A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration

Soukayna M'Sirdi, Wenceslas Godard, Marc Pantel

A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration

Soukayna M'Sirdi[1,2], Wenceslas Godard[1], and Marc Pantel[2]

[1]Airbus Group Innovations, Toulouse, France, surname.lastname@airbus.com
[2]IRIT, Toulouse, France, surname.lastname@irit.fr

***Abstract*—In this paper we propose a framework for the automated integration and timing analysis of IMA (Integrated Modular Avionics) applications on multi-core environments. To do so, we present a derivation of the response time analysis formulation by Kim et al. in [12] that takes into account inter-task interference due to sharing the access to the main memory. We adapt the work in [12] to propose a sufficient schedulability test that is adapted both to IMA systems and heterogeneous multi-core platforms. We then exploit this test to guide the design space exploration during the SW/HW integration phase, to select a partition-to-core allocation so that all deadlines are met despite the existence of hardware interference.**

Keywords: multi-core – IMA – interference – response time analysis

I. Introduction

A. IMA Applications

Modern avionic software systems are designed according to the Integrated Modular Avionics (IMA) architecture model, where (i) applications are defined as one or several software partitions; (ii) several partitions can share the same hardware resources, provided that constraints on time and space isolation are respected between each partition. Such isolation requirements are imposed for safety reasons, to prevent the propagation of a fault that happened inside a specific partition, to all other partitions in the software embedded in a system.

IMA software is organized in two layers: at the top level, a schedule is set in advance to plan the executions of the partitions in a TDMA-like fashion. To do so, activation offsets and time windows are allocated to each partition. We refer to this top level schedule as the global schedule of the system. Within the boundaries of the time window of a given partition, a local scheduler dynamically executes the software tasks inside each partition, usually according to fixed priority preemptive policy. The local scheduler operates with no knowledge about the global schedule. In the other hand, the global schedule has to suit the tasks needs, in the sense that the sizes of the partitions time windows must adapt to the tasks running during these time interval. In this paper, we compute the size of a partition's time window such that the task with the lowest priority level only executes once per window. How we compute the activation offsets of the time windows is out of the scope of this paper though.

During the integration phase of a system design, the allocation of the software platform onto the hardware platform is done, as well as the generation of a valid static global schedule. Setting in advance the global schedule enables to enforce an execution plan that has been verified and validated first. In particular, the global schedule must fit the needs of the local schedule inside the time window of each partition, and always ensure the respect of every deadline defined in the entire system. To verify and validate the global schedule, a formal proof of correctness, such as with static timing analysis, is the preferred choice for certification authorities.

B. Multi-Cores and Inter-Task Interference

In multi-core processors, all cores have simultaneous access to shared resources, through shared interconnections. All the requests made to one resource cannot be processed at the same time, which results in waiting delays at runtime. Such interference between tasks of different partitions, simultaneously executing on different cores, significantly extend the execution times of the tasks, and thus, the sizes of the time windows of the partitions. Since these interference delays may lead to deadline violations, they must be bounded during timing analysis.

However, the sharing level implied by multi-core environments leads to an explosion of the number of possible situations to consider to find the Worst Case Execution Times (WCET) of the tasks, so that static timing analysis at code level on multi-core is quite difficult and not solved today. On the other hand, the electronic market is evolving so fast that single-core processors will soon become obsolete, and will not be produced anymore. Avionic system designers will then have no choice but to move to multi/many-core designs. This represents a major challenge for IMA systems, since no consolidated approach to guide the transition of IMA applications from single- to multi-core platforms has been put forward so far.

C. Contributions

In this paper, we propose a framework for automated integration and timing analysis, of legacy IMA software on multi-core processors. To do so, we first present a sufficient schedulability test that is interference-aware, and compatible both with IMA applications and heterogeneous multi-core processors. The test is based on an extension of the response time analysis presented in [12], and computes safe bounds on inter-task memory interference.

As a second contribution, we exploit the resulting IMA, multi-core interference-aware response time analysis, in an optimization framework to automate the SW/HW allocation and scheduling analysis during the system integration phase. The schedulability test is used as a guide to perform design space exploration, and to find an optimized partition-to-core mapping. The optimization criteria we rely on is the minimization of the total workload of the system. By exploiting such an interference-aware schedulability test inside the allocation search, the solution is guaranteed to preserve the feasibility of the final system, despite the additional delays that might appear due to resource sharing.

Finally, our framework enables to produce bounds for the tasks Worst-Case Response Times (WCRT), as well as for the inter-task memory interference. Such bounds are crucial to formally prove the feasibility of the system, but also to determine the minimum requirements for the time window of each partition, which is mandatory in order to set in advance a valid scheduling plan of the global system.

D. Paper Outline

The paper is organized as follows: section I introduces IMA systems, our problem statement and contributions. Section II gives some insight on the integration process and avionic requirements. Section III gives an overview of the state of the art related to our work. Section IV presents our system model. Section V then presents our extension of the interference-aware response time analysis to IMA and multi-core contexts. Section VI presents the allocation process implemented in our framework. Section VII presents the implementation and results of our work, and section VIII finally concludes our paper.

II. Avionics Systems

A. IMA Systems Integration

During the integration phase of a system design, the software platform is allocated to the hardware platform. The person in charge of this task is called the system integrator. Software functions are usually designed by one or several different suppliers, and are delivered to the system integrator before the beginning of the integration phase. As tasks are scheduled dynamically, their schedule is not set in advance, but each supplier must have verified and validated the schedulability of the tasks of each delivered partition. The job of the system integrator is then to take care of the global schedule, by computing the size and activation offset of the time windows of each partition. We consider legacy code, which means the schedulability analysis performed by each supplier has been verified for single-core environments. Thanks to the sufficient test we present in this paper, the tasks schedulability is verified again during the integration phase, to take into account the fact that partitions will be running in parallel, by including additional inter-task and inter-partition interference delays due to sharing the main memory. One schedulability test that may be used is the response time analysis, which leads to the computation of a bound on the WCRT of each task.

