interferences** that may endanger these properties.

A. Defining Interferences

At hardware level, when several co-running tasks, not sharing any data, are trying to concurrently


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.

access the same shared hardware resource (such as the main memory, the interconnect or some I/Os), some arbitration mechanisms are involved at hardware level allowing one of the tasks to access the resource while delaying the others, resulting in a contention, as depicted in Figure 1.

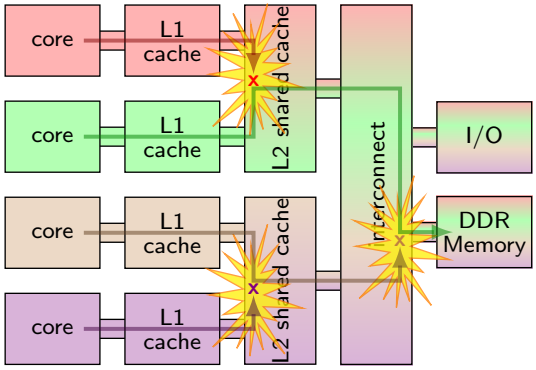

Figure 1. Interferences due to Concurrent Accesses to Shared Hardware Resources.

From the delayed tasks point of view, these extra delays, introduced because of the other co-running tasks unpredictable behavior, are **interferences** breaking the time isolation principles required by avionic standards [13]–[15].

Several studies [11], [12] have already quantified the maximum impact of such interferences on the WCET, exhibiting a potential impact completely offsetting the potential performance gain using multi-core architectures. Multi-core induced interferences therefore need to be controlled, in order to keep the impact on the WCET manageable.

In the next section, we will present some guidelines on how to build an interference-free multi-core system usable in an avionic context.

B. Building an Interference-free System

In order to build an interference-free computing system, we consider both the hardware and the software aspects in a coupled approach.

The main focus for the reduction of interference in safety-critical systems is to ensure spatio-temporal isolation between conflicting operations. This isolation should be provided not only for the targeted shared resource (e.g. shared memory access port), but also along the data and control paths between the initiator and the target (e.g. the interconnect buses or network).

Dealing with interferences at hardware level: On the multi-core processor market, two families of multi-core architectures are available: The fist well represented family, **shared memory** multi-core architectures, illustrated by Figure 1, usually have a memory hierarchy characterized by a large global shared memory coupled with smaller cache memories closer to the cores to provide faster average memory accesses. Such cache memory structures have the advantage to be completely transparent from the programming point of view with no memory management at all in the source code. However, we can hardly predict the time behavior of these cache memories as every data access from the source can either hit or miss depending on runtime behavior.

The second smaller family correspond to **distributed memory** multi-core architectures, and is illustrated by Figure 2.

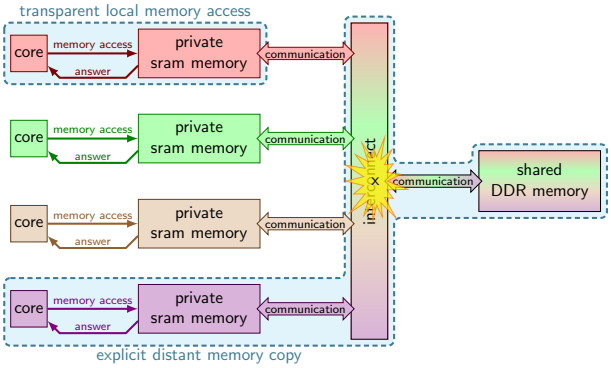

Figure 2. Distributed Memory Architecture

These architectures make a clear distinction between the globally shared memory and smaller private memories tightly coupled with each core. Contrary to shared memory architectures, regular memory instructions (load and stores) can only access the local private memory. Data transfers between private memories or to the globally shared memory have to be made explicit in the source code. If on one hand, adding explicit memory management and data transfers to the application complexify the application, on the other hand it enhances predictability: only these data transfers over the interconnect are interference-prone while memory access to the local private memories could be performed seamlessly with no risk of interferences.

Our main objective being to ensure spatial and temporal isolation, selecting a multi-core COTS that

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.

is not sharing all the hardware resources across the whole system provides us with the opportunity to drastically reduce interference scenarios. Beyond that,

- Some I/O peripherals can be **statically allocated** to a specific core, effectively adding constraints to the application deployment but preventing conflicting accesses.
- A **network-on-chip** could also offer the opportunity to route several transactions along different paths, providing both spatial and timing isolation with better performance than a classical time-shared bus.

Finally, architectural features designed for high scalability (NoC, distributed memories, minimization of centralized resources, etc.) are also beneficial for determinism as they reduce points of contention.

Overall, distributed memory architectures allow us to perform core computation phases in isolation within private memory scope. Code section performing fetch and data movement will be managed as explicit communications over the interconnect. With regards to WCET computation, such a scheme allows us to be much less pessimistic than with unpredictable cache memories. The main drawback is the lack of legacy support because of the need to explicitly control code and data movement in the system.

In critical systems however, and especially in the avionics domain, explicit communication is already an established and enforced principle to achieve spatial isolation, using the APEX interface [16] for safe inter-partition communication. The impact of such a change can therefore be affordable.

For the rest of the paper, we make the assumption that only accesses to the memory subsystem and the I/Os are prone to interference in a multi-core processor. A complete certification process would enforce us to first identify all possible sources of interference, making sure that all of them are taken into account by the proposed approach. Such an identification is not trivial, and beyond the scope of this paper.

An evaluation of the shared memory systems versus distributed memory systems with regards to predictability is presented in Section 5.

Dealing with interferences at software level: At software level, it has been a common practice to rely both on scheduling techniques and execution models [17]–[19] to enforce predictability.

