Partition Scheduling on an IMA Platform with Strict Periodicity and Communication Delays

Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik

Partition Scheduling on an IMA Platform with Strict Periodicity and Communication Delays

Ahmad Al Sheikh[1,2], Olivier Brun[1,2] and Pierre-Emmanuel Hladik[1,2]
[1] CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
{ aalsheik, brun, pehladik }@laas.fr

Abstract

Integrated Modular Avionics (IMA) architectures employ a high-integrity, partitioned environment that hosts multiple avionics functions of different criticalities on a shared computing platform. IMA provides benefits of flexibility and scalability and allow for weight savings. However the complexity of the integration process is increased due to the sharing of resources. One of the main problem encountered during the integration process is to find a proper scheme for scheduling avionics functions onto offered processing units while ensuring at the same time execution, resource and safety demands.

In this paper, we present a MILP formulation for the scheduling of avionics functions on the resources of an IMA architecture. This mixed integer linear programming formulation takes into account resource constraints as well as temporal constraints related to ensuring a feasible schedule for the strictly periodic avionics functions. The objective function is to maximize the evolution margin for the functions, so that spare resources can be allocated to meet the resource demand growth of the hosted functions.

1. Introduction

The IMA architectures allow to execute avionics functions on a shared computing platform while respecting hard segregation constraints (spatial and temporal) between them and providing reliable communications. The processing units, called modules, can host several avionics applications and execute them independently. Segregation constraints are enforced using complete partitioning, not only on a functional basis, but also for what concerns memory allocations, in order to avoid interference between the independent applications and hence failures.

The IMA architecture is replacing the previous so-called federated architecture where resource sharing was rather limited and the dependencies between systems were well understood [17]. The IMA platform is generally used in conjunction with the ARINC[1] specifications, and in particular ARINC 653 [2] that defines the support for robust partitioning in on-board systems.

1.1. Partition segregation

As mentioned above, applications provide certain functionalities and need to be partitioned with respect to space (spatial partitioning) and time (temporal partitioning). A partition is therefore a program unit of the application designed to satisfy these partitioning constraints. The number of partitions per application is based on design related decisions and does not concern the work presented in this paper. For what follows, the notion of partition is going to be used and not that of application.

Each partition is allocated a set of spatial resources (such as system memory). To avoid error propagation, for instance, memory allocations must not overlap or even be shared between partitions. Hence, whilst configuring modules, the system designer has to ensure that these constraints are respected.

Each partition is also characterized by a time budget and a period. Partitions are executed strictly periodically, which means that the time separating two successive executions (instances) of the same partition is strictly equal to the partition period. Since partitions allocated to the the same module share the overall execution time (i.e. only one partition can be executed by the module at a time), the system designer has to ensure the temporal segregation of these partitions by a proper scheduling. In other words, he has to compute offsets (i.e. start time of the first instances) for the partitions such that there is no overlap in time between partition executions.

Due to the strict periodicity of partition executions, the schedule carried out on each module is by itself pe-

[1] Aeronautical Radio, Incorporated (ARINC), established in 1929, is a major provider of transport communications and systems engineering solutions for eight industries: aviation, airports, defense, government, healthcare, networks, security and transportation.

riodic [14] of a period defined by the least common multiple (LCM) of the module's partition periods–given that all partitions are released at once for the first time–. This period through which the schedule repeats with the same pattern is called the Major Frame (MAF). It provides a sufficient time-horizon for ensuring a correct temporal scheduling [6]. Figure 1 shows the execution of two partitions and the corresponding MAF.

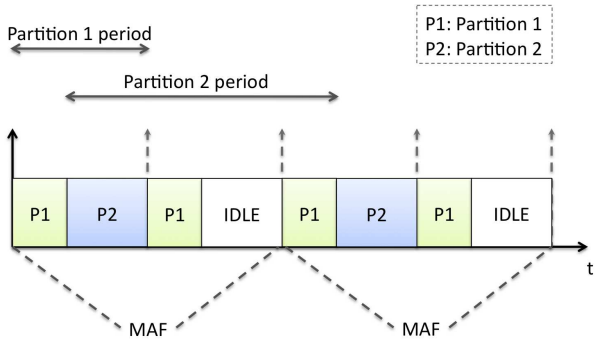

Figure 1. Partition execution in a MAF.

1.2. Partition communication

Interaction between partitions is limited to exchanging data through the communication backbone. Partitions resident on the same module communicate via locally dedicated API ports, whereas partitions resident on different modules communicate through the AFDX (Avionics Full Duplex Switched Ethernet) network which is specified by the ARINC 664 specification [3]. This network offers two-way communications based on the well known switched Ethernet. Further details on the communication backbone can be found in [3] and [7].

In airborne systems, communications occur in the framework of processing chains through which some kind of data is treated sequentially by a certain number of partitions. Data usually originate from a sensor or user input, and after processing by one or more partitions, a command is send to an actuator or screen (e.g. displaying altitude after analyzing altimeter readings). For ease of presentation, we shall assume that a processing chain starts at the first partition of the chain and ends at the last one. In addition, multicast is ignored and not included in our work. Figure 2 shows a simple example of a chain where partition 1 sends some kind of data to partition 2 which itself manipulates and transmits it back to partition 1. This partition finally sends a command or result to partition 3, the final consumer in the chain.

It is the responsibility of the system designer to allocate partitions to modules in such a way that the end-to-end delay of a processing chain does not exceed a certain predefined threshold.