After the production of WCRT bounds, the integrator is able to assess the minimum size required for the time window of each partition: the window must be large enough for the tasks inside the considered partition to complete their execution, even in the worst-case. Once all sizes have been set, the integrator decides of activation dates for the beginning of each time window, starting from the instant when the system is switched on. IMA partitions are periodic, i.e. a time window must be reserved for each partition at a periodic rate. We consider this information as implying not only a period, but also an implicit deadline per partition, equal to the period, since each partition must be assigned a window before its next periodic activation. As a consequence, one important step of the integration process is to verify that, for each partition, time window sizes and activation dates match the periodicity of the partition. If it is not the case, the integrator must consider another SW/HW allocation, for instance by increasing the number of embedded processors for instance. The integrator is likely to have to try different SW/HW allocations before finding a configuration that complies to the requirements of all the described steps.

Eventually, the chronograph resulting from the global schedule consists in the pattern that will be repeated cyclically on the system, for as long as it is switched on. This pattern is also called the major frame. In this paper, we do not address the schedule generation activity of the integration phase.

B. Avionics Requirements

Final validation of avionic systems is done through a thorough certification process, shepherded by various regulations and recommended practices (like [5], [4] for instance). In 2014, the Certification Authorities Software Team (CAST) submitted a first position paper regarding multi-core processors [1]; static, assymetric scheduling is recommended, as it enables to reuse existing code without modification, and since global scheduling would require further proofs and analyses when looking for certification. CAST encourages the privatization of shared resources as much as possible. This goes in the sense of IMA needs, where the more private to each partition are the resources, the better it is. As a consequence, we assume the main memory and shared caches are partitioned into areas private to each core. This reduces the number of inter-core interference to consider: dividing shared cache levels into areas private to each core suppresses inter-core cache interference, as tasks on different cores are unable to evict each other's data. However, partitioning the main memory into core-private areas does not suppress inter-core interference, as long as the memory controller is still shared by the cores. Because all requests cannot be handled simultaneously by the controller, it results in waiting delays at runtime, and thus, inter-core interference, even in the case of a main memory

privatized at core level. In this paper, whenever we mention main memory interference, we actually refer to the inter-core interference due to sharing the memory controller.

To be in line with CAST multi-core study: (i) we consider static partitioned scheduling for partitions: each partition is statically assigned to a core, where it will be activated from the beginning till the end of the life cycle of the system; (ii) main memory and shared caches are partitioned at core level. However, some avionic functions might implement inter-partition communications. In such cases, we assume memory banks are exceptionally shared, between the two partitions concerned; these banks store the shared data.

III. State of the Art

In the literature, no consolidated approach to guide the transition of IMA applications from single-core to multi-core platforms has been put forward so far, so the problem is still open. To our knowledge, our work is the first to combine, inside one approach for the allocation search: (i) an interference-aware multi-core schedulability test; (ii) the computation of a maximum bound for memory interference per task; (iii) the computation of a maximum bound for the WCRTs of the tasks taking interference into account. In addition, our approach is also the first to consider, at the same time: (i) IMA architectures: tasks are defined inside partitions, which implies additional activities and verifications to be handled during software integration, like the computation of the sizes of the time windows of the partitions; (ii) the possible heterogeneity of the platform: in heterogeneous multi-cores, the execution duration of a piece of code depends on the core it is executed on: as a consequence, each task has one execution time in isolation per core.

Table I summarizes the characteristics of related work. The main characteristic of our contribution is the joint management of the allocation and scheduling analysis, with interference considerations. To our knowledge, only [15] and [10] handle these two activities at the same time. However, the interference-aware schedule generated in [15] relies only on a selection of measured execution times of the tasks when running simultaneously with each other, thus no safe bound on interference per task is produced. In [10], model checking is performed to obtain a reliable schedule, and thus safe bounds on tasks worst-case interference. However the author makes strong assumptions on the hardware architecture, which are currently verified only in the Kalray MPPA multi-core COTS [2], and neither the heterogeneity nor the IMA environment are taken into account. In [9] the interference-aware schedule is produced after data path analysis thanks to a detailed hardware model of the platform, which implicitly leads to interference bounds. The work presented in [13] relies on runtime monitoring to dynamically adapt a scheduling plan, thanks to regular comparisons of the remaining time available for each task before its deadline, and its theoretical remaining execution time required to finish, computed in isolation at observation points. Thus no bound on task interference is given, and the correctness of the final schedule is more difficult to prove. In [6], the authors propose an interference-aware multi-core response time analysis that takes into account inter-task interference due to sharing access to the DRAM, cache and bus resources. The framework does not implement any automatic allocation search process though, and is applicable neither to heterogeneous platforms, nor to IMA software architectures. Eventually, the IMA architecture is only considered in scheduling analyses, and except for the work in [9], the heterogeneity is considered only for the allocation search without interference issues.

IV. Model of the System

Let N_p and N_t respectively denote the total number of partitions and tasks of the software platform, and N_c the number of cores of the multi-core processor of the hardware platform. Tasks are denoted as τ_i, and are ordered by their unique priority levels i: τ_i has a higher priority than τ_j if i is smaller than j. We model tasks as a vector $\tau_i = (C_i, D_i, T_i, H_i)$ with $C_i = (C_i^1 \dots C_i^{N_c})$ and $H_i = (H_i^1 \dots H_i^{N_c})$. In the vector C_i, each C_i^k is the worst-case execution duration of τ_i when running in isolation on the core k. The C_i^k parameters can be deduced from static WCET analysis of the code, with tools like OTAWA [7] for example. Similarly, H_i gives the maximum number of data requests to the memory that τ_i can issue depending on its core assignment; a bound on each element of H_i can be extracted after static code analysis. T_i is the period of τ_i, and D_i its deadline.