In [20] the authors are proposing the **AER execution model** to decompose application tasks into A (acquisition), E (execution), and R (restitution) phases. The idea is to decouple interference-prone communication phases from computation-based execution phases.

This model allows us to run execution phases in parallel with a minimal risk of interference, while keeping the communication (acquisition and restitution) phases strictly sequential as shown in Figure 3.

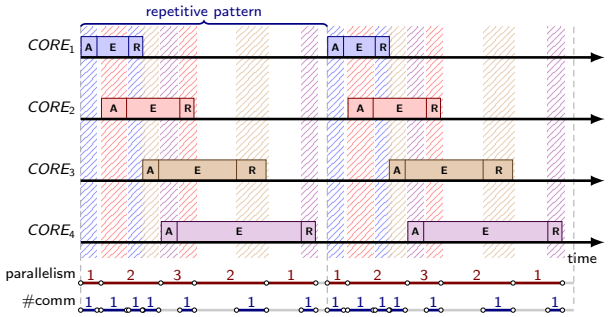

Figure 3. AER Communication Models to Limit Interferences

Coupling **distributed memory** multi-core architectures, with this **execution model** should allow us to have execution phases mapped to the private memories not performing any accesses to the interconnect, and being able to fully run in parallel without generating interferences. Explicit communications, generating traffic on the interconnect, will be performed during acquisition and restitution phases, keeping them sequential to avoid interferences.

Definition of the execution model. In this paper, we consider the distributed-memory configuration of the C6678 architecture presented in Section 4. We highlight three rules to ensure the implementation to be predictable on such a multi-core chip.

Rule 1: the platform memory areas are statically configured.

a) Data and code are stored in the local private memories (L2 configured as SRAM for the C6678) to reduce implicit accesses to the shared resources (TeraNet, DDR or peripherals) as much as possible.

b) Tasks communicate via message passing. Messages are stored in specific areas of the local memories, called Message Passing Areas (MPAs - also in the L2 SRAM). The notion of MPAs is

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.

inspired by the architectural design of the Intel SCC [21], which includes on-chip memories (MPB - message passing buffers) dedicated to message passing.

Rule 2: Tasks are scheduled through partitioned non-preemptive off-line scheduling, also referred as dispatcher.

a) Periodic tasks: tasks are allocated to a core, and on each core, a local schedule is computed with a valid length.
b) Aperiodic tasks: static slots are allocated that respect the requirements expressed as a maximal number of activations during a given amount of time.

Rule 3: All the accesses to the shared resources (MPAs and DDR) are temporally segregated, as defined in the AER model [20]: acquisition and restitution phases must be executed in isolation with other communication phases. Delays for explicit communication between cores or with the external memory and peripherals are pre-computed using a measurement-based approach to determine an upper bound.

3. Automatic Tool-flow to Generate Interference-free Systems

The previous section has introduced a set of rules to be followed in order to obtain interference-free systems. Therefore, the application designer has to map the application according to those rules on the target. Since the execution model is based on static rules which must be applied off-line (off-line mapping and scheduling), we have to offer automatic ways to generate the intereference-free coding of applications.

A. Tool-flow Overview

The tool-flow works in two steps as illustrated in Figure 4. The first objective is to formalize the complete dispatching problem for the application executing on the target hardware platform constrained by the execution model. We describe formally the problem of allocation as a constraint programming problem. We use the OPL tool from IBM ILOG [22] to compute valid solutions. Several benchmarks have been made to highlight the OPL capabilities by playing with different dimensions of the problem (number of tasks, size of the exploration window, platform capacities, ...). Even if the complexity of the problem is exponential, the obtained results are quite good.

The second objective is to automatically generate the header files for the application code. This way, the user can directly execute the mapped application on the target. Two different header files are generated: the first gives the static mapping of tasks on cores and the spatial allocation of the communication buffers in the MPA; the second file contains the local scheduling for each core.

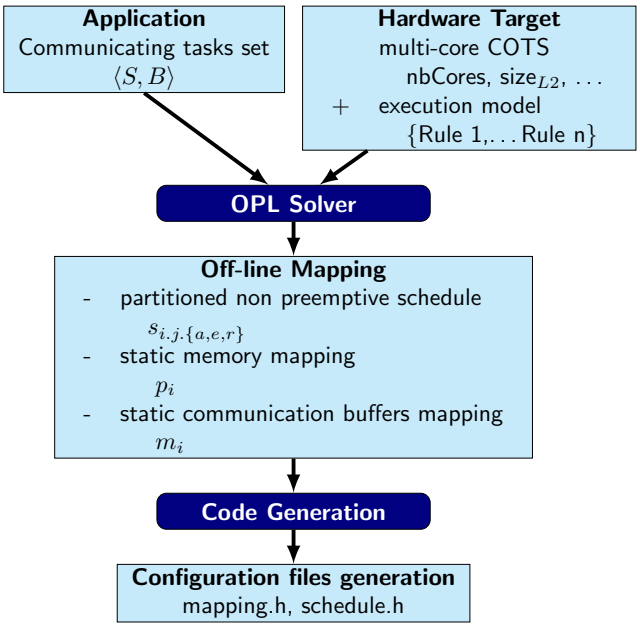

Figure 4. Tool-flow Overview

B. Constraint-based Mapping Problem Formulation

The mapping problem has two inputs: an application model and a platform model. For the application model, we consider an ARINC653-like structure. For the platform model, we consider the distributed memory architecture programmed following the interference-free execution model. The output of the off-line mapping consists of three items:

1) a static mapping of tasks to the cores and their local memories;
2) a static mapping of the communication buffers to the MPAs (message passing areas);
3) a partitioned non-preemptive off-line schedule for each core.

The problem of partitioning and scheduling tasks is equivalent to a multi-dimensional bin-packing prob-

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.

lem, which is NP-hard as shown by [23]. Several formulations of variations of this problem can be found in the literature, e.g. [24]–[27]. The variations stand in the application model and/or platform model, and consequently the constraints to be applied.