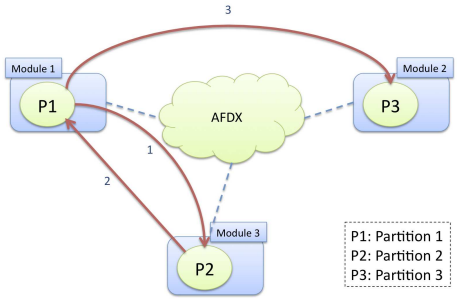

Figure 2. Processing chain consisting of 3 partitions.

1.3. Problem description

Currently, given the small number of partitions, partition mapping and scheduling is carried out by system experts (i.e. system designer) without particular aiding tools. Some of the simulation tools used for performance analysis of real-time systems and the associated schedulability analysis include, among others, Cheddar [19] and MAST [11]. In the near future, the number of partitions is expected to increase significantly, and it will become indispensable to automate the integration process. In particular, system designers will need specific tools to aid decision making for what concerns partition mapping and scheduling onto modules. Such tools will also be of utmost utility when adding new partitions to the system, which necessitates the computation of a new allocation and schedule.

The work presented in this paper proposes to automate the choices carried out by the designer for the spatial and temporal partitioning. As shown below, few works have discussed this automation in decision making in the context of IMA architectures [9].

1.4. Related work

Frameworks and models for allocating partitions to modules were studied in [17] and [5], based on directives derived from safety and operational reliability requirements. However, these works do not address the issue of scheduling the partitions allocated to each module.

Most studies on the scheduling problem consider loose periodicity where task instances are not obliged to be exactly one period apart. The authors in [8] presented a MILP formulation for non-periodic task scheduling on a set of processors with latency conditions. In [20], [21] and [22], the authors present a MILP formulation to solve a multi-processor periodic task scheduling problem, but with no emphasis on strict periodicity and considering task instances (on a time horizon equivalent to the LCM of all partition periods) as we have done in [4].

Our present work is closely related to works by [13] and [14]. Jan Korst [14] proposed a heuristic for minimizing the number of processors used for scheduling a set of periodic tasks. The strict periodicity case was handled and a condition for the schedulability of two periodic tasks was proposed. In [13], a similar problem where a multiprocessor schedule was searched to minimize the cycle time while considering strict periodicity, latency and precedence constraints. For this purpose, a heuristic was proposed alongside an exact branch and cut algorithm. Our approach differs from these earlier work in the desired objective and the implemented techniques.

1.5. Organization

Section 2 presents a mathematical model to our problem integrating allocation, scheduling and communication delay constraints. Section 3 proposes a method for simplifying our problem using graph theory. Section 4 presents some experimentations on the model. Section 5 finally concludes, spotting the light on the main results, and presents some perspective on ongoing and future work.

2. Formal definition of the problem

We consider a set $\Pi = \{\pi_1, \ldots, \pi_n\}$ of partitions that have to be mapped and scheduled on a set $\mathcal{P} = \{p_1, \ldots, p_m\}$ of parallel processors (modules).

Modeling a module. Each module $p_j \in \mathcal{P}$ is characterized by the available memory capacity M_j, and the maximum number of partitions K_j the module can host. Inter-module communication is characterized by the delay matrix $\Delta = \delta_{j,l}, \forall (p_j, p_l) \in \mathcal{P}^2$, where $\delta_{j,l}$ represents the maximum communication delay between modules p_j and p_l for $j \neq l$. It is assumed that $\delta_{j,j} = 0$.

For safety requirements, the modules are arranged into cabinets representing groups of modules sharing communication means. Each cabinet $\mathcal{C}_i$, $i = 1 \ldots C$, is powered independently, i.e. any cabinet can be powered off (thus shutting down corresponding modules) without affecting other cabinets and their contained modules. We suppose that $\mathcal{P} = \cup_{i=1}^{C} \mathcal{C}_i$, and $\forall i = 1 \ldots C, \forall j = 1 \ldots C$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$.

Modeling a partition. A partition π_i has the following attributes:

- T_i, the partition period,
- b_i, the time budget of the partition, i.e. the duration of partition execution,
- m_i, the memory budget of the partition, i.e. required memory capacity.

It is assumed that partitions can execute from the beginning of the MAF. In addition, the first temporal execution for a partition π_i must terminate before the end of the period. We let t_i denote the date at which partition π_i is executed for the first time. Due to strict periodicity, the k^{th} invocation of the partition is executed at time $t_i + (k-1)T_i$.

Two given partitions may be in exclusion for security reasons, i.e. they cannot be hosted by the same module. We let $\mathcal{E}$ denote the set of couples $(\pi_i, \pi_k) \in \Pi^2$ such that partitions π_i and π_k must be executed on different modules. Similarly, two partitions can be in exclusion on cabinet level, meaning that they cannot co-exist in the same cabinet, and we let $\mathcal{E}_c$ denote the set of couples $(\pi_i, \pi_k) \in \Pi^2$ such that partitions π_i and π_k must be executed in different cabinets.

Modeling a processing chain. Partitions may be involved in receiving and sending data along a processing chain (path) as described in section 1.2. We let $\Lambda = \{\lambda_1, \ldots, \lambda_o\}$ be the set of all processing chains in the system. Each processing chain λ_c is a sequence of partitions through which data is transferred, and $\lambda_c(k)$ represents the k^{th} partition to handle the data in the chain. A processing chain is characterized by the maximum tolerated end-to-end delay $\mathcal{L}_{\lambda_c}^{max}$ after which critical (or even hazardous) situations may be encountered.

Constraints. The following represent the constraints imposed on partition allocation and scheduling:

(C1): Each partition must be allocated to one and only one module.