We model partitions as a vector $\pi_i = (E_i, P_i)$, where E_i is the size of the time window to be reserved for π_i, and P_i is the period of π_i. As explained in the introduction, we compute E_i such that the task with the lowest priority level only executes once per window. To ensure an accurate time window size, E_i is computed after the response time analysis, to guarantee a window large enough for the tasks to complete their execution in the worst-case:

$$\forall i, E_i = \max_{\tau_j \in part(i)} (R_j) \quad (1)$$

where $part(i)$ contains the tasks belonging to the partition π_i. Indeed, by definition, the WCRTs correspond to the situation where every task is preempted as many times as possible during its execution. In our case, we consider all tasks of a partition to be released simultaneously at the beginning of each time window. Priority levels are unique, i.e. two different tasks cannot have the same priority level. As such, for a given partition, the task with the lowest priority level has been preempted as many times as possible by all other tasks of the same partition. As a consequence, if τ_j is the task with the lowest priority level of π_i, then R_j accounts not only for the execution time of τ_j, but also for the execution times of all other tasks of the

Related Work	Avionic Systems	Heterogeneity	Allocation	Interf.-Aware Schedule	Interference Bounding
Bradford et al. [9]	yes	yes	no	yes	yes
Paolieri et al. [15]	no	no	yes	yes	no
Baruah et al. [8]	no	yes	yes	no	no
Tamas et al. [16]	no	yes	yes	no	no
Giannopoulou et al. [10]	no	no	yes	yes	yes
Kritikakou et al. [13]	no	no	no	yes	no
Altmeyer et al. [6]	no	no	no	yes	yes
our work	yes	yes	yes	yes	yes

TABLE I: Summary of related work contributions

partition. τ_j is then the task with the biggest WCRT, and can be used to compute the required size of the time window of its partition as described in equation (1).

The task-to-partition mapping is defined by the matrix $PART$ as follows:

$$PART_{ji} = \begin{cases} 1 & \text{if } \tau_i \in \pi_j, \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Inter-partition communications are modeled by the matrix M as follows:

$$M_{ij} = \begin{cases} 1 & \text{if } \pi_i \text{ and } \pi_j \text{ exchange data,} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Except for E_i, all these parameters are given as inputs to our framework. Additional parameters are computed though. To perform the scheduling analysis, the following variables are defined, for each task τ_i:

- B_i is a maximum bound on inter-task interference due to sharing the main memory;
- R_i is the WCRT of the task;
- $iso(i)$ is the element C_i^k of the vector C_i where k corresponds to the index of the core to which τ_i has been allocated.

These three sets of variables are computed automatically by our framework during schedulability analysis, as will be explained later in this paper.

To perform the allocation search, we implement in our framework the matrix a, defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ is allocated to core } i, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Eventually, to ease the explanations of the interference mathematical model, we also introduce the following sets:

- $part(i)$ contains the tasks belonging to the partition π_i: $part(i) = \{\tau_j, PART_{ij} = 1\}$
- $core(p)$ contains the tasks belonging to the core p: $core(p) = \{\tau_i, a_{pj} = 1, \tau_i \in part(j)\}$

V. IMA Response Time Analysis

A. Definition

The response time analysis first computes the WCRTs R_i of the tasks τ_i, and then, compares the results with the corresponding deadlines: tasks are schedulable if and only if R_i is smaller than D_i. In non-IMA, single-core systems, R_i is computed as the fixed-point solution of the following iterative equation, defined by M. Joseph and P. Pandya [11]:

$$R_i^{k+1} = C_i^{'} + \sum_{\substack{\forall j, \\ \tau_j \in hp(\tau_i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j^{'} \quad (5)$$

where k is the iteration number and $C_i^{'}$ is the execution duration of τ_i in isolation. For the first iteration, R_i^0 is set to $C_i^{'}$. The second term in (5) refers to the maximum waiting delay due to the preemption of τ_i by tasks with a higher priority. The set containing such tasks is denoted $hp(\tau_i)$.

B. Transposition to IMA, Multi-Core Environments

1) Multi-Core: In equation (5), all tasks of the system are on the same single-core processor. In multi-core environments, the tasks are rather dispatched among the cores of the processor. In theory if interferences are ignored, the situation is equivalent to considering N_c independent single-core processors, so that equation (5) can be easily reused: the set $hp(\tau_i)$ will then refer to the tasks of higher priorities that are mapped on the same core than τ_i, since only the tasks on the same core as τ_i could be able to preempt it. As a consequence, if τ_i is mapped to core p:

$$R_i^{k+1} = iso_i + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in core(p)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso_j \quad (6)$$

with iso_i being equal to C_i^p by definition. To compute iso_i during the WCRT analysis, the core on which τ_i is allocated must be retrieved. However, in IMA architectures, tasks belong to partitions, and it is the partitions – not the tasks – that are assigned to a core. Thus, to compute $iso(i)$: (i) the partition to which τ_i belongs is identified first, (ii) the core to which the corresponding partition has been assigned is identified, and then (iii) $iso(i)$ is deduced as equal to the element of the vector C_i corresponding to the core identified in the second step. The mathematical equation corresponding to the computation of $iso(i)$ is the following:

$$\forall i,\ iso(i) = \sum_{p=1}^{N_p} PART_{pi} \times \left(\sum_{k=1}^{N_c} a_{kp}.C_i^k \right) \quad (7)$$

Indeed, $PART_{pi}$ will be non null only if π_p is the partition to which τ_i belongs; then, the term a_{kp} will be non null only if π_p is allocated to core k, which

leads to the identification of the index k of the core on which π_p, and thus τ_i, is mapped, and then to the duration C_i^k.