1) ***System Model****:* The system model comprises the application and the platform models.

Definition 1 (Application model): The task set $S = \{\tau_1, \ldots, \tau_{nbTasks}\}$ consists of tasks, each $\tau_i = (T_i, WCETa_i, WCETe_i, WCETr_i, size_i)$ being characterized by its period Ti; its WCETs during the three phases A (acquisition), E (execution), R (restitution); its size denoted $size_i$ including both the data and the instructions memory size.

We write $\tau_{i.j}$ to denote the j-th job of task τ_i. Each job is decomposed in the three phases $\tau_{i.j.a}$ for the acquisition, $\tau_{i.j.e}$ for the execution and $\tau_{i.j.r}$ for the restitution. In the sequel, $\tau_{i.j.\{a,e,r\}}$ will refer to a sub-job $\in \{\tau_{i.j.a}, \tau_{i.j.e}, \tau_{i.j.r}\}$. Each sub-job has a release time $r_{i.j}$ and an (absolute) deadline $d_{i.j}$; they are subject to precedence since $\tau_{i.j.a}$ must execute before $\tau_{i.j.e}$ which must execute before $\tau_{i.j.r}$. Thus $\tau_{i.j.a} = (r_{i.j}, d_{i.j}) \rightarrow \tau_{i.j.e} = (r_{i.j}, d_{i.j}) \rightarrow \tau_{i.j.r} = (r_{i.j}, d_{i.j})$ with $r_{i.j} = jT_i$ and $d_{i.j} = (j+1)T_i$.

Usually, when computing off-line scheduling, the schedule (or dispatch) is computed in the hyper-period = $lcm(T_i)$. We propose a way to reduce the size of the length of the schedule: we define the *windowSize* to be the duration on which tasks are unrolled as jobs. To be correct, the schedule on the *windowSize* should be expendable on the hyper-period. The idea is to schedule all jobs released in $[0, windowSize]$, even if the deadline is later than *windowSize*. On this window, we know exactly the number of released jobs for each task τ_i which is denoted $nbJobs_i$.

The set of communication buffers $B = \{b_1, ..., b_{nbBuffers}\}$ describes the communication between tasks. Each buffer $b_j = (src_j, dsts_j, sizeb_j)$ is characterized by its unique source (*src*), its destinations (*dsts*), and its size (*sizeb*).

Definition 2 (Platform model): The multi-core contains *nbCores* cores. Each core owns a two-level hierarchy of SRAM memories and includes a specific on-chip memory area, named the *MPA* (message passing area), that is used to exchange data between processor cores and that is stored physically in the L2. Therefore, to describe the chip, we rely on the following inputs: $size_{L2}, size_{MPA}$.

The platform comes with the rules of the execution model $\{Rule\ 1, Rule\ 2, Rule\ 3\}$ described in the Section 2B.

2) ***Decision Variables****:* Our approach relies on two decision variables:

1) One for modeling the spatial mapping of tasks to cores. We map each task to exactly one core, such that all jobs (for all phases A, E, R) of task τ_i are scheduled on the same core. Let $p_i \in \{1, ..., nbCores\}$ with $i \in \{1, \ldots, nbTasks\}$ be the decision variables that model the mapping of tasks to cores. It takes the value q if task τ_i is mapped to core q;
2) One for computing a valid off-line schedule. Let $s_{i.j.\{a,e,r\}} \in \{1, ..., windowSize\}$ with $i \in \{1, \ldots, nbTasks\}$ and $j \in \{1, ..., nbJobs_i\}$ be the decision variables which represent the start of each job $\tau_{i.j.a}$ (resp. $\tau_{i.j.e}$, $\tau_{i.j.r}$). It takes the value q if job $\tau_{i.j.\{a,e,r\}}$ starts at time q.

For the communication, each buffer b_i is mapped to the communication area on a given core. We assume that producers always put their output data to the local MPA, that is on the same core they run. Let $m_i \in \{1, \ldots, nbCores\}$ with $i \in \{1, \ldots, nbBuffers\}$ be the variables that model the mapping of buffers to the MPAs. It takes the value q if buffer b_i is mapped to MPA's core q. Since m_i can directly be derived from p_i, it is not a decision variable.

3) ***Constraints****:* This section is dedicated to identify the constraints that must be fulfilled by the decision variables. We decompose those constraints per rule.

a) Spatial Mapping Constraints (Rule 1): To fulfill Rule 1.a), we must map tasks such that their sizes do not exceed the size of the local memory. Since *size_L2* is the capacity of the local memory, we use the following constraint to bound the occupied memory:

$$\forall q \leq nbCores, \left(\sum_{i \leq nbTasks} (p_i = q) \times size_i \right) \leq size\_L2$$

Concerning the buffers, we assume that they are always placed in the MPAs belonging to the producer. Thus we simply have the following constraint:

$$\forall i \leq nbBuffers, m_i = p_{src_i}$$

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.