(C2): For partitions belonging to the same module, overlap in execution times must be avoided.

(C3): The first instance of a partition has to be executed before the end of the corresponding period.

(C4): On each module, the sum of partitions' memory budgets must not exceed the module's memory capacity.

(C5): The number of partitions on each module must not exceed the maximum number of partitions the module can host.

(C6): If two partitions π_i and π_k are in exclusion at the module level, i.e. $(\pi_i, \pi_k) \in \mathcal{E}$, they must be allocated to two different modules.

(C7): If two partitions π_i and π_k are in exclusion at the cabinet level, i.e. $(\pi_i, \pi_k) \in \mathcal{E}_c$, they must be allocated to modules belonging to two different cabinets.

(C8): The partitions of each processing chain λ_c must be allocated in such a way that the resulting end-to-end delay does not exceed a certain defined upper bound $\mathcal{L}_{\lambda_c}^{max}$.

2.1. Defining the allocation and scheduling problem

The allocation problem amounts to finding a function that associates a module to each partition–all partition executions are in effect carried out on the same module–, such that all resource and scheduling constraints are verified for all modules. An allocation can be represented as a vector of binary variables $\mathbf{a} = (a_{i,j})$ such that:

$$a_{i,j} = \begin{cases} 1 & \text{if partition } \pi_i \text{ is assigned to module } p_j, \\ 0 & \text{otherwise.} \end{cases}$$

Only one module can be allocated to partition $\pi_i \in \Pi$ (constraint C1), hence:

$$\sum_{p_j \in \mathcal{P}} a_{i,j} = 1 \quad , \forall \pi_i \in \Pi. \tag{1}$$

Since partitions execute strictly periodically, a schedule is entirely defined by first partition execution dates which we represent by the vector $\mathbf{t} = (t_i)$.

2.2. Scheduling constraints

Note that the mth instance of partition π_i is executed in the time interval $I_m(t_i) = [t_i + (m-1)T_i, t_i + (m-1)T_i + b_i]$. According to constraint C2, if partitions π_i and π_k are allocated to the same module, any two invocations m and n of these partitions must not overlap in time. This can be expressed as follows:

$$\forall(\pi_i, \pi_k) \in \Pi^2, \forall m, n \in \mathbb{N}^*, \forall p_j \in \mathcal{P}$$
$$a_{ij} = a_{kj} = 1 \Rightarrow I_m(t_i) \cap I_n(t_k) = \emptyset.$$

Korst proposed in [14] (p.65) a necessary and sufficient condition to enssure that two partitions do not overlap in time.

Theorem 1. *Execution of partitions π_i and π_k on the same module never overlaps in time if and only if*

$$b_i \leq (t_k - t_i) \mod g_{i,k} \leq g_{i,k} - b_k, \tag{2}$$

where $g_{i,k}$ is the GCD of T_i and T_k.

This theorem assists in validating a schedule, i.e. there is no overlapping in execution times between partitions on modules, if and only if each couple of partitions (on a module) verify condition (2). This condition can be equivalently written as the following linear constraints:

$$\forall(\pi_i, \pi_k) \in \Pi^2, \forall p_j \in \mathcal{P},$$
$$b_i - (2 - a_{i,j} - a_{k,j})\, Z \leq (t_k - t_i) - q_{k,i}\, g_{k,i} \leq g_{i,k} - b_k + (2 - a_{i,j} - a_{k,j})\, Z, \tag{3}$$

where $q_{k,i}$ is an integer variable representing the quotient from the modulo operation in (2) (i.e. $q_{k,i} = \left\lfloor \frac{t_k - t_i}{g_{i,k}} \right\rfloor$) and Z is a large constant that ensures that the constraint is not active unless $a_{k,j} = a_{i,j} = 1$. The introduction of Z to obtain conditional constraints was inspired from [16].

To properly linearize the problem, the quotient $q_{k,i}$ must verify the following constraint for every partition couple allocated the same module,

$$0 < (t_k - t_i) - q_{k,i}\, g_{i,k} < g_{i,k}. \tag{4}$$

Clearly, given that b_i and b_k are positive, verifying constraints (3) imply the validity of (4). Hence, (4) represent redundant constraints.

Given the fact that

$$\frac{t_k - t_i}{g_{i,k}} - 1 < \left\lfloor \frac{t_k - t_i}{g_{i,k}} \right\rfloor \leq \frac{t_k - t_i}{g_{i,k}},$$

A bound for $q_{k,i}$ can be supplied:

$$\frac{-T_i + b_i}{g_{i,k}} - 1 < q_{k,i} \leq \frac{T_k - b_k}{g_{i,k}}. \tag{5}$$

In addition, a schedule $\mathbf{t}$ must satisfy the requirement that each partition π_i executes for the first time in the interval $[0, T_i]$ (constraint C3):

$$0 \leq t_i \leq T_i - b_i\,, \forall \pi_i \in \Pi. \tag{6}$$

In the following, we let $\zeta(\mathbf{a}, \mathbf{b})$ denote the set of all feasible schedules $\mathbf{t}$ satisfying (3) and (6), where $\mathbf{b} = (b_i)$ is the vector representing partition time budgets.

