Mapping real-time communicating tasks on a distributed IMA architecture

Émilie Bérard-Deroche, Jean-Luc Scharbarg, Christian Fraboul

▶ **To cite this version:**

Émilie Bérard-Deroche, Jean-Luc Scharbarg, Christian Fraboul. Mapping real-time communicating tasks on a distributed IMA architecture. 21st International Conference on Emerging Technologies and Factory Automation (ETFA 2016), Sep 2016, Berlin, Germany. pp.1-8. hal-02348198

HAL Id: hal-02348198
https://hal.science/hal-02348198

Submitted on 5 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/22338

Official URL

DOI : https://doi.org/10.1109/ETFA.2016.7733586

To cite this version: Deroche, Emilie and Scharbarg, Jean-Luc and Fraboul, Christian *Mapping real-time communicating tasks on a distributed IMA architecture.* (2016) In: 21st International Conference on Emerging Technologies and Factory Automation (ETFA 2016) (ETFA), 6 September 2016 - 9 September 2016 (Berlin, Germany).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Mapping real-time communicating tasks on a distributed IMA architecture

Emilie Deroche*‡, Jean-Luc Scharbarg‡ and Christian Fraboul‡
*Airbus Group SAS, Airbus Group Innovations, 18, rue Marius Terce, 31025 Toulouse, France
Email: Emilie.Deroche@airbus.com
‡University of Toulouse, IRIT-INPT/ENSEEIHT, 2, rue Charles Camichel, 31000 Toulouse, France
Email: {Emilie.Deroche, Jean-Luc.Scharbarg, Christian.Fraboul}@enseeiht.fr

***Abstract*—Current avionics architectures implemented on large aircraft use complex processors, which are shared by many avionics applications according Integrated Modular Avionics (IMA) concepts. Using less complex processors on smaller aircraft such as helicopters leads to a distributed IMA architecture. Allocation of the avionics applications on a distributed architecture has to deal with two main challenges. A first problem is about the feasibility of a static allocation of partitions on each processing element. The second problem is the worst-case end-to-end communication delay analysis: due to the scheduling of partitions on processing elements which are not synchronized, some allocation schemes are not valid. This paper first presents a mapping algorithm using an integrated approach taking into account these two issues. In a second step, we evaluate, on a realistic helicopter case study, the feasibility of mapping a given application on a variable number of processing elements. Finally, we present a scalability analysis of the proposed mapping algorithm.**

I. Introduction

Helicopter and aircraft industries attempt to reduce weight and power consumption. The Integrated Modular Avionics (IMA) architecture is a first step in this direction: instead of having one function per processor like in federated architectures, several functions share the same processor. Most of the time, communication means are also shared to reduce the number and the weight of cables [1] [2].

Avionics systems are composed of an increasing number of more and more complex functions. It leads to avionics architectures composed of powerful and complex processors. Such processors cannot be used in small aircraft and helicopters (cost, space, ...).

A classical solution to deal with this problem is to have a larger number of (possibly less complex) processors that can be distributed among the whole helicopter. The problem is then to distribute avionics functions on these processors in such a way that timing properties are guaranteed. The first constraint is to guarantee that the set of functions allocated to a given processor is schedulable on this processor. Such scheduling is built statically in IMA. The second constraint is to guarantee that end-to-end delay constraints are not exceeded: for example, the delay between the completion of the data acquisition process and the recording of the resulting data in the non volatile crash-protected recording medium should be guaranteed [3]. When data acquisition is done by function f_1 and recording is done by function f_2, delay between start of f_1 and end of f_2 should not exceed 0.5 seconds. If f_1 and f_2 are allocated to different processors, communication delay has to be considered in the analysis.

978-1-5090-1314-2/16/$31.00

Thus distribution of avionics functions has to deal with both scheduling of partitions and end-to-end delay analysis. Previous work has been devoted to similar problems.

Many of them take into account allocation problems. The allocation can be done on-line, i.e. partitions are executed on the first available processor, or off-line, i.e. a planning is done on the basis of the temporal characteristics of partitions. For certification and reliability reasons, the scheduling is done off-line. In [4] authors proceed in two steps: first they schedule partitions on execution nodes, second they route flows on the avionics network. Work proposed in [5] deals with the allocation of partitions trying to minimize the communication costs.