The size of the buffers that are mapped to a core must not exceed the available amount of on-chip memory in that core's MPA. The appropriate constraint to fulfill Rule 1.b) is:

$$\forall q \leq nbCores, \left(\sum_{i \leq nbBuffers} (m_i = q) \times sizeb_i \right) \leq size\_MPA$$

b) Off-Line Scheduling Constraints (Rule 2): In order to allocate enough execution time for each job in the off-line schedule, we have to assume that each job consumes its WCET. Under non-preemptive scheduling, the end time depends solely on the start time $s_{i.j}$ and WCET. Tasks have been divided in three sub-tasks that must be executed in order, meaning that first A has to be executed, then E and at last R. There is no constraint to run those sub-tasks "in sequence" since other sub-tasks can execute between two phases. This leads to the following constraints:

$$\forall i \leq nbTasks, j \leq nbJobs_i, \begin{cases} s_{i.j.a} + WCETa_i \leq s_{i.j.e} \\ s_{i.j.e} + WCETe_i \leq s_{i.j.r} \end{cases}$$

The execution of a job cannot start before its release time, and because of the phase decomposition, it entails that the A phase must start after the release time. A release time is the instant at which a job can start while the start time is exactly the instant when it starts. In the same way, the latest valid time to finish execution is the job's deadline. This applies to the R phase. This leads to the following constraints:

$$\forall i \leq nbTasks, j \leq nbJobs_i, \begin{cases} r_{i.j} \leq s_{i.j.a} \\ s_{i.j.r} + WCETr_i \leq d_{i.j} \end{cases}$$

The constraints have to take into account that only one job can be executed on a core at a time. When τ_i and τ_k are executed on the same core, any job $\tau_{i.j.\{a,e,r\}}$ is scheduled either before or after job $\tau_{k.l.\{a,e,r\}}$, and the start times of the two jobs must be separated by at least the WCET of the job that executes earlier.

$$\forall i,k \leq nbTasks, j \leq nbJobs_i, l \leq nbJobs_k, (i,j) \neq (k,l)$$
$$p_i = p_k \implies \forall o,p \in \{a,e,r\},$$
$$s_{i.j.o} + WCETo_i \leq s_{k.l.p} \vee s_{k.l.p} + WCETp_k \leq s_{i.j.o}$$

c) AER Constraints (Rule 3): The last rule states that A and R phases must be executed in isolation. This can be expressed in a similar way than the formula above which ensures that only one job executes at a time on a core, except that it applies wherever the tasks are mapped: job $\tau_{i.j.\{a,r\}}$ is scheduled either before or after job $\tau_{k.l.\{a,r\}}$, and the start times of the two jobs must be separated by at least the WCET of the job that executes earlier.

$$\forall i,k \leq nbTasks, j \leq nbJobs_i, l \leq nbJobs_k, (i,j) \neq (k,l)$$
$$\implies \forall o,p \in \{a,r\},$$
$$s_{i.j.o} + WCETo_i \leq s_{k.l.p} \vee s_{k.l.p} + WCETp_k \leq s_{i.j.o}$$

*4) **Optimization Criteria**:* When there are several solutions for the constraint programming problem, it is interesting to express some preferences among the solutions. For this, the idea is to define a numerical function from the set of solutions whose values depend on a quality criterion. The objective is then to maximize this function. For the mapping problem, many optimization criteria can be envisioned. As an example, we studied two of them:

The first criterion is to minimize the number of cores used by the partitioning. In that case, the objective will be *min ncores* where *ncores* is constrained as follows:

$$ncores = \sum_{q \leq nbCores} \max_{i \leq nbTasks} (p_i = q)$$

A second possibility is to smooth the utilization on the cores. In that case, the objective will be *min usagemax* where *usagemax* is constrained as follows:

$$usagemax =$$
$$\min_{q \leq nbCores} \left(\sum_{i \leq nbTasks} (p_i = q) \frac{WCETa_i + WCETe_i + WCETr_i}{T_i} \right)$$

4. Experimental Setup

In this section, we present both the avionic application we evaluated and the target hardware platform we considered.

A. Avionic Use-case

As a representative avionic **test application** that may benefit from a multi-core architecture, we used an experimental version of an FMS, based on an operational FMS architecture from Thales, and written in a multi-threaded way for this study.

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.

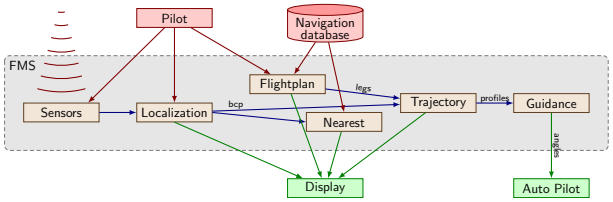

Figure 5. Flight Management System

This test application, which functional view appears in Figure 5, aims at performing in-flight guidance of aircrafts following a set of predefined flight plans. During the flight, the FMS is then in charge of (1) determining the plane localization and (2) computing the trajectory in order to follow these flight plans.

This application, presented in detail in [20] is composed of 9 periodic tasks and 7 aperiodic tasks. Table I summarizes the time critical constraints as well as the memory footprint (code+internal data) requirements associated with each task, and taken into account by the automatic toolflow to build an interference-free scheduling.

periodic task	period	memory footprint	output size
SENS_{C1}	200ms	6352 B	744 B
LOC_{C1}	200ms	8048 B	408 B
LOC_{C2}	1.6s	2280 B	408 B
LOC_{C3}	5s	4360 B	168 B
LOC_{C4}	1s	3996 B	136 B
NEAR_{P1}	1s	2816 B	920 B
TRAJ_{R1}	200ms	7744 B	368 B
TRAJ_{R2}	300ms	7744 B	184 B
TRAJ_{R3}	300ms	7744 B	184 B
aperiodic task	max activation	memory footprint	output size
SENS_{A2}	2 every 200ms	1416 B	256 B
SENS_{A3}	2 every 200ms	1840 B	256 B
LOC_{A1}	2 every 200ms	1616 B	128 B
LOC_{A2}	5 every 5s	1768 B	216 B
LOC_{A3}	5 every 1s	2024 B	216 B
FLPN_{A1}	1 every 200ms	3152 B	1320 B
TRAJ_{A1}	1 every 200ms	1176 B	184 B

Table I. Real-time Constraints and Memory Footprint Associated with the Test Application