2.3. Resource constraints

Memory constraints C4 can be expressed as:

$$\sum_{\pi_i \in \Pi} a_{i,j}\, m_i \leq M_j, \forall\, p_j \in \mathcal{P}. \tag{7}$$

Constraints C5 on the number of hosted partitions per module are equivalent to:

$$\sum_{\pi_i \in \Pi} a_{i,j} \leq K_j,\ \forall p_j \in \mathcal{P}. \tag{8}$$

Exlusion constraints C5 prohibiting co-location of two partitions on the same module can be written as:

$$a_{i,j} \leq 1 - a_{k,j},\ \forall p_j \in \mathcal{P}, \forall(\pi_i, \pi_k) \in \mathcal{E}. \tag{9}$$

Whereas cabinet-level exclusion constraints C7 are modeled as:

$$a_{i,j} \leq 1 - \sum_{p \in \mathcal{C}_l} a_{k,p},\ \forall p_j \in \mathcal{C}_l, \forall l \leq C, \forall(\pi_i, \pi_k) \in \mathcal{E}_c. \tag{10}$$

Furthermore, it is possible to enrich the model by a new type of constraints to facilitate problem solving. These constraints are a result of a simple corollary of Theorem 1.

Corollary 1. *Two partitions π_i et π_k can be allocated the same module only if*

$$g_{i,k} \geq b_i + b_k \tag{11}$$

is verified.

Proof. This condition can be easily deduced from condition (2) of Theorem 1. □

It is consequently easy to add exclusion constraints similar to those of C6 between partitions that do not verify Corollary 1. We let $\mathcal{E}_a$ be the set of partition couples $(\pi_i, \pi_k) \in \Pi^2$ such that $b_i + b_k > g_{i,k}$, and add the following constraints:

$$a_{i,j} \leq 1 - a_{k,j}, \ \forall p_j \in \mathcal{P}, \forall (\pi_i, \pi_k) \in \mathcal{E}_a. \tag{12}$$

We denote by $\mathcal{A}$ the set of vectors $\mathbf{a}$ satisfying (1), (7)-(12). It represents the set of all possible allocations satisfying the resource constraints.

2.4. Communication delay constraints

Constraint C8 indicates that $\forall \lambda_c \in \Lambda$, the end-to-end communication delay $\mathcal{L}_{\lambda_c}$ must be less than or equal to $\mathcal{L}_{\lambda_c}^{max}$. The end-to-end communication delay can be expressed as:

$$\mathcal{L}_{\lambda_c} = \sum_{i=1}^{|\lambda_c|-1} L_{\lambda_c(i),\lambda_c(i+1)} + b_{\lambda_c(|\lambda_c|)} \leq \mathcal{L}_{\lambda_c}^{max}, \forall \lambda_c \in \Lambda, \tag{13}$$

where $L_{\lambda_c(i),\lambda_c(i+1)}$ represents the communication delay between the consecutive partitions $\lambda_c(i)$ and $\lambda_c(i+1)$ and $b_{\lambda_c(|\lambda_c|)}$ is the execution time for the last partition in the chain. (13) indicates that the end-to-end communication delay is the sum of consecutive partition-to-partition communication delays in the chain, i.e. each partition in the sequence sends data to the following one.

Partition-to-partition communication delay, between π_i and π_k for instance, represents the time needed to process data by π_i and send the result to π_k, either through the AFDX network, or locally through API ports. L_{π_i,π_k} can hence be written under the following form:

$$L_{\pi_i,\pi_k} = b_i + T_k + \sum_{p_j \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} a_{i,j} a_{k,l} \delta_{j,l} \quad , (\pi_i, \pi_k) \in \Pi^2 \tag{14}$$

The component $\sum_{p_j \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} a_{i,j} a_{k,l} \delta_{j,l}$ adds the inter-module transmission delay between p_j and p_l, where partitions π_i and π_k are respectively located, due to using the AFDX network. It should be noted that, if the two partitions are located on the same module, then this component will be zero; given that $\delta_{j,j} = 0$. The period T_k is also added to equation (14) indicating, with messages being read at the beginning of partition execution, (i) the worst-case for data acquisition after reception by the destination module when the two partitions are on different modules, and (ii) an upper bound on data reception when the two partitions are on the same module. Figure 3 represents the three delay components appearing in equation (14).

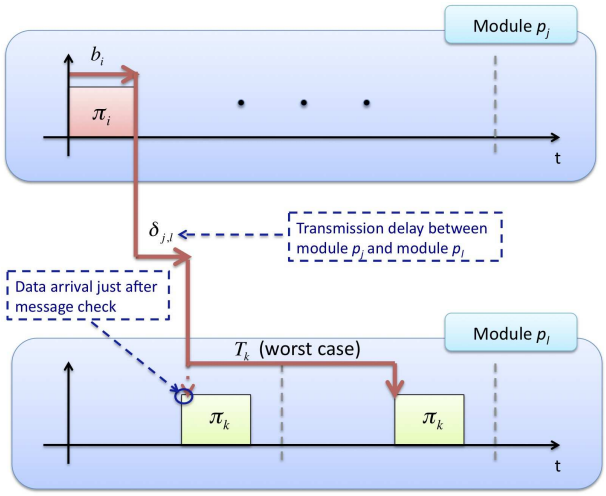

Figure 3. Delay components for partition couple communication.