Several approaches integrate temporal analysis of embedded systems. Many works have been devoted to the worst-case delay analysis on AFDX network [6] [7], the temporal requirements verification of systems [8], and the complexity of communication delays [9]. ASIIST is a tool that has been proposed in [10]. It permits to verify the scheduling of partitions and to calculate bus delays according to a predefined allocation of partitions and mapping of communications.

In these previous works, each part has been studied separately: first an allocation is defined, then a possible scheduling is found and finally end-to-end delay constraints are checked. If one of these steps is not validated, a new scheduling or a new allocation has to be defined again.

The contribution of this paper is to propose a mapping algorithm, which copes altogether with static allocation of partitions and with a guaranteed applicative end-to-end delay.

Section II gives main modeling assumptions. Section III explains the proposed mapping algorithm on an illustrative example. A realistic case study is analyzed in Section IV. Section V addresses scalability issue of the mapping algorithm. Section VI concludes the paper and proposes directions for future research.

Fig. 1. APEX channel

II. Main modeling assumptions

The goal is to allocate and schedule a set of partitions $\mathcal{P}$ on a set of processing elements $\mathcal{E}$ taking into account applicative temporal constraints. First, partitions attributes and IMA scheduling are specified in section II-A. Then, end-to-end applicative constraints are defined thanks to a communication semantic. Finally, computation of end-to-end communication delays according to this semantic is developed in section II-C.

A. Partition and scheduling description

Let $\mathcal{P} = \{P_1, ..., P_n\}$ be the set of partitions to be distributed. Each partition P_i is characterized by the couple (T_i, C_i): T_i is the period of the partition P_i and C_i is its worst case execution time (WCET). Each partition P_i is strictly periodic: the duration between two consecutive executions is exactly T_i. Periods are assumed harmonic. This is a classical assumption for avionics systems. Each WCET has to verify:

$$0 \leq C_i \leq T_i \tag{1}$$

Each partition implements a read-execution-write semantic: at the beginning of its execution, the partition reads data from its input ports; at the end of its execution, it writes data in its output ports. Partitions exchange data through virtual channels, called APEX channel in the standard ARINC 653 [11], as depicted in Figure 1. At any time only the last data written in the buffer is considered (sampling port).

Let $\mathcal{E} = \{PE_1, ..., PE_{maxNbPE}\}$ be the set of processing elements PE. A subset of $\mathcal{P}$, noted p, is allocated on a processing element PE. We first have to insure that the temporal load of PE_j is not exceeded:

$$\sum_{P_i \in p} \frac{C_i}{T_i} \leq 1 \tag{2}$$

Second, partitions in p have to be scheduled on PE. IMA scheduling on each processing element PE_j is based on a MAjor Frame MAF_j: it statically defines the execution pattern of all the partitions allocated to PE_j. This static scheduling is done off-line, with no pre-emption allowed. Such a MAF is depicted in Figure 2. MAF length of PE_j is the least common multiple of the periods of partitions in p. Since periods are harmonic, MAF length is the largest period among partitions on p:

$$Hyp_j = LCM_{P_i \in p}(T_i) = \max_{P_i \in p}(T_i) \tag{3}$$

In Figure 2, p includes two partitions P_1 and P_2. MAF length is the period of P_2.

Each MAjor Frame MAF_j is composed of a subset of periodic intervals s of length $t_{interval}$:

$$t_{interval} = \min_{P_i \subset PE_j}(T_i) \tag{4}$$

Thus we have:

$$MAF_j = \{s_1, ..., s_{\frac{Hyp_j}{t_{interval}}}\} \tag{5}$$

Each partition P_k in p is allocated one slot every $\frac{T_k}{t_{interval}}$ interval. In Figure 2, P_1 is allocated one slot in every interval, while P_2 is allocated one slot every two intervals.

r_i defines the initial time of the first slot allocated to partition P_i in MAF_j.