Example 1: With the notations given in the constraint programming formulation, there are nbTasks= 16 *tasks. The task* SENS_{C1} *is defined as* $(T_i, WCETa_i, WCETe_i, WCETr_i, size_i) = (200, 2, 11, 2, 6352)$. *For a* windowSize *of 400 ms, the periodic task* SENS_{C1} *is unrolled twice and generates two jobs:*

$$SENS_{C1}.1: \quad (0, 200)$$
$$SENS_{C1}.2: \quad (200, 400)$$

The aperiodic tasks are unrolled on the windowSize and treated as periodic tasks. Thus, for a windowSize *of 400 ms, the aperiodic task* SENS_{A2} *is unrolled into four jobs.*

SENS_{C1} *produces a unique output buffer named* SensorsData *which has a unique consumer* LOC_{C1}. *Thus, this generates the following buffers:*

$$SensorsData: \quad (SENS_{c1}, \{LOC_{c1}\}, 744)$$

SENS_{C1} *consumes several input buffers including:*

$$GpsConfig: \quad (SENS_{A2}, \{SENS_{c1}\}, 160)$$

B. Hardware Platform

As evaluation platform, we selected the Texas Instruments TMS320C6678 [28] depicted in Figure 6 and later referenced as C6678. This multi-core platform is composed of 8 C66x DSP cores clocked at 1 GHz. These cores implement a VLIW instruction set architecture and can issue up to 8 instructions in the same clock cycle. Cores are connected altogether via the TeraNet on-chip network that also provides access to devices such as the I/O interfaces, the DMA engines, or the main memory.

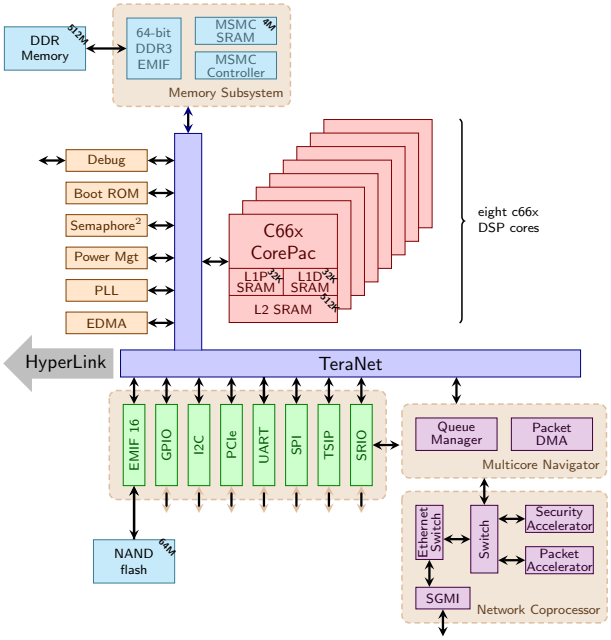

Figure 6. Texas Instruments TMS320C6678

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.

The memory hierarchy is organized as follows: each core embeds a 32 KB level 1 program memory (L1P), a 32 KB level 1 data memory (L1D), and a 512 KB level 2 memory (L2) that may contain instructions and data. Through the TeraNet interconnect, the cores can also access a shared level 3 SRAM memory (MSMC) of 4 MB, as well as the shared external DDR3 memory of 512 MB.

The selection of the C6678 architecture was not because of the DSP cores (we don't actually use the DSP features), and we are only using this architecture as a regular multi-core. The most interesting feature of this particular architecture is its ability to configure its level 1 and level 2 memories (L1P, L1D and L2) either as regular caches or as SRAM memories.

Example 2: With the notations given in the constraint programming formulation, the platform is defined as nbCores$= 8$, $size_{L2} = 520192$ *and* $\text{size}_{MPA} = 3972$.

This feature allowed us to evaluate both the shared memory and the distributed memory configurations presented in Section 2, the corresponding results being presented in Section 5.

5. Results

This section summarizes the experimental results obtained with the tool-flow and on the real target in bare-metal mode.

A. Scalability of the Constraint Programming Solving Approach

To assess the scalability of the approach with respect to the development host, we experimented several parameters:

- the duration of the windowSize;
- the size of the tasks set;
- the size of the MPAs.

The applicability of the proposed approach is tied to the ability of the constraint programming solving (CPS) tool to compute a deployment in an acceptable timeframe. The above-mentioned parameters can have a significant impact on both the search time and the memory usage of the CPS tool.

1) Influence of the WindowSize*:* During a time window, each task τ_i has to be executed a number of time depending on a ratio between the task period and the *windowSize* duration. The schedule of each of these executions, is translated into additional constraints at the CPS tool level. Therefore, the *windowSize* has a significant impact on the number of constraints to be solved.

Figure 7 displays the correlation between the *windowSize* value and respectively the search time and the memory footprint required by OPL to solve the dispatching problem.

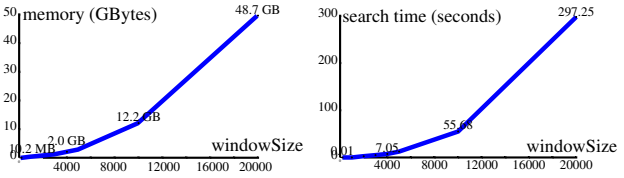

Figure 7. Memory Occupation (GB) and Search Time (s) with Respect to *WindowSize* (ms)

We observe that the occupied memory is multiplied by 4 when the *windowSize* is multiplied by 2. Given our host hardware configuration (63 GB), the maximum *windowSize* which can be processed is about 20000 ms. We therefore cannot run OPL with windowSize=40000 ms (the task set hyper-period) since it would require a memory space of (at least) 200 GB. Memory occupation is thus the limiting factor for analyzing wide time periods.

2) Influence of the Task Sets Size: Similarly, increasing the number of tasks will increase the number of constraints to be solved, thus impacting the complexity of the dispatching problem. To test the scalability of the approach with respect to the number of tasks, we artificially increased their number by instantiating the test application several times. With the default *windowSize*, the test application requires at least 2 cores to be mapped, so no more than 4 full instances of this application could be mapped to the multi-core platform.