The product $a_{i,j} a_{k,l}$ poses a problem on the linearity of the model (equation (14) is nonlinear). In order to utilize the MILP formulation, a reformulation has to be done [8]. Equation (14) becomes:

$$L_{\pi_i,\pi_k} = b_i + T_k + \sum_{p_j \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} z_{i,k,j,l} \delta_{j,l} \quad , (\pi_i, \pi_k) \in \Pi^2 \tag{15}$$

where the continuous variable $z_{i,k,j,l} \in [0,1]$ replaces the bilinear term $a_{i,j} a_{k,l}$ and has to satisfy the following linearization constraints,

$$z_{i,k,j,l} \leq a_{i,j}, \tag{16}$$
$$z_{i,k,j,l} \leq a_{k,l}, \tag{17}$$
$$z_{i,k,j,l} \geq -1 + a_{i,j} + a_{k,l}, \tag{18}$$

$\forall (\pi_i, \pi_k) \in \Pi^2, \forall (p_j, p_l) \in \mathcal{P}^2$, which guarantee that $z_{i,k,j,l} = a_{i,j} a_{k,l}$. The following constraints (19)-(20) represent some observations,

$$z_{i,k,j,l} = z_{k,i,l,j}, \tag{19}$$
$$z_{i,i,j,j} = a_{i,j}. \tag{20}$$

The rather large number of linearization constraints (16)-(18) ($3n^2m^2$ constraints) can slow down the solution process considerably. For this reason, we generate reduction constraints [15] by multiplying the allocation constraints (1) by $a_{k,l}$ to obtain the following,

$$\sum_{p_j \in \mathcal{P}} z_{i,k,j,l} = a_{k,l} \quad , \forall (\pi_i, \pi_k) \in \Pi^2, \forall p_l \in \mathcal{P}. \tag{21}$$

Proposition 2. *If the constraints* (1) *and* (21) *hold, provided* (19)*, we have* $z_{i,k,j,l} = a_{i,j}a_{k,l}$ *and in particular* $z_{i,k,j,l} \in \{0,1\}$*. Therefore constraints* (1), (19), (21) *imply the linearization constraints* (16)-(18)*, [15].*

We denote by Θ the set of vectors $\mathbf{a}$ satisfying (13), (19), (21) and $z_{i,k,j,l} \in [0,1]$. This set represents all possible allocations where the communication delay constraints are respected.

2.5. Formulation as a mixed integer linear program

A feasible solution to our problem is a couple $(\mathbf{a}, \mathbf{t})$, where $\mathbf{a}$ is an allocation and $\mathbf{t}$ is a schedule, satisfying constraints (1), (3), (6), (7)-(10), (12), (13), (19) and (21). In other words,

$$\mathbf{a} \in \mathcal{A} \cap \Theta,$$
$$\mathbf{t} \in \zeta(\mathbf{a}, \mathbf{b}).$$

In practice, it will often happen that several such solutions are feasible. In such a case, it is desirable to choose the one which ensures better evolution capacity for partitions, e.g. to permit adding new functionalities, without the need to reconsider all decisions (allocation and scheduling) already taken. Figure 4, for example, represents two possible solutions (S1) and (S2) for an allocation/scheduling problem with two modules and four partitions. It is obvious that the second solution (S2) offers more spare time in front of every partition execution, thus enabling us to augment partition execution times if necessary. In the first solution, however, it is impossible to evolve available partitions on the first module, for what concerns time budgets.

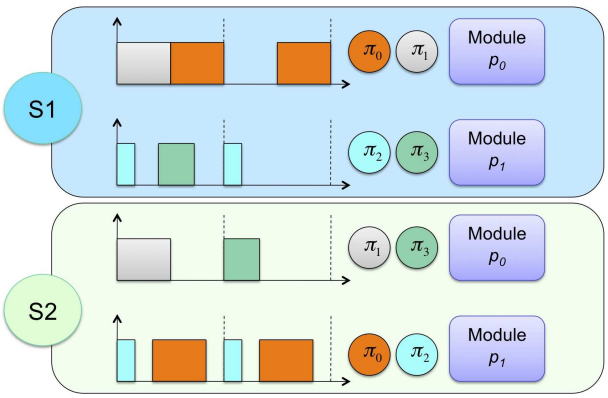

Figure 4. Choosing a better solution.

For this reason, the problem is expressed as an optimization problem. The objective is to find a solution that maximizes the time budgets which can be allocated to partitions. We aim for maximizing a coefficient α by which all initial partition time budgets (from problem input) can be multiplied, whilst respecting all system constraints.

Figure 5 shows the impact of α on a schedule constituted of two partitions on a given module. Hashed rectangles represent initial time budgets, whereas the larger filled ones represent the maximum allocable time budgets.

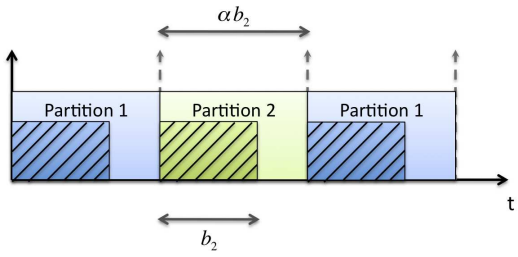

Figure 5. Scheduling using α.

The allocation and scheduling problem is hence formulated as follows,

$$\text{Max}_{\mathbf{a},\mathbf{t}}\alpha$$
$$\text{s.t.}$$
$$\mathbf{a} \in \mathcal{A} \cap \Theta,$$
$$\mathbf{t} \in \zeta(\mathbf{a}, \alpha\,\mathbf{b}).$$