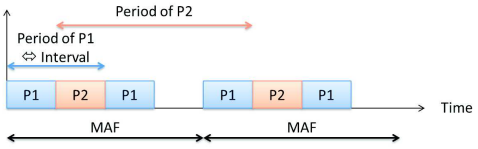

Fig. 2. Execution pattern

The load in each interval of a MAF should not exceed $t_{interval}$:

$$\sum_{P_i \subset Interval_j} C_i \leq t_{interval} \tag{6}$$

B. Communication and end-to-end delay constraints

As previously mentioned, partitions exchange data. A communicating partitions chain is an ordered sequence of partitions with an end-to-end applicative constraint from the first partition of the chain to the last one. Typically the first partition transmits a message to the second one, which transmits a message to the third one and so on.

More formally, a set of communicating partitions chains $\mathcal{C} = \{c_1, ..., c_m\}$ is considered. Each chain c_i is a n-uplet composed of a subset p of $\mathcal{P}$:

$$c_i = \{P_1, ..., P_k\} \tag{7}$$

The end-to-end delay constraint of chain c_i is denoted $D_{c_i,max}$. It is defined according to the Button-to-action (BTA) semantic defined in [12] as depicted in 3: it consists in finding the worst-case duration between the generation of a data by the first partition in the chain and its first utilization by the destination partition.

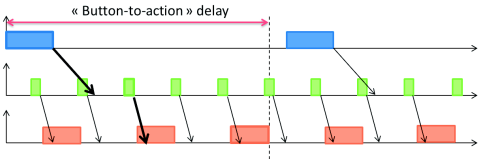

Fig. 3. Button-to-Action semantic

C. Allocation and end-to-end delay validation

An allocation a_i is defined as the set of partitions allocated to each processing element as well as the MAF of each processing element. An allocation is total if all the partitions in $\mathbb{P}$ are allocated to a processor. Otherwise, the allocation is partial.

Fig. 4. Delay between two communicating partitions in the same processing element

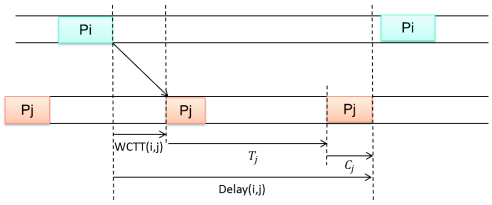

Fig. 5. Delay between two communicating partitions in different processing elements

An allocation a_i is valid if:

- all the partitions allocated to a processing element PE_k are schedulable on PE_k,
- the end-to-end delay constraints D_{c_i} are guaranteed for all the communication chains.

Given an allocation a_m, this delay is calculated for all the chains c_k. It consists in summing all the WCETs of the partitions contained in c_k and each pair of consecutive partitions (P_i, P_j) in c_k, the worst-case distance, noted $delay(i,j)$, between the end of P_i and the beginning of P_j:

$$D_{c_k,a_m} = \sum_{(P_i,P_j)\in c_k,(P_i,P_j)\subset a_m} \big(delay(i,j)\big) + \sum_{P_i\in c_k} C_i \quad (8)$$

Worst-case distance $delay(i,j)$ depends on the allocation.

If the two communicating partitions P_i and P_j are on the same processor, the communication is local: the delay between the source and the destination partition execution depends on the MAF as depicted in figure 4. In this case, the first execution of the destination partition just after the source one has to be considered. Let r_i (respectively r_j) be the release time in the MAF of the source (respectively destination) partition. Delay is given by:

$$delay(i,j) = \min_{k\in\mathbb{N}}(r_j + k\cdot T_j - (r_i + C_i)) \geq 0 \quad (9)$$

If partitions are on different processors, the communication is remote: transmission latency has to be taken into account in the delay calculation, i.e. the Worst Case Traversal Time ($WCTT(i,j)$) of a message generated by the source partition P_i to the destination one P_j as illustrated in figure 5. We also have to take into account the worst-case delay between the reception of the message and the next execution of the destination partition. In the worst case, the message arrives just after the beginning of the destination partition execution and it has to wait for nearly one period of the destination partition P_j. Overall the worst case delay between the end of the source partition execution and the destination one is given by:

$$delay(i,j) = WCTT(i,j) + T_j \quad (10)$$

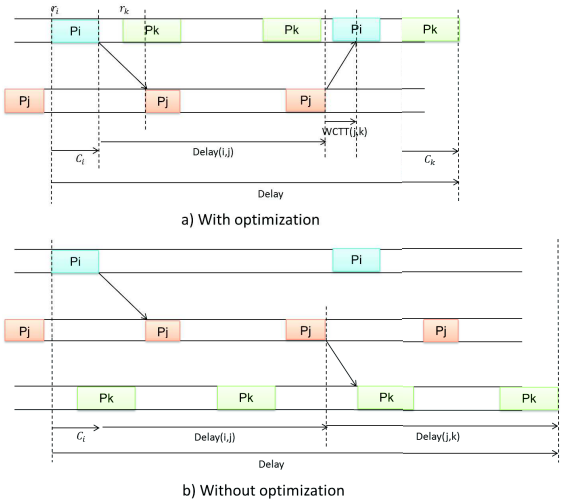

Fig. 6. Computation of the delay in case of a loop

This worst-case delay can be reduced when a communication chain comes back to a previous processing element as depicted in figure 6. In this situation, the button-to-action delay depends on the first partition allocated to the same processor as the destination one.

Let:

- $lastCouple(c_k)$ be the last couple of partitions of the chain c_k,
- c'_k be a communication chain with $c'_k \subset c_k$.

The first and the last partitions of c'_k are allocated to the same processor. The aim is to find the next execution of the destination partition following the reception of the data by the processor as illustrated in figure 6. The delay calculus is done in two steps: the first one consists in calculating the arrival time of the data back to the first processor and second finding the next execution of the destination partition thanks to the previous calculus. Then, the BTA delay is given by:

$$\begin{aligned} if\exists k\in\mathbb{N} \quad & / \\ 0 \leq \min_{k\in\mathbb{N}} \quad & \Big(\quad r_j - r_i + k\cdot T_j + C_j \\ & - \quad \Big(\sum_{(P_m,P_n)\in c'_k - lastCouple(c'_k)} delay(m,n) \\ & + \quad C_i + WCTT(lastCouple(c'_k))\Big)\Big) \end{aligned}$$

$$delay_{c'_k} = r_j - r_i + k\cdot T_j + C_j \quad (11)$$

III. Mapping algorithm illustration

We illustrate the mapping algorithm on a set of partitions $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ which requires to be allocated to up to 3 processing elements, i.e. $\mathcal{E} = \{PE_1, PE_2, PE_3\}$. The characteristics of the partitions are depicted in Table I, which gives for each partition its WCET (equal to C_i) and its period (T_i).

TABLE I
REAL-TIME SPECIFICATIONS OF PARTITIONS OF THE ILLUSTRATIVE EXAMPLE

Partitions	C_i (ms)	T_i (ms)
P_1	3	10
P_2	2	10
P_3	2	20
P_4	4	40
P_5	1	40
P_6	4	40

Three communication chains are defined as follows:

$$\mathcal{C} = \{c_1, c_2, c_3\}$$
$$c_1 = \{P_1, P_2, P_3\}, D_{c_1,max} = 30ms$$
$$c_2 = \{P_2, P_5\}, D_{c_2,max} = 40ms$$
$$c_3 = \{P_4, P_5, P_6\}, D_{c_3,max} = 60ms$$

We assume, on this example, that if the communication between two communicating partitions is remote, the WCTT is equal to 5 ms, and 0 if the communication is local.

The partitions are first sorted by increasing periods. Valid global allocations of all partitions on the available processing elements are found thanks to an in-depth research across a tree whose nodes correspond to partial allocations of a limited number of partitions on a given amount of processing elements.

Each new child node consists in allocating a new partition on an increased number of processing elements. The number of child nodes is given by the lowest number between the available processing elements $maxNbPE$ and the used processing elements plus one. A partial allocation is valid when a child node validates the scheduling and the temporal constraints at the same time. We generate such child nodes until it remains a non-allocated partition.

The in-depth search algorithm is illustrated on the above example and depicted in figure7.

The first step consists in allocating the first partition P_1 to the first processing element PE_1 in the partial allocation a_1. The MAF is composed of 1 slot which lasts 10 ms, i.e. the period of the partition P_1. The MAF is composed of only one slot and the release time r_1 is 0:

$$MAF_1 = \{s_1\} \tag{12}$$
$$s_{1,MAF_1} = \{r_1 = 0\} \tag{13}$$

As there is only one partition allocated, the current BTA delays D_{c_1,a_1}, D_{c_2,a_1} and D_{c_3,a_1} are respectively equal to 7, 3, 9. The MAF satisfies both scheduling and temporal constraints, so the partial allocation a_1 is valid.