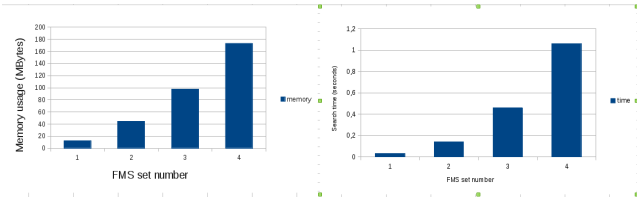

Figure 8. Memory Occupation and Search Time VS the Number of Application Instances

Figure 8 shows the correlation between the number of test application instances and respectively the

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.

search time and the memory footprint required by OPL to solve the dispatching problem.

Compared to the *windowSize* parameter, the number of tasks has a significantly smaller impact on the scalability of the tool. Furthermore, OPL managed to compute a deployment with 4 instances of the test application.

3) Influence of the MPA Size: While previous parameters were impacting the scheduling-related constraints with a significant impact on the number of constraints to be solved, the *MPA size* restrict the available space to map the communication buffers.

We evaluated the scalability of our approach against several values of the *MPA size*. With a MPA size of 1200 B, no solution can be found, as the largest data could not be mapped. However, unlike previous parameters, changing the *MPA size* does not increase the number of constraints, and we were not able to observe any impact on neither the search time nor the memory footprint of the CPS tool.

B. Interference Evaluation of Shared Memory VS Distributed Memory Architectures

In Section 2B, we exhibited that the C6678 platform allowed us to compare shared memory multi-core architectures to distributed memory multi-core architectures with respect to performance and their ability to control interferences.

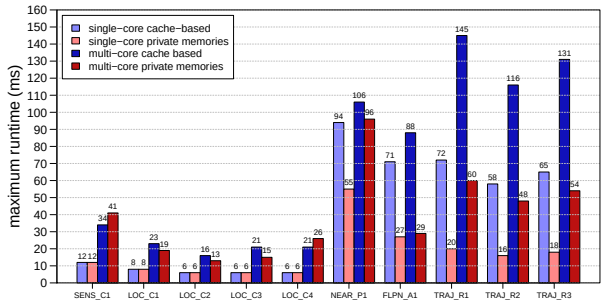

Figure 9. Shared Memory VS Distributed Memory evaluation

Figure 9 corresponds to this evaluation, showing the different individual runtimes of the different tasks composing the test application. The first two configurations correspond respectively to running these tasks sequentially first on cache-based single-core architecture, next on a private-memory-based single-core architecture.

While this single-core deployment exhibits no interferences (all tasks are run sequentially), the private-memory-based configuration already shows some significant performance improvement, with an maximum speedup of 3.61 compared to the cache-based solution.

The last two configurations of Figure 9 correspond respectively to a deployment on a cache-based, and a private-memory-based multi-core architectures. All tasks are run in parallel with no special care for interferences, that can be observed as the speed-down compared to their single-core counterparts.

We observe similar speed-down for individual task runtimes while moving from a single-core to a multi-core configuration, with a speed-down of 2.7 for the cache-based version, and a speed-down of 3.0 for the private-memory-based version.

The overall impact on the maximum task runtimes while shifting the test application from a single-core architecture to a multi-core architecture is summarized in Figure 10.

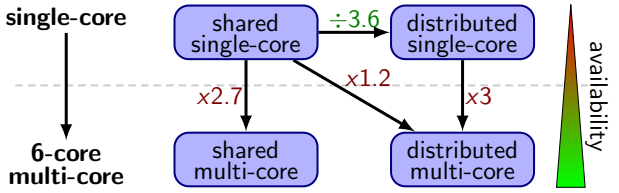

Figure 10. Impact on Individual Task Runtime

Note that such speed-down for shifting from a single-core to a multi-core version can drastically offset the expected performance gain. If the distributed memory version exhibiting less interference could be better suited for time-critical software, it is necessary to be able to control the remaining interferences to achieve an acceptable impact on WCET and overall performance.

C. Evaluation of the AER Communication Model to Eliminate Interferences

To eliminate remaining interferences, we applied the AER communication model presented in Section 2B, using the toolflow presented in Section 3 focusing only on the distributed-memory configuration.

To quantify the impact on interferences of the proposed model, we compared maximum runtime of the tasks composing the test application, against the

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.

single-core standalone version that is interference-free. The results appear in Figure 11.

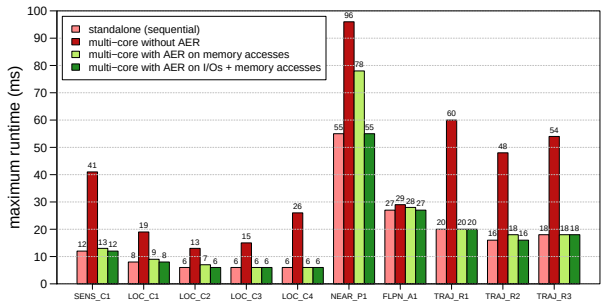

Figure 11. Evaluating the AER Execution Model on the C6678 Distributed Memory Architecture

The first two red configurations respectively show the results of the standalone interference-free version, and the multi-core version without AER, thus maximizing interferences. It corresponds to the same distributed memory results appearing in Figure 9. Compared to the single-core deployment, this first multi-core version exhibits a +200% increased individual runtime in average.

The last two green configurations correspond to worst observed runtimes when applying AER principles on the C6678. For the first one, we applied the AER principles only on memory accesses. Compared to the single-core deployment, we were able to reduce the runtime increase to only +16%.