In more detail, the complete formulation is:

$$\left\{\begin{array}{l}
\text{Max}_{\mathbf{a},\mathbf{t}}\alpha \\
s.t. \\
\sum_{p_j \in \mathcal{P}} a_{i,j} = 1, \quad \forall \pi_i \in \Pi, \\
\sum_{\pi_i \in \Pi} a_{i,j}\, m_i \leq M_j, \quad \forall p_j \in \mathcal{P}, \\
\sum_{\pi_i \in \Pi} a_{i,j} \leq K_j, \quad \forall p_j \in \mathcal{P}, \\
a_{i,j} \leq 1 - a_{k,j}, \quad \forall p_j \in \mathcal{P}, \forall (\pi_i, \pi_k) \in \mathcal{E}, \\
a_{i,j} \leq 1 - \sum_{p \in \mathcal{C}_l} a_{k,p}, \quad \forall l \leq C, \forall (\pi_i, \pi_k) \in \mathcal{E}_c, \\
a_{i,j} \leq 1 - a_{k,j}, \quad \forall p_j \in \mathcal{P}, \forall (\pi_i, \pi_k) \in \mathcal{E}_a, \\
t_i \geq 0, \quad \forall \pi_i \in \Pi, \\
t_i \leq T_i - \alpha\, b_i, \quad \forall \pi_i \in \Pi, \\
(t_k - t_i) - q_{k,i}\, g_{i,k} \geq \alpha\, b_i - (2 - a_{i,j} - a_{k,j})\, Z, \\
\qquad \forall p_j \in \mathcal{P}, \forall (\pi_i, \pi_k) \in \Pi^2, \\
(t_k - t_i) - q_{k,i}\, g_{i,k} \leq g_{i,k} - \alpha\, b_k + (2 - a_{i,j} - a_{k,j})\, Z, \\
\qquad \forall p_j \in \mathcal{P}, \forall (\pi_i, \pi_k) \in \Pi^2, \\
\sum_{i=1}^{|\lambda_c|-1} \left(b_{\lambda_c(i)} + T_{\lambda_c(i+1)} + \sum_{p_j \in \mathcal{P}} \sum_{p_l \in \mathcal{P}} z_{i,k,j,l} \delta_{j,l} \right) \\
\qquad + b_{\lambda_c(|\lambda_c|)} \leq \mathcal{L}^{max}_{\lambda_c}, \quad \forall \lambda_c \in \Lambda, \\
\sum_{p_j \in \mathcal{P}} z_{i,k,j,l} = a_{k,l}, \quad \forall (\pi_i, \pi_k) \in \Pi^2, \forall p_l \in \mathcal{P}, \\
z_{i,k,j,l} = z_{k,i,l,j}, \quad \forall (\pi_i, \pi_k) \in \Pi^2, \forall (p_j, p_l) \in \mathcal{P}, \\
q_{k,i} > \frac{-T_i + b_i}{g_{i,k}} - 1, \quad \forall (\pi_i, \pi_k) \in \Pi^2, \\
q_{k,i} \leq \frac{T_k - b_k}{g_{i,k}}, \quad \forall (\pi_i, \pi_k) \in \Pi^2, \\
a_{i,j} \in \{0,1\}, \quad \forall p_j \in \mathcal{P}, \forall \pi_i \in \Pi, \\
z_{i,k,j,l} \in [0,1], \quad \forall (\pi_i, \pi_k) \in \Pi^2, \forall (p_j, p_l) \in \mathcal{P}.
\end{array}\right.$$

We therefore search for maximizing the minimum evolution potential of the partitions in the system (temporal execution-wise). We will, hence, find partitions capable of evolving with a potential corresponding to this minimum value, and others capable with a more greater one.

3. Allocation initialization implementing graph theory

In this section, we search for reducing the number of problem variables, which may reduce the time required for solving the MILP presented in Section 2.5. For this, we propose a pretreatment phase based on graph theory. This pretreatment must guarantee that the optimal solution remains accessible. The method illustrated hereafter was inspired from the work carried out by Korst in [14] and is applied when all modules are identical, which is the case in reality. In the case where the modules are heterogeneous, the method no longer remains efficient.

A graph G, in which each node is associated to a partition, is constructed. An arc connecting two nodes is established if the two corresponding partitions are not involved in an exclusion (e.g. condition (11) is not verified). If two nodes are not connected, it is impossible to map the two corresponding partitions on the same module.

The connected components of G represent partition sets that must be mapped onto different modules. The search for a Maximal Independent Set (MIS) in G [18], allows obtaining a set of totally independent partitions, that must consequently be allocated different modules. Algorithms for finding connected components and maximal independent sets are not exposed in this paper and are based on graph theory basics and some of the algorithms developed in [1] and [18].

For every connected component of G, partitions corresponding to nodes in the MIS are placed on distinct modules–since partitions of this set cannot be placed on the same module–, this allows us to minimize the number of problem allocation variables and accelerate the resolution process.

Figure 6 represents the graph representation for a set of six partitions. Each node couple is connected by an arc if no exclusion exists between the corresponding partitions, according to previously defined exclusion constraints (Section 2.3). For example, partitions 1 and 2 are likely to be placed on the same module, while partitions 1 and 3 surely do not co-exist on the same one. Nodes 1, 3 and 6 are found to represent a MIS, hence, by fixing the allocations for partitions 1, 3 and 6 on three different modules, we are able to minimize in effect the problem's allocation variables from six to three.

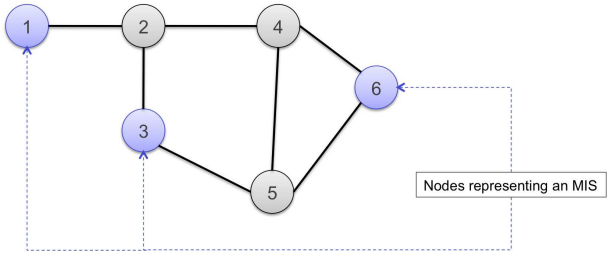

Figure 6. Graph representation for a set of partitions.