The second step consists in allocating the partition P_2 using the previous valid partial allocation, a_1. Here, P_2 can be allocated to either PE_1, or to a new processing element PE_2: two new partial allocation children are generated, a_2 and a_3.

If we take the partial allocation a_2, the partition P_2 has the same period as P_1 and can be allocated before or after P_1 on the processing element PE_1. When we build the corresponding MAF, P_2 can be allocated in the same slot:

$$s_{1,MAF_1} = \{r_1 = 0, r_2 = 3\}. \tag{14}$$

Adding this partition, we have to verify the end-to-end delay of the partial chain c_1, D_{c_1,a_2}, that can be computed by:

$$D_{c_1,a_2} = delay(1,2) + C_1 + C_2 + C_3 \tag{15}$$

$delay(1,2)$ corresponds to the minimum distance between the end of the partition execution P_1 and the beginning of P_2 as described in figure 4. In this case, k is equal to 0.

$$delay(1,2) = \min_{k\in\mathbb{N}} \big(r_2 + k\cdot T_2 - (r_1 + C_1)\big) = 3 + 0*10 - 3 = 0 \tag{16}$$

thus, we have:

$$\begin{aligned} D_{c_1,a_2} &= delay(1,2) + C_1 + C_2 + C_3 \\ &= 0 + 3 + 2 + 2 = 7 \leq D_{c_1,max} \end{aligned} \tag{17}$$

As this partial allocation is valid, we try to add the partition P_3 either on this PE_1 processing element or on the new one PE_2. So we have two potential partial allocations: a_4 and a_5. On node a_4, P_3 is allocated to the same processing element as P_1 and P_2 but it does not have the same period. The MAF's hyperperiod has to be modified (and corresponds to the period of the added partition). This MAF becomes:

$$MAF_1 = \{s_1, s_2\} \tag{18}$$

When enlarging the MAF, previous slots are replicated. We only have to validate the allocation of a new partition on the first slots to avoid useless verification (same cyclic MAF). Slots can be characterized by:

$$\begin{aligned} s_{1,MAF_1} &= \{r_1 = 0, r_2 = 3, r_3 = 5\} \\ s_{2,MAF_1} &= \{r_1 = 0, r_2 = 3\} \end{aligned} \tag{19}$$

The end-to-end delay of the extended c_1 chain has to be verified:

$$D_{c_1,a_4} = delay(1,2) + delay(2,3) + C_1 + C_2 + C_3 \tag{20}$$

In that case:

$$D_{c_1,a_4} = 0 + 0 + 3 + 2 + 2 = 7 \leq D_{c_1,max} \tag{21}$$

All constraints are verified, so a_4 is a valid partial allocation.

In a fourth step, we try to allocate P_4 , either to the same processing element PE_1 in the partial allocation a_6, or to the new processing element PE_2 in the partial allocation a_7. If we take a_7 case, the MAF on PE_1 is not modified and the MAF on PE_2 is quite simple: $MAF_2 = \{s_1\}$ with $s_1 = \{r_4 = 0\}$. As D_{c_3,a_7} is equal to 9, it is a valid node.

The fifth step deals with the allocation of P_5, either on PE_1, PE_2 or on a new processing element PE_3 which corresponds to the allocations a_8, a_9, a_{10} respectively. In the case of node a_{10}, the communication between P_2 and P_5 leads to:

$$D_{c_2,a_{10}} = delay(2,5) + C_2 + C_5 \tag{22}$$
$$delay(2,5) = WCTT(2,5) + T_5 = 5 + 40 = 45 \tag{23}$$