2) IMA partitions: In IMA architectures, further refinement of equation (6) can be given, according to the two-level schedule. A task can be executed only within the time window of its partition. This implies that the only tasks susceptible to preempt τ_i must belong to the same partition as τ_i. As a consequence, equation (6) becomes:

$$R_i^{k+1} = iso(i) + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso(j) \qquad (8)$$

C. Main Memory Interference Model

As mentioned in section I-B, the response time analysis (5) is often compositionally augmented to consider additional latencies likely to appear in practice. In our case, we add to the response time analysis in equation (6), a bound B_i of the maximum interference delay each task can suffer due to sharing the main memory:

$$R_i^{k+1} = iso(i) + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso(j) + B_i \qquad (9)$$

As shown in figure 1, the memory is divided into banks, and each bank is divided into columns and rows. When a request is treated by the memory controller, the bank to access is identified first, then the row. When a task issues a memory request, the waiting delay suffered before the request is satisfied depends on the order in which the pending requests are treated by the memory controller. We implemented the DRAM memory model presented in [12], where a detailed model of intra- and inter-bank access delays is given, based on a realistic memory model implementing FR-FCFS (First Ready-First Come First Serve) protocol.

The approach in [12] presents two computation methods, and chooses the minimum value of the two produced bounds for each task. The maximum bounds produced thanks to the two methods will be referred to as $B_{i,method_1}$ and $B_{i,method_2}$ respectively.

We present here the mathematical model of memory interference of [12], as well as our modifications leading to our final response time analysis formulation. Due to lack of space, we will only give a brief description of the model, and for a more detailed explanation, interested readers are invited to read [12].

1) Method 1: Request Driven Approach: In the request driven approach, the maximum delay suffered by a request on one core is assessed. To do so, two types of interference are considered: inter- and intra-bank interference.

For a request req issued by one core p, the worst case situation happens when: (i) every other core issued a request just before req; (ii) none of these requests targets the same memory bank as req; (iii) the treatment of each of these requests take the longest latency

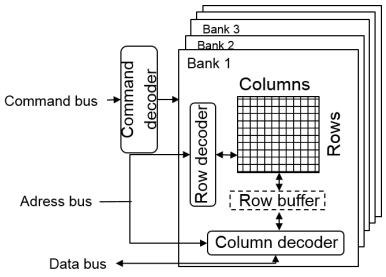


Fig. 1: Model of Memory Banks

possible, l_{max} – which happens when neither the previous bank, nor the previous row were accessed just before, and the type of the request is always different than the preceding one. The computation of l_{max} is explained in detail in [12], and depends on standard DRAM timing parameters that can be found in the DRAM datasheet (see table 1 of [12] for instance). As a consequence, the inter-bank interference delay for a request issued by the core p is:

$$RD_p^{inter} = \sum_{\substack{\forall q, q \neq p\ and \\ shared(p,q) = \varnothing}} l_{max} \qquad (10)$$

where $shared(p, q)$ is the set of memory banks shared by the cores p and q. In our model, the sets $shared(p, q)$ can be deduced from matrices a and M. In particular, $shared(p, q)$ is empty only if no partition on core q is sharing a memory area with any partition on core p. This translates into the terms a_{pi}, a_{qj} and M_{ij} being equal to zero for all partition π_i belonging to p and all partition π_j belonging to q. As a consequence, the emptiness of $shared(p, q)$ can be assessed in our model as follows:

$$(shared(p,q) = \varnothing) \equiv \left(\sum_{i=1}^{N_p} \sum_{j=1}^{N_p} a_{pi} \times a_{qj} \times M_{ij} = 0 \right) \qquad (11)$$

Equation (10) is the inter-bank interference delays, in the case where no request to the same bank as req was produced. In the case where requests to the same bank as req are issued, the longest interference delay for req to be serviced happens when: (i) all the other cores q that share access to the same bank as core p emitted a request before req; (ii) all these requests concern access to a different row; (iii) a memory reordering is happening. If L is the row-conflict service time, to open a row before accessing a column, then the worst-case delay per such request is $L + RD_q^{inter}$. The intra-bank interference delay suffered by req issued by the core p is thus:

$$RD_p^{intra} = reorder(p) + \sum_{\substack{\forall q, q \neq p\ and \\ shared(p,q) \neq \varnothing}} (L + RD_q^{inter}) \qquad (12)$$

As for l_{max}, L depends on standard DRAM parameters that can be found in the memory's datasheet (see [12] for a detailed description). $Reorder(p)$ computes the delay of req due to the reordering effect (see [12] for a detailed description). Following the same reasoning than

in the previous paragraphs, $shared(p,q)$ is non empty only if there exists a partition on core q that is sharing a memory area with a partition on core p. This translates into the terms a_{pi}, a_{qj} and M_{ij} being both equal to one for at least one partition π_i belonging to p and one partition π_j belonging to q. As a consequence, the emptiness of $shared(p,q)$ can be assessed as follows:

$$(shared(p,q) \neq \varnothing) \equiv \left(\sum_{i=1}^{N_p} \sum_{j=1}^{N_p} a_{pi} \times a_{qj} \times M_{ij} \neq 0 \right) \tag{13}$$

Finally, the total maximum interference delay a request originating from the core p can experience is equal to:

$$RD_p = RD_p^{inter} + RD_p^{intra} \tag{14}$$

Each task τ_i of core p having H_i^p requests to issue, the total maximum interference delay directly caused by the issuing of these requests is bound by $H_i^p \times RD_p$. Tasks being preemptible though, the cost of memory requests of tasks with higher priorities has also to be accounted for. Therefore, the bound on τ_i's memory interference is defined as follows:

$$B_{i,method_1} = H_i^p \times RD_p + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil H_j^p \times RD_p \tag{15}$$

2) Method 2: Job Driven Approach: In the second method, the author of [12] focuses on how many interfering memory requests are generated during the execution of a task. The maximum number of requests generated by a core p during a time interval t, $A_p(t)$, depends on the number of generated requests by its tasks that are executed during that same time interval:

$$A_p(t) = \sum_{\forall \tau_i \in core(p)} \left\lceil \frac{t}{T_i} \right\rceil H_i^p \tag{16}$$

As an IMA environment, during the execution of τ_i, only the tasks of the same partition as τ_i are eligible to emit requests for a given core. Thus the maximum number of requests that core p can generate during the execution of τ_i is:

$$A_p(t_i) = \sum_{\forall \tau_j \in part(i)} \left\lceil \frac{t_i}{T_j} \right\rceil H_j^p \tag{17}$$

where t_i is the time interval during which τ_i is executed.