4. Experimentation

We hereafter present some experimentation on the optimization problem presented in Section 2 along with the pretreatment proposed in Section 3. The MILPs where solved using the solver CPLEX [12] from the ILOG community on an eight core system–each rated at 3.2GHz–with 8MB cache and 32GB system RAM. The solver was noticed to utilize one of the eight cores and 4.7GB of Ram at most.

Twenty-three problems, in which the number of modules and partitions varied from 1 to 4 and 6 to 80 respectively, are demonstrated in Table 1. Partition periods were taken in time intervals as shown in the same table, whereas partition time budgets were considered of small order as compared to the periods. Partition periods were generated in a manner limiting the LCM between them based on a method similar to that indicated in [10], and hence, obtaining a not so large MAF on each module. Communication was considered so that 40 to 60 percent of partitions were involved in some kind of processing chain (acquisition and transmission of data). Partition exclusions were generated such that 40 percent of partitions at maximum were involved. Memory capacity and partition count on modules were generated to impose some constraints on the problems. Memory requirements for partitions, for instance, were generated such that their sum equals the total memory capacities of modules multiplied by a certain utilisation factor (taken as 50%).

In addition, the problem was solved by considering time budgets ($t_i\,, \forall \pi_i \in \Pi$) as integer and continuous variables respectively. Theoretically, solving for t_i continuous should give some relaxation to the problem, and consequently, we should expect faster computation times (CpuTime).

Table 1 is divided into two parts, experiments 1 to 11 where periods ranged up to couple of hundreds, and experiments 12 to 23 where periods ranged up to couple of thousands. Values of α demonstrate the optimization carried out, where for example in experiments 4 and 16, partition time budgets can be increased by half ($\alpha = 151\%$) without affecting the proposed allocation and schedule.

Computation times for experiments 1 to 7 and 13 to 17 were insignificant, whereas for the rest, and depending on

Table 1. Time required for solving the mixed integer linear program

Experiment	Module count	Partition count	Partition period range	CpuTime t_i : int	CpuTime t_i : real	α t_i : int	α t_i : real
1	3	6	[50,150]	0.04s.	0.01s.	100%	111,1%
2	3	6	[60,300]	0.05s.	0.01s.	135.938%	136.364%
3	3	6	[40,350]	0.03s.	0.01s.	134.615%	137.931%
4	3	6	[120,400]	0.03s.	0.01s.	151.613%	151.899%
5	4	10	[30,360]	0.1s.	0.07s.	145%	145.161%
6	4	10	[90,400]	0.3s.	0.08s.	114.286%	115.385%
7	4	10	[80,400]	0.22s.	0.05s.	108%	108.108%
8	4	20	[20,360]	2hr.	1s.	115.385%	115.385%
9	4	20	[10,270]	5.57hr.	6.5s.	110%	125%
10	4	30	[10,720]	24hr.+	24hr.+	(200%)	(200%)
11	4	80	[20,450]	24hr.+	24hr.+	-	-
12	1	10	[600,2400]	14.53mn.	3.14mn.	136%	136.364%
13	3	6	[500,1500]	0.02s.	0.01s.	126.126%	126.126%
14	3	6	[600,3000]	0.2s.	0.01s.	136.25%	136.364%
15	3	6	[400,3500]	0.21s.	0.01s.	137.692%	137.931%
16	3	6	[1200,4000]	0.62s.	0.01s.	151.875%	151.899%
17	4	10	[300,3600]	0.09s.	0.07s.	145.116%	145.161%
18	4	10	[900,4000]	1.22s.	0.08s.	115.352%	115.385%
19	4	10	[800,4000]	0.38s.	0.05s.	108%	108.108%
20	4	20	[200,3600]	12.07s.	2.36s.	115.385%	115.385%
21	4	20	[100,2700]	6hr.	15.7mn.	123.333%	125%
22	4	30	[100,7200]	24hr.+	24hr.+	(200%)	(200%)
23	4	80	[200,4500]	24hr.+	24hr.+	-	-

the complexity of the problem, became more important. The difference in computation times between experiments 12 and 17, with the latter being presumably harder, is a clear indication that the complexity of a problem arises not only from the number of components (modules and partitions) but also from any of the component attributes (such as partition periods). Experiments 10, 11, 22 and 23 were stopped after 24 hours of execution. Experiments 10 and 22 found feasible solutions but could not prove optimality in the indicated time. Experiments 11 and 23 were very complex that no solution was even found for the same amount of time. This brings us to the point that real complex systems and even future ones, where number of modules and partitions may be of importance, may pose a problem for the MILP formulation.

It should be also noted that, though not presented in Table 1, the proposed pretreatment phase had a great impact on reducing the computation times as can be clearly seen in Table 2, representing a set of five separate problems.

What seems to be interesting however, is that solving for continuous values of time budgets improves computation times (e.g. from about 5 and a half hours to 6.5 seconds in experiment 9) while obtaining α values that are close to those from solving the original problem (without relaxation of time budgets), especially for greater order of periods. This enables us to propose a heuristic that

Table 2. The impact of pretreatment phase

Exper.	Module count	Partition count	CpuTime (no pretreat.)	CpuTime (pretreat.)
a	3	6	0.31s.	0.01s.
b	4	8	0.78s.	0.25s.
c	4	8	3.87s.	0.77s.
d	8	16	1.9hr.	29s.
e	8	16	5days	11mn.

is based on solving the problem for $t_i, \forall \pi_i \in \Pi$, continuous, then by guarding the computed allocations, we resolve sub-MILP problems based on finding an optimal scheduling on each module. We admit that this two phase heuristic may be inefficient in some cases where solving with relaxation, as was shown in Table 1, may require a colossal amount of time for computation. Nevertheless it seemed interesting to investigate this method and test it on a simple set of experiments as shown in Table 3.