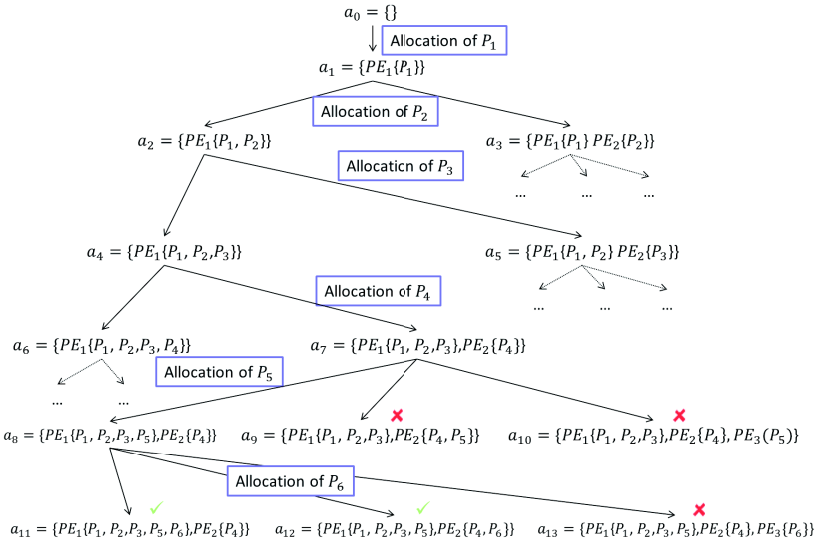

Fig. 7. An in-depth research in an allocation tree

$$D_{c_2,a_{10}} = 45 + 2 + 1 = 48 > D_{c_2,max} \tag{24}$$

We notice that if partitions P_2 and P_5 are not on the same processing element, the end-to-end constraint D_{c_2} is not verified. a_9 and a_{10} cannot have new child node.

In a_8 partial allocation node, we add P_5 in the processing element PE_1. As the partition P_5 has a higher period, the hyperperiod has to be modified and the MAF has to be extended again. 2 new slots have to be added, s_3 and s_4 replicate the slots s_1 and s_2 respectively:

$$MAF_1 = \{s_1, s_2, s_3, s_4\} \tag{25}$$

As there is enough time in the first slot to run P_5, it is allocated to this slot:

$$\begin{aligned} s_1 &= \{r_1 = 0, r_2 = 3, r_3 = 5, r_5 = 7\} \\ s_2 &= \{r_1 = 0, r_2 = 3\} \\ s_3 &= \{r_1 = 0, r_2 = 3, r_3 = 5\} \\ s_4 &= \{r_1 = 0, r_2 = 3\} \end{aligned} \tag{26}$$

Allocating it in the first processing element permits to valid end-to-end delay $D_{c_2,max}$.

In a sixth step, we try to allocate P_6. Three child nodes are then generated: a_{11}, a_{12} and a_{13}. a_{13} is not valid due to communication delay. a_{11} and a_{12} are valid after a scheduling search. In the case of the allocation a_{12}, the partition P_6 is delayed to begin at 15 ms in order to valid D_{c_3} with a loop as illustrated in figure 8. In the case of the allocation a_{11}, the partition P_6 can only be scheduled at the end of the second or the fourth slots as depicted in figure 9, otherwise the scheduling does not respect equation 6. The solution consists in shifting step by step the less frequent partitions in other slots: P_5 is shifted to s_{2,MAF_1}. Then, P_6 is scheduled in the second slot, just after P_5, validating D_{c_3}.

Finally, the validated global allocations are a_{11} and a_{12}.

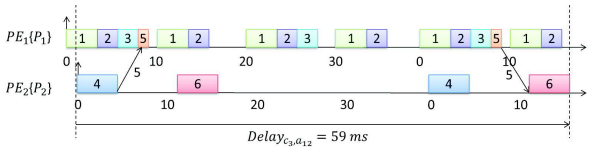

Fig. 8. Valid scheduling schemes in allocation a_{12}

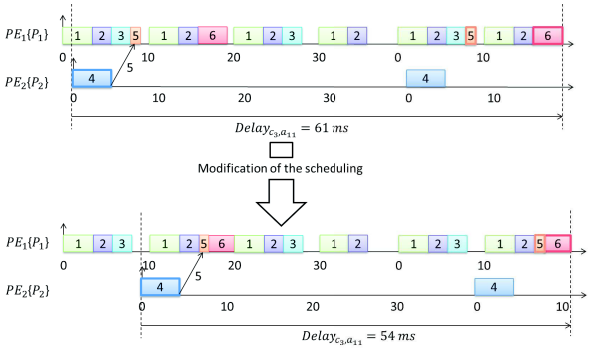

Fig. 9. Modification of the scheduling in allocation a_{11}

IV. Mapping algorithm on a case study

This work is based on an helicopter case study described in [13]. The aim is to distribute its avionics functions on a larger number of less complex processors.