Once the maximum number of memory requests generated during a given time interval has been expressed thanks to the definition of A_p, one can express inter- and intra-bank interference delay imposed on a core p during a time interval t, respectively:

$$JD_p^{inter}(t) = \sum_{\substack{\forall q, q \neq p \ and \\ shared(p,q) = \varnothing}} A_q(t) \times l_{max} \tag{18}$$

$$JD_p^{intra}(t) = \sum_{\substack{\forall q, q \neq p \ and \\ shared(p,q) \neq \varnothing}} (A_q(t) \times L + JD_q^{inter}) \tag{19}$$

Finally, to consider the maximum memory interference delay τ_i can suffer, equations (18) and (19) compute the elapsed time between the beginning and the end of the execution of τ_i. To produce a bound that is safe in the worst-case situation, the time interval to consider is R_i. As a consequence, if τ_i is allocated on the core p, then the maximum bound for the memory interference it can experience during its execution is given by:

$$B_{i,method_2} = JD_p^{inter}(R_i) + JD_p^{intra}(R_i) \tag{20}$$

D. Final Definition of R_i and B_i

Methods 1 and 2 being about the computation of maximum bounds, B_i is set to the less pessimistic of the two:

$$B_i = \min\left(B_{i,method_1}, B_{i,method_2}\right) \tag{21}$$

Eventually, the final formula to compute safe bounds on the WCRTs to perform the analysis is given in equation (9), where B_i is computed according to equation (21).

VI. Allocation Process

During the allocation process, the system integrator decides on which processor each partition will be executed. In the case of assymetric scheduling on a multi-core processor, the integrator should decide on which core each partition will be allocated. An allocation then refers to a partition-to-core mapping, each partition being supposed to run on the same core from the beginning till the end of life of the system.

After an allocation has been chosen, the system integrator then sets a global schedule for each core of the multi-core. To do so, the integrator first computes the size of each partition time window, and then sets an offset for the activation of each window inside the major frame of each core. The size of a time window depends on the tasks supposed to run within its boundaries: the window should be large enough for all these tasks to complete their executions. The offset of a window should be chosen so that the end of the window is before the beginning of the next period of the partition.

If these conditions are verified, the resulting global schedule on each core are considered to be valid, as well as the partition-to-core allocation that led to such schedules. If on the contrary, it is not possible to find a valid schedule for at least one core of the multi-core, another allocation must be selected. The allocation phase is thus important, since the validity of the computed global schedules depends on it.

We propose to automate the SW/HW allocation process of the design phase of a system. To do so, we perform an exhaustive search for a valid allocation using constraint programming. The constraints represent the scheduling requirements of the system. To be able to do so, we rely on equation (9) to guide the search. Figure 2 summarizes our exploration process:

a) Step 1: the first step consists in choosing an allocation, in order to evaluate if it is a potential solution for the SW/HW integration. The allocation is chosen automatically by the framework, among all the possible combinations. Eventually, all allocations will be evaluated during the exploration process.

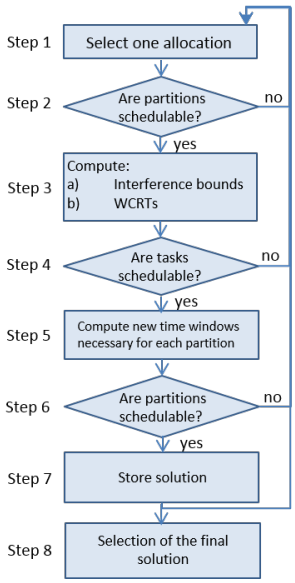


Fig. 2: exploration process

b) Step 2: The second step represents an early verification that each partition window can end before the next period. This cannot be verified with certainty before knowing the activation offsets in the global schedule, nor before computing the tasks WCRT with equation (9). However, if one window happens to be larger than the period of the corresponding partition, then the selected allocation could never lead to a valid global schedule. As a consequence, we add the following constraints, to help the early rejection of such invalid allocations:

$$\forall i, E_i \leqslant P_i \tag{22}$$

where E_i is computed according to equation (1), and each R_i with equation (8). If this condition is not respected, the search process goes back to step 1 and a new allocation is chosen.

c) Step 3: If step 2 is successful, R_i and B_i are computed for all tasks thanks to equations (9) and (21).

d) Step 4: The schedulability of the tasks in the multi-core allocation currently under evaluation is assessed. Tasks are schedulable if and only if:

$$\forall i,\ R_i \leqslant D_i \tag{23}$$

e) Step 5: If step 4 is successful, we update the sizes E_i of the time windows of the partitions: now that the interference-aware WCRTs of the tasks are available, we have to check that the time windows are still large enough in the worst-case, i.e. when the durations of the tasks are equal to their R_i. If it is not the case, the window size of the corresponding partitions are increased accordingly thanks to equation (1).

f) Step 6: The sixth step is based on the same principle as the second step, evaluated with the new E_i values. If a window E_i is larger than the corresponding period P_i, it means that the currently assessed allocation may lead to deadline violations due to memory interference. The search process then rejects the currently evaluated allocation as invalid, and goes back to the first step to choose a new allocation.

g) Step 7: If the sixth step is successful though, the current allocation is stored as a valid solution.