This remaining extra runtime indicates that all the interferences have not been eliminated. This is especially the case for task `NEAR_P1` which exhibit a +41% increase on the runtime compared to the single-core version.

Beyond memory accesses, another possible source of interference-prone communications on the interconnect are the I/O accesses. All the considered tasks are performing such I/Os to send information to the display. Being responsible for the computation of the nearest airports during the flight, the task `NEAR_P1` is the task producing most of the I/O accesses.

By applying the same AER principles we applied to memory accesses to the I/O accesses, we should be able to eliminate these remaining interferences. To do so, we centralized the I/O accesses on a dedicated core, the other cores performing memory accesses when requesting an I/O access. The results appear as the last configuration of Figure 11, with no more speed-down compared the single-core version, indicating that all the interferences have been eliminated.

D. Performance Evaluation

Using both a distributed memory configuration of the C6678 platform and the AER communication model, we managed to completely eliminate the interferences, enabling the usage of such a platform for time-critical systems, including avionics. But performance-wise we have not yet quantified the gain over a single-core platform.

By completely eliminating the interference overhead, we have made the individual runtime of each task to be the same as for a similar interference-less single-core implementation, as shown in Figure 11. Therefore, these runtimes cannot be used to quantify the performance gain of the multi-core version.

From the time-critical application point of view, the performance is sufficient if the application is able to match all its deadlines. However, what can be compared system-wise is the ability of the platform to run more application concurrently without endangering existing application and their own deadlines.

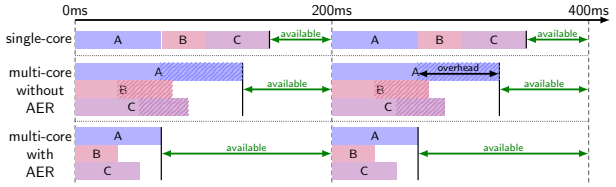

Figure 12. Defining Availability

To do so, we are defining the system availability as the remaining amount of the time window that is freely usable to run additional tasks, as presented in Figure 12. For single-core deployment, this is the remaining time not used in the window period. For interference-prone multi-core, this is the remaining time in this period once all the tasks have terminated their job, including the overhead due to interferences. For interference free multi-core this is the remaining time in this period once all the tasks have finished their job in the current window.

Availability defined this way is still pessimistic for deployment using the AER model, as this model would allow more execution phases to run in parallel. But the results would be dependent on the amount of communication versus computation that the new tasks would perform. Considering that these new tasks

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.

could potentially only perform communication we are able to compute an lower bound on availability, as presented in Figure 13.

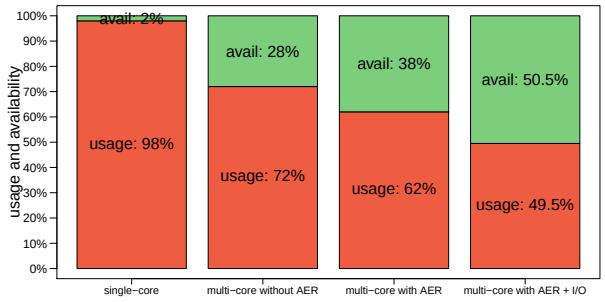

Figure 13. Availability Results

Figure 13 shows the availability results after mapping the test application for different hardware configurations with a time window of 200ms. The first bar corresponds to a single-core deployment that is just able to fit on the system with only 2% of the the 200ms period still available. Second and Third results correspond to deployments on a multi-core architecture that are respectively not caring about interferences and applying AER principles on memory accesses only. These interference-prone deployments of the application exhibit more availability but are not able to ensure the deadlines due to the interferences. The last bar corresponds to the deployment on our interference-free multi-core, and exhibit the most availability with more than 50% of the period being freely available to run more tasks / applications.

6. Related Works

An alternative path to obtain an easy analyzable interference-free system would be to design deterministic and predictable multi-core architectures rather than relying on existing multi-core COTS from the consumer electronic domain.

Such an approach has been defined in PREDATOR [29], and followed by several research projects such as MERASA [30] / parMERASA [31] and PRET [32].

Several studies also target the memory systems and structural hazards for avoiding uncertainty factors. Bui et al. [33] pursue the temporal isolation among multiple processes through some changes introduced at the microarchitecture level, the memory architecture, the network-on-chip, and the instruction-set architecture. Reineke et al. [34] propose a novel dynamic random access memory (DRAM) controller in which the DRAM device is considered as multiple resources shared between one or more clients individually, which improves the predictability of the memory access latency. Lickly et al. [35] focus on integrating timing instructions to a tailored thread-interleaved pipeline. For example, a “deadline” instruction allowing the programmer to set a lower bound deadline on the execution time of a segment of code through accessing cycle-accurate timers.

Alternatively, the PROARTIS project [36] proposes an interesting approach for the WCET problem on multi-cores by proposing architecture designs with randomness. Thanks to the randomness properties of such designs probabilistic approaches can be applied to compute accurate and probabilistic WCETs.

In [37], [38], the author presents an orthogonal approach with a control software scheduling resource accesses coupling MMU management with cache locking techniques. During each time slot, one core is granted a total access to memory hierarchy, restricting other cores to the usage of their first level cache and stalling them in case of a cache miss.

Finally, [39] provides an evaluation of various control software techniques to enable the usage of multi-core COTS for safety critical applications. Most of the studied approaches also focus on getting close to a deterministic usage of multi-core COTS.

7. Conclusion

In this paper, we have developed a complete methodology and associated toolchain allowing to build an interference-free deployment of safety critical applications on a multi-core system.

This methodology relies on an interference-free paradigm in order to favor the certification processor as well as the reuse of single core based design and analysis techniques. Based on a programming model decoupling communication and execution phases and a formal problem formulation as a set of constraints, an appropriate mapping and scheduling is obtained avoiding interferences and favoring parallel task execution.