Figure 10 describes the applicative architecture, i.e. seven partitions, P_1 to P_7: four partitions, P_1 to P_4, can be distributed in several processing elements whereas three ones, P_5 to P_7, have to be allocated to all of them.

The existing allocation is depicted in figure 11 (1 processing element, allocation 1). One envisioned evolution of this system is to replace each processing element by a set of less powerful ones. Figure 11 depicts 4 among 14 possible allocation

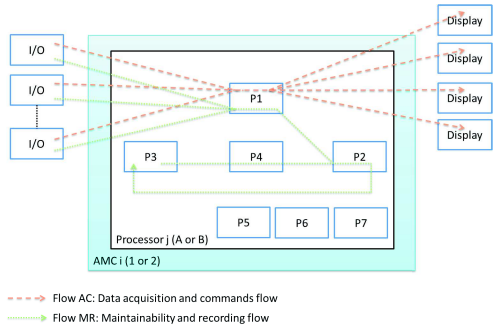

Fig. 10. Main flows of communicating partitions

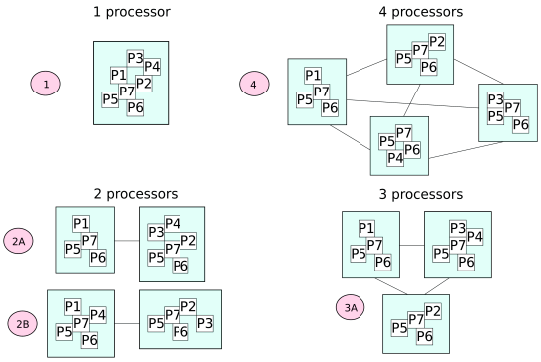

Fig. 11. Possible allocation schemes

schemes on sets of 2, 3 and 4 processing elements.

We focus on the three communicating partitions as depicted in figure 12. At the end of their execution, partitions send their computed data, whose sizes are given above the outcome arrow. One constraint taken into account in this case study is to avoid any deterioration of running with or without a distribution of partitions: the button-to-action delay cannot exceed 78 ms, the current value given by the case study.

We have taken 4 different types of processing elements. The corresponding WCET for each partition is detailed in Table II. In this study, the WCTT is equal to 2 ms.

The following results detailed in Table III give the number

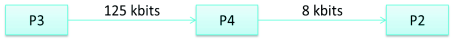

Fig. 12. The communication chain c_1

TABLE II
REAL-TIME SPECIFICATIONS OF PARTITIONS ACCORDING TO THE PROCESSOR TYPE

Partitions	$Proc_1$	$Proc_2$	$Proc_3$	$Proc_4$	Period
P_1	10	12	13	14.5	25
P_2	10	10	11	11.5	50
P_3	6	6.5	7	8	100
P_4	6	6.5	7	8	50
P_5	5	5.5	6	6.5	100
P_6	2	2	2.5	2.5	100
P_7	1	1	1	1.5	100

TABLE III
VALID PROCESSOR ALLOCATIONS ACCORDING TO THE PROCESSOR TYPE

Number of processors	$Proc_1$	$Proc_2$	$Proc_3$	$Proc_4$
2	3	2	2	1
3	2	2	2	0
4	0	0	0	0

of valid allocations according to the processor type and the number of processing elements.

When the processor type $Proc_4$ is used, we note that only one allocation is valid. This single case is composed of two processing elements where partitions P_2, P_3 and P_4 are on the same processing element. The best scheduling is depicted in figure 13 and shows that there is no waiting time between the execution of communicating partitions.