h) Step 8: The last step of the process is the selection of a solution among the valid allocations that were stored at step 7. We propose to do so according to an optimization criteria. Our framework selects the solution that minimizes the total workload of the system. We justify our choice by the fact that this parameter is a performance characteristic, but also because it is proportional to inter-task interference delays, which add up to the tasks executions. Because of these delays, (i) the extra time available before tasks deadlines are reached at runtime is drastically shortened, and (ii) the integrator is given less flexibility for the setting of a valid global schedule for each core in the considered allocation. The objective function we defined in our framework is the following:

$$minimize \sum_{i=1}^{N_t} \frac{R_i}{T_i} \tag{24}$$

VII. Implementation

We implemented the optimization problem proposed in this paper, to assess the benefits that can be drawn from its use. To do so, we compared several qualities of the solution returned by our approach, to the solutions that would have been returned by "classic" approaches, meaning approaches without interference consideration. As explained before, no such classic approach exists, so we modified a copy of our optimization problem into a classic allocation problem: we remove the interference bound in equation (9), but we still compute it thanks to equation (21), to get the corresponding interference. The resulting process is referred to as "classic approach" or "interference-oblivious solving" in the rest of the paper. We implemented the optimization problem in CPLEX [3], that has been running on a computer with an Intel Core i7 2.20 GHz processor with 16Gb of RAM.

A. Presentation of the Case Study

We derived a case study from the report [14], which presents a specification of an avionics Mission Control Computer (MCC) system. No mention is made about an IMA architecture, nor about maximum numbers of requests to the main memory by the tasks, but for the sake of the analysis, (i) we consider each function to correspond to a partition, as if each one had been designed by a different supplier; (ii) we randomly generate H_i^k values for each task, ranging from low to intensive usage of the main memory, respectively 1 and 40 requests per microsecond [12]. We also built the C_i vectors by deriving the execution time in isolation given for each task in the report. The data corresponding to our case study is summarized in table II.

Partition	tasks C_i (ms)	T_i (ms)	D_i (ms)	P_i (ms)
1	C_1=(8, 7.2, 7.6, 6.4)	55	55	480
	C_2=(6, 5.4, 5.7, 4.8)	80	80	
2	C_3=(2, 1.8, 1.9, 1.6)	40	40	480
	C_4=(2, 1.8, 1.9, 1.6)	80	80	
	C_5=(2, 1.8, 1.9, 1.6)	480	200	
3	C_6=(4, 3.6, 3.8, 3.2)	40	40	480
	C_7=(1, 0.9, 0.95, 0.8)	480	40	
	C_8=(2, 1.8, 1.9, 1.6)	480	40	
	C_9=(1, 0.9, 0.95, 0.8)	480	200	
4	C_{10}=(1, 0.9, 0.95, 0.8)	10	5	480
	C_{11}=(7, 6.3, 6.65, 5.6)	100	100	
	C_{12}=(1, 0.9, 0.95, 0.8)	480	200	
	C_{13}=(1, 0.9, 0.95, 0.8)	4800	200	
	C_{14}=(2, 1.8, 1.9, 1.6)	480	400	
	C_{15}=(6, 5.4, 5.7, 4.8)	480	400	
5	C_{16}=(1, 0.9, 0.95, 0.8)	1920	40	1920
	C_{17}=(1, 0.9, 0.95, 0.8)	1920	40	
	C_{18}=(6, 5.4, 5.7, 4.8)	52	52	
	C_{19}=(6, 5.4, 5.7, 4.8)	52	52	
	C_{20}=(8, 7.2, 7.6, 6.4)	52	52	
	C_{21}=(1, 0.9, 0.95, 0.8)	100	200	
	C_{22}=(2, 1.8, 1.9, 1.6)	1000	1000	
	C_{23}=(1, 0.9, 0.95, 0.8)	1920	200	
	C_{24}=(1, 0.9, 0.95, 0.8)	1920	200	
6	C_{25}=(1, 0.9, 0.95, 0.8)	100	200	480
	C_{26}=(2, 1.8, 1.9, 1.6)	480	400	
7	C_{27}=(2, 1.8, 1.9, 1.6)	200	200	480
	C_{28}=(3, 2.7, 2.85, 2.4)	480	100	
8	C_{29}=(5, 4.5, 4.75, 4)	1000	400	1920
	C_{30}=(1, 0.9, 0.95, 0.8)	1920	200	
	C_{31}=(10, 9, 9.5, 8)	1920	800	

(a) Description of the tasks and partitions

H_i used for the first test
H_1 = (160000, 144000, 152000, 128000)
H_2 = (30000, 27000, 28500, 24000)
H_3 = (42000, 37800, 39900, 33600)
H_4 = (70000, 63000, 66500, 56000)
H_5 = (44000, 39600, 41800, 35200)
H_6 = (80000, 72000, 76000, 64000)
H_7 = (21000, 18900, 19950, 16800)
H_8 = (14000, 12600, 13300, 11200)
H_9 = (15000, 13500, 14250, 12000)
H_{10} = (12000, 10800, 11400, 9600)
H_{11} = (133000, 119700, 126350, 106400)
H_{12} = (2000, 1800, 1900, 1600)
H_{13} = (26000, 23400, 24700, 20800)
H_{14} = (20000, 18000, 19000, 16000)
H_{15} = (6000, 5400, 5700, 4800)
H_{16} = (1000, 900, 950, 800)
H_{17} = (26000, 23400, 24700, 20800)
H_{18} = (192000, 172800, 182400, 153600)
H_{19} = (240000, 216000, 228000, 192000)
H_{20} = (80000, 72000, 76000, 64000)
H_{21} = (7000, 6300, 6650, 5600)
H_{22} = (52000, 46800, 49400, 41600)
H_{23} = (34000, 30600, 32300, 27200)
H_{24} = (29000, 26100, 27550, 23200)
H_{25} = (14000, 12600, 13300, 11200)
H_{26} = (24000, 21600, 22800, 19200)
H_{27} = (48000, 43200, 45600, 38400)
H_{28} = (60000, 54000, 57000, 48000)
H_{29} = (135000, 121500, 128250, 108000)
H_{30} = (340000, 306000, 323000, 272000)
H_{31} = (10000, 9000, 9500, 8000)

(b) Vectors H_i used in the first test

TABLE II: Case Study Data

B. Tests Performed

As we will explain in detail in the next paragraphs, we performed two different experimentations. In both cases, our goal is to compare the solution returned by our framework with the solution that would have been provided by interference-oblivious processes. The comparison is made on the basis of the workload reduction achieved thanks to our framework, but also on the interference percentage, and the slowdown suffered by the tasks, in the selected solution.