We applied this methodology to an avionic test application on a particular multi-core architecture managing to eliminate the interferences in the system, while still exploiting the performance in term of parallelism inherent to the multi-core COTS platform.

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.

The scalability of the constraint problem solving tool has also been evaluated.

However, the proposed approach is still far from being certifiable: By avoiding interference-prone scenarios, we are not providing any real guarantee that we are considering nor eliminating all interference channels.

Also the test application is running bare-metal. The execution model has therefore been implemented as part of it. Such an execution model needs to be developed as part of the real-time operating system (RTOS) with additional certification constraints to be taken into account.

Finally, we have shown that our approach fits well with distributed memory systems, but most of the available multi-core COTS follow the shared memory paradigm. For such hardware targets, our approach can rely on prefetch to drastically reduce the number of interferences while not being able to completely avoiding all of them, for instance because of cache conflicts.

References

[1] E. Bailey, "Study report on anionics systems for the time frame 2007, 2011 and 2020," *European Organisation for the Safety of Air Navigation (EOSA)*, vol. EUROCONTROL, 2004.

[2] C. Ebert and C. Jones, "Embedded software: Facts, figures and future," *Computer*, vol. 42, no. 4, pp. 42–52, 2009.

[3] D. Dvorak and M. Lyu, "NASA study on flight software complexity," *Jet Propulsion*, p. 264, 2009.

[4] C. Ferdinand, F. Martin, C. Cullmann, M. Schlickling, I. Stein, S. Thesing, and R. Heckmann, "Program analysis and compilation, theory and practice," in, T. Reps, M. Sagiv, and J. Bauer, Eds., 2007, pp. 12–52.

[5] R. Heckmann and C. Ferdinand, "Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation," in *Proceedings of the conference on Design, Automation and Test in Europe*, ser. DATE'05, 2005, pp. 618–619.

[6] R. Kirner, I. Wenzel, B. Rieder, and P. Puschner, "Using measurements as a complement to static worst-case execution time analysis," in *Intelligent Systems at the Service of Mankind*, vol. 2, 2005.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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*.

[14] ——, *Do-178b: Software considerations in airborne systems and equipment certification*, 1992.

[15] ——, *Do-254: Design assurance guidance for airborne electronic hardware*.

[16] ARINC, *ARINC Specification 653: Avionics Application Software Standard Interface*, Aeronautical Radio INC, 2005.

[17] E. Betti, S. Bak, R. Pellizzoni, M. Caccamo, and L. Sha, "Real-time i/o management system with cots peripherals," *IEEE Trans. Computers*, vol. 62, no. 1, pp. 45–58, 2013.

[18] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on cots hardware," in *25th International Conference Architecture of Computing Systems (ARCS'12)*, ser. Lecture Notes in Computer Science, vol. 7179, Springer, 2012, pp. 98–110.

[19] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable

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.

execution model for COTS-based embedded systems," in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011)*, 2011.
[20] 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.
[21] Intel Labs, "SCC external architecture specification (EAS)," Intel Corporation, Tech. Rep., 2010.
[22] IBM ILOG, *Cplex optimization studio*, 2014. [Online]. Available: http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/.
[23] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory*, vol. 21, pp. 257–298, 1976.
[24] C. Ekelin, "An optimization framework for scheduling of embedded real-time systems," PhD thesis, Chalmers University of Technology, 2004.
[25] K. Schild and J. Würtz, "Scheduling of time-triggered real-time systems," *Constraints*, vol. 5, no. 4, pp. 335–357, Oct. 2000.
[26] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *J. Syst. Softw.*, vol. 81, no. 1, pp. 132–149, Jan. 2008.
[27] W. Puffitsch, E. Noulard, and C. Pagetti, "Off-line mapping of multi-rate dependent task sets to many-core platforms," *Real-Time Systems*, 2015.
[28] Texas Instruments, "TMS320c6678 Multicore fixed and floating-point digital signal processor," Texas Instruments Incorporated, Tech. Rep. SPRS691D, 2013.
[29] PREDATOR, "Design for predictability and efficiency. http://www.predator-project.eu/,"
[30] T. Ungerer, F. J. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf, H. Cassé, 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, no. 5, pp. 66–75, 2010.
[31] T. Ungerer, E. Quinones, H. Ozaktas, I. Broster, J. Fernandes, and S. Kehr, "Parmerasa: Multi-core execution of parallelised hard real-time applications supporting analysability," *Workshop on Advanced Real-time Architectures (ARPA)*, 2012.
[32] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07, San Diego, California: ACM, 2007, pp. 264–265.
[33] D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke, "Temporal isolation on multiprocessing architectures," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11, San Diego, California: ACM, 2011, pp. 274–279.
[34] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, "PRET DRAM controller: Bank privatization for predictability and temporal isolation," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '11, Taipei, Taiwan: ACM, 2011, pp. 99–108.
[35] 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, 2008, pp. 137–146.
[36] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, "Proartis: Probabilistically analyzable real-time systems," *Transactions on Embedded Computing Systems*, vol. 12, no. 2s, p. 26, May 2013.
[37] X. Jean, "Hypervisor control of COTS multicores processors in order to enforce determinism for future avionics equipment," PhD Thesis, Telecom ParisTech, 2015.
[38] X. Jean, D. Faura, M. Gatti, L. Pautet, and T. Robert, "A software approach for managing shared resources in multicore processors for IMA systems," in *Digital Avionics Systems Conference (DASC)*, 2013.
[39] X. Jean, J. Le Rhun, D. Gracia Pérez, S. Girbal, and M. Gatti, "Which deterministic software for hard real-time systems using COTS multi-core processors?" In *Proceedings of the 34th Digital Avionics Systems Conference*, ser. DASC'2015, Prague, Czech Republic, 2015.

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.