We can notice that the application of this two phase method gave very close optimal results, which means that this heuristic may give near optimal results in more acceptable computation times.

5. Conclusion

In this paper, we have proposed a MILP formulation model for the mapping and scheduling problem of avion-

Table 3. Two phase heuristic

Exper.	CpuTime original	CpuTime two phase	α original	α two phase
7	0.22s.	0.05s.	108 %	108 %
8	2hr.	10.37s.	115.385 %	114.286 %
9	5.57hr.	26.68mn.	110 %	100 %
15	0.21s.	0.01s.	137.692 %	136.667 %
16	0.62s.	0.01s.	151.875 %	151.875 %
18	1.22s.	0.8s.	115.352 %	115.352 %
21	6hr.	16mn.	123.333 %	123.333 %

ics applications. The particularity of this work resides in the strict periodicity of partition executions and the diversity of system constraints. The experimentations have shown the efficiency of the proposed approach, including a pretreatment phase where some initialization can be made. Computation times, however, for quite complex architectures with a significant number of components, can become somewhat sluggish.

In the future, it would be normal to find several hundreds of partitions running on several dozens of modules. The exact method presented in this paper will be used to assess the performance of an adapted heuristic that should be capable of handling more complex systems efficiently. Our interest now is to develop such a heuristic, referring to local search algorithms.

Acknowledgement

The work presented in this paper was conducted under the research project SATRIMMAP (SAfety and Time Critical Middleware for future Modular Avionics Platform) which is supported by the French National Agency for Research (ANR).

References

[1] R. Ahuja, T. Magnanti, J. Orlin, and K. Weihe. *Network flows: theory, algorithms, and applications*. Prentice hall Englewood Cliffs, NJ, 1993.

[2] Airlines electronic engineering committee (AEEC). Avionics application software standard interface. *ARINC specification 653*, (part 1), 2006.

[3] Airlines electronic engineering committee (AEEC). Avionics full duplex switched ethernet (afdx) network. *ARINC specification 664*, (part 7), 2009.

[4] A. Al-Sheikh, O. Brun, and P.-E. Hladik. Decision support for task mapping on ima architecture. In *Junior Researcher Workshop on Real-Time Computing (JRWRTC2009), pp.31-34*, October 2009.

[5] P. Bieber, J. Bodeveix, C. Castel, D. Doose, M. Filali, F. Minot, and C. Pralet. Constraint-based Design of Avionics Platform–Preliminary Design Exploration. In *4th European Congress ERTS Embedded Real Time Software*, Toulouse, 2008.

[6] A. M. Campbell and J. R. Hardin. Vehicle minimization for periodic deliveries. *European Journal of Operational Research*, 165(3):668 – 684, 2005.

[7] H. Charara and C. Fraboul. Modelling and simulation of an avionics full duplex switched Ethernet. In *Telecommunications, 2005. Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop. AICT/SAPIR/ELETE 2005. Proceedings*, pages 207–212, 2005.

[8] T. Davidovic, L. Liberti, N. Maculan, and N. Mladenovic. Mathematical programming-based approach to scheduling of communicating tasks. Technical Report G-2004-99, Cahiers du GERAD, December 2004.

[9] C. Fraboul and F. Martin. Modeling advanced modular avionics architectures for early real-time performance analysis. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 0:181, 1999.

[10] J. Goossens and C. MaCq. Limitation of the hyper-period in real-time periodic task set generation. *Proceedings of the RTS Embedded System (RTS'01)*, pages 133–148, 2001.

[11] M. Harbour, J. García, J. Gutiérrez, and J. Moyano. MAST: Modeling and analysis suite for real time applications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, page 0125. Published by the IEEE Computer Society, 2001.

[12] ILOG CPLEX. `http://www.ilog.com/products/cplex/`.

[13] O. Kermia and Y. Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *Proceedings of ISCA 20th international conference on Parallel and Distributed Computing Systems, PDCS*, volume 7. Citeseer, 2007.

[14] J. Korst. *Periodic multiprocessor scheduling*. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

[15] L. Liberti. Automatic reformulation of bilinear MINLPs. Technical Report 2004.24, DEI, Politecnico di Milano, July 2004.

[16] W. Roux. *Une approche coherente pour la planification et l'ordonnacement de systèmes de production complexes*. PhD thesis, LAAS-CNRS, report n^o 97248, 1997.

[17] L. Sagaspe and P. Bieber. Constraint-based design and allocation of shared avionics resources. In *26th AIAA-IEEE Digital Avionics Systems Conference*, Dallas, 2007.

[18] A. Sharieh, W. Al_Rawagepfeh, M. Mahafzah, and A. Al Dahamsheh. An Algorithm for Finding Maximum Independent Set in a Graph. *European Journal of Scientific Research*, 23(4):586–596, 2008.

[19] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada: The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies*, pages 1–8. ACM, 2004.

[20] S. Thanikesavan and U. Killat. Global scheduling of periodic tasks in a decentralized real-time control system. In *2004 IEEE International Workshop on Factory Communication Systems, 2004. Proceedings*, pages 307–310, 2004.

[21] S. Thanikesavan and U. Killat. Static scheduling of periodic tasks in a decentralized real-time control system using an ilp. In *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops*, page 639, Washington, DC, USA, 2005. IEEE Computer Society.

[22] S. Thanikesavan and U. Killat. A satisficing momip framework for reliable real-time application scheduling. In *DASC '06: Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 187–194, Washington, DC, USA, 2006. IEEE Computer Society.