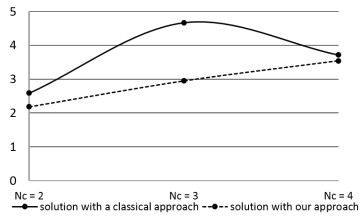
1) Test 1: In the first test, the number of cores N_c of the multi-core platform is the only variable parameter. We ask, both our framework and a classic allocation search, to find an allocation of the eight partitions described in table II on N_c cores. The H_i vectors used in the first test are described in table IIb. The maximum possible number of cores is eight, corresponding to the situation where each partition is allocated alone on a core. The result of the allocation search is either a valid SW/HW allocation, or the answer that there exists no valid solution for the number of cores considered. As we will explain in detail in subsection VII-C "Results", no solution was found for N_c greater than or equal to five; that is the reason why the C_i and H_i vectors in table II only contain four elements each, and not eight.

2) Test 2: Since memory interference depends on the H_i values, the comparison results and the realized performance gains are very data-dependent. To get a general appreciation of the optimization gain achievable thanks to our framework, we test different intensities of memory utilization by the tasks. Such a second test also enables to see the evolution of inter-partition interference with the intensity of the memory usage by the tasks. As mentioned earlier, memory utilization intensity varies approximately between 1 and 40 requests per microsecond per task. As a consequence in this second test, there are two variable parameters: N_c and the vectors H_i. We use data from table IIa but not from table IIb, and we build the vectors H_i as follows instead, for all tasks τ_i and all cores k:

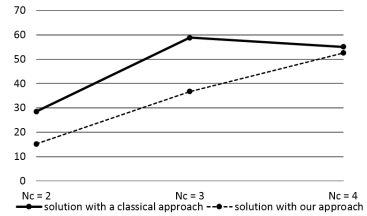
$$H_i^k = C_i^k(in\,\mu s) \times x, \quad x \in \{1, 10, 20, 30, 40\} \quad (25)$$

C. Results

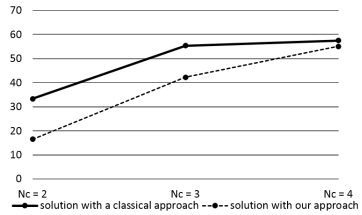
For both tests, our framework was unable to find solutions for N_c greater than 4. On the contrary, in the classic approach, the search always finds a solution, but after analysis of the output memory interference bounds B_i, the solution appears not to be valid in reality: some bounds lead to actual WCRTs being bigger than the deadline of the corresponding tasks, which invalidates the feasibility of the selected solution. The difference between our framework and the classic approach being the interference consideration, we can draw the conclusion that our framework prevents from choosing allocations that, later on during the schedule planning phase, appear to be infeasible. This also implies that our work enables to reduce the time spent during the last phases of a system design.



(a) total workload



(b) interference percentage in the total workload



(c) median slowdown percentage

Fig. 3: Results of the first test

In terms of computation time, solutions were always found in less than 10 seconds with two cores, 62 seconds with three cores and 7 minutes with four cores.

1) Test1: Figure 3 shows the results of the first test. According to the figure, our framework always returns a better solution than the classic approach, the difference between the two solutions being more noticeable for three cores. We were able to show some significant performance enhancement, with up to 36.9% of workload reduction, 46.6% of interference reduction and 50.4% slowdown reduction.

2) Test 2: Figure 7 displays the results of the second test. The graphs on the left show the results corresponding to our framework, whereas the graphs on the right correspond to the solution with classic approaches for the allocation search. The same legend is used in all graphs, to ease results interpretation. In all runs performed, the solution found with our framework is better than with classic approaches, by allowing up to 39.6% workload reduction, 54.5% interference reduction and 44.4% slowdown reduction.

Eventually, for a given value of N_c, one can see the evolution of workload, interference and slowdown depending on the intensity of memory usage by the tasks. All three parameters increase with the usage intensity. The only exception is for four cores in figures 4b and 5b: workload and interference obtained with x equal to 40 in equation (25) are respectively lower than with x equal to 30, and is quite close to the solution found by our framework. Memory interference can only increase with the number of memory requests, since tasks using intensively the main memory are more likely to suffer more from interference than the tasks that only perform a small number of requests to the memory per execution. This implies that the classic approach managed by chance to find an optimized solution for x equal to 40, but failed to do so for x smaller than 40. This can be considered as an expected observation, since the objective function (24) in classic approaches ignores the B_i terms in the computation of the WCRTs.

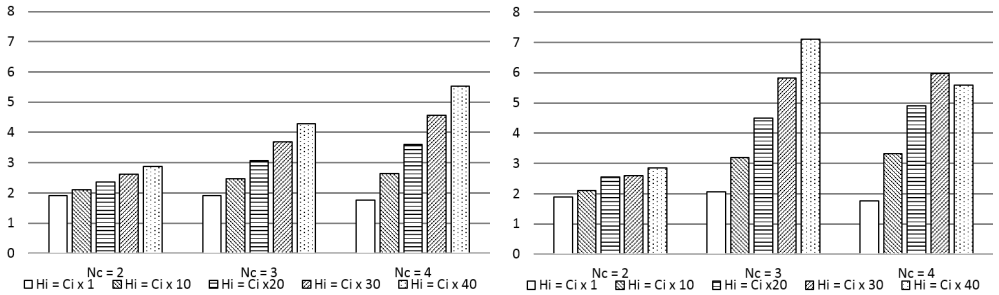
Another remark for a given number N_c can be made when we compare our framework to interference-oblivious search. The speed of the increase of workload, interference and slowdown respectively is slower in figures 4a, 5a and 6a than in figures 4b, 5b and 6b, which is an interesting quality for a system integrator. The difference of speed is more visible in the graphs of figures 4a and 5a than in figure 6a though. This is due to the fact that our optimization criteria is the workload reduction. In the future, it might be interesting to experiment multi-objective cost functions, to have multiple parameters guiding the search and solution selection.

VIII. Conclusions and Future Work

In this paper we have presented a derivation of the response time analysis in [12] to adapt it to IMA systems and heterogeneous multi-core platforms. The resulting response time analysis gives a sufficient schedulability test for IMA applications in heterogeneous multi-core environments. We then proposed to reuse this schedulability test to guide the design space exploration during the SW/HW integration design phase of IMA systems. To do so, we formulated an optimization problem to perform a timing-aware SW/HW allocation search. On the two implemented tests, our results showed that our approach enabled a real performance gain, by achieving up to 54.5% workload reduction, 54.9% memory interference reduction, and 50.4% slowdown reduction at runtime. In the future, we plan on refining the interference model, in order to take more than just main memory interference into account.

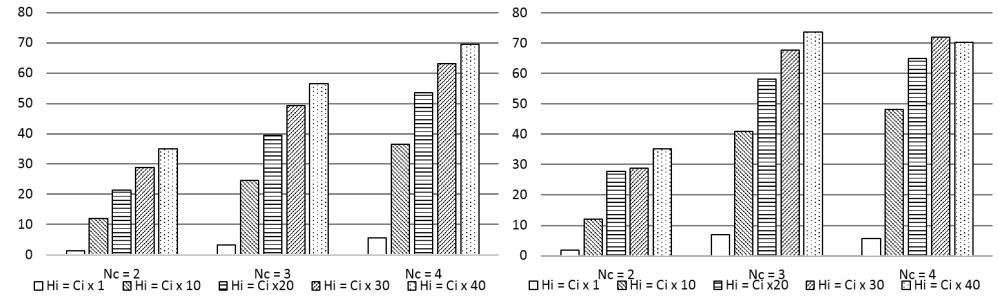
References

[1] CAST-32, "Multi-core Processors", 2014.
[2] "http://www.kalrayinc.com/".
[3] IBM ILOG, Cplex CP Optimizer. Website: http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/.
[4] RTCA Radio Technical Commission for Aeronautics, "Software Considerations in Airborne Systems and Equipment Certification", 1992.
[5] SAE International, "Aerospace Recommended Practice ARP4754 – Guidelines For Development Of Civil Aircraft and Systems",issued 1996; published 2010.
[6] Sebastian et al. Altmeyer. A generic and compositional framework for multicore response time analysis. In *Real-Time Network and Systems (RTNS), 2015 23th International Conference on*, pages 129–138. ACM, 2015.
[7] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: an open toolbox for adaptive WCET analysis. In *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010. Proceedings*, pages 35–46, 2010.



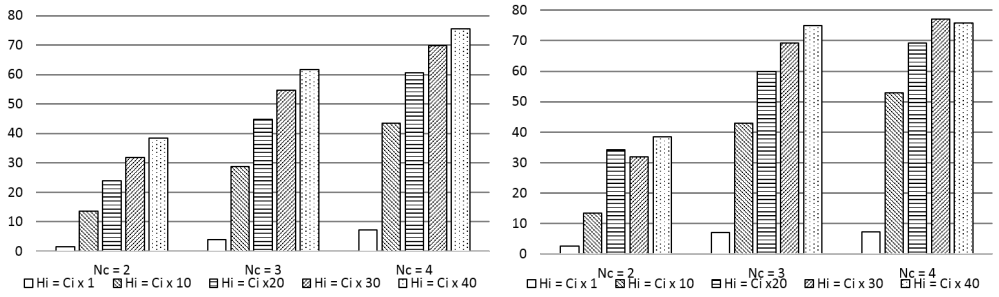
(a) in the solution obtained with our framework (b) in the solution obtained with classic approaches

Fig. 4: average workload per core (%)



(a) in the solution returned by our framework (b) in the solution returned by classic approaches

Fig. 5: average interference due to memory sharing (%)



(a) in the solution returned by our framework (b) in the solution returned by classic approaches

Fig. 6: average inter-task slowdown (%)

Fig. 7: Results of the second test: our approach (4a; 5a; 6a) versus classic approach (4b; 5b; 6b)

[8] Sanjoy K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *33rd International Conference on Parallel Processing (ICPP 2004), 15-18 August 2004, Montreal, Quebec, Canada*, pages 467–474, 2004.

[9] Richard Bradford, Shana Fliginger, Rockwell Collins, Cedar Rapids, Sibin Mohan, Rodolfo Pellizzoni, Cheolgi Kim, Marco Caccamo, Lui Sha, et al. Exploring the design space of ima system architectures. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5–E. IEEE, 2010.

[10] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[11] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[12] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark H. Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pages 145–154, 2014.

[13] Angeliki Kritikakou, Christine Rochange, Madeleine Faugère, Claire Pagetti, Matthieu Roy, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 139, 2014.

[14] Douglas Locke, Lee Lucas, and John Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[15] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Robert I. Davis, and Mateo Valero. IA^3: An interference aware allocation algorithm for multicore hard real-time systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 280–290, 2011.

[16] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 24–33, 2011.