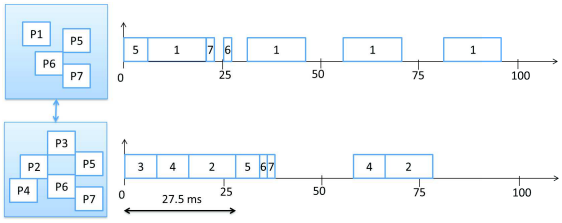

Fig. 13. A valid configuration with 2 processors $Proc_4$

Other allocations with two processing elements $Proc_4$ were not processed. Let us take one of them: partitions P_1, P_3, P_5, P_6 and P_7 are allocated to one processing element and P_2, P_4, P_5, P_6 and P_7 to the other one. As illustrated on figure 14, first partition P_3 runs and transmits its data at the end of its execution. Then, the message arrives at destination processor, but P_4 has already begun its execution: the data will be used at the next cycle of P_4, i.e. 50 ms later. At the end of the second execution of partition P_4, there is no waiting time to execute the partition P_2. The end-to-end delay is equal to 79.75 ms, which is 1.75 ms too much.

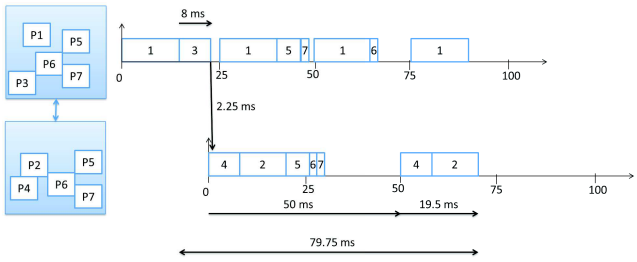

Fig. 14. A scheduling configuration of 2 PEs $Proc_4$ showing an exceeded delay

In the case where 4 processing elements are used, we note that no valid allocation exists due to the delay which exceeds 78 ms: as a message can arrive when a partition has just begun its execution, the waiting delay undergone by the received data is equivalent to the period of the destination partition. Here, all communicating partitions are on different processing

elements. In a worst case study, the messages can arrive just after the beginning of destination partitions: the added waiting time is the sum of the periods of destination partitions, in our example, two times 50 ms. Thus, the waiting time is higher than the button-to-action delay.

When the WCETs are lower, i.e. with the processor types $Proc_1$, $Proc_2$ or $Proc_3$, it exists two allocations with 3 processing elements which meet the different constraints: allocation, scheduling and delay. These allocations are composed of a first processing element which contains partitions P_1, P_5, P_6 and P_7. The second and the third ones both contain P_5, P_6 and P_7. P_2, P_3 and P_4 are allocated to the different processing elements according to the communications as illustrated on figure 15: either P_3 and P_4 (configuration 3A) or P_2 and P_4 (configuration 3B) are on the same processing element. In these allocations, a single remote communication exists in the chain. Delays in other allocations exceed the button-to-action delay because the waiting time becomes too important like the allocation to four processing elements: as communicating partitions are on different processing elements, the worst delay depends on the period of the destination partitions. To avoid too high delays, two communicating partitions whose destination partition period is large must be preferably allocated on the same processing element.

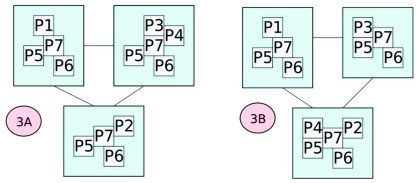

Fig. 15. Possible allocations to 3 specific processors

V. Mapping algorithm scalability analysis

In this section, we are looking for the limits of this algorithm on a larger architecture. Different parameters are modified at partitions level:

- the number of partitions: from 10 up to 20,
- the WCET noted C: from 5 ms up to 25 ms.

The partitions have here the same period $T = 25ms$.

The set $\mathcal{C}$ is composed of N chains c of two communicating partitions: $c_i = \{P_{2i-1}, P_{2i}\}$. The end-to-end delay $D_{c_i,max}$ is the same for all the chains and is equal to either 20 ms (a very restrictive delay) or 40 ms.

These partitions are allocated to a set of 10 up to 20 processing elements PE.

A first test consists in allocating 10 partitions to up to 10 processing elements, taking into account different values of the WCET. Here, $D_{c_i,max}$ is equal to 20 ms. We notice in Tables IV and V that when the WCET increases, analysis times and numbers of generated MAFs respectively decrease. A quick cut of the branches is done due to unschedulable partitions or unsatisfied end-to-end temporal constraints in partial allocations when the WCET is high.

TABLE IV
Analysis time (s) for 10 partitions when $D_{c_i,max} = 20ms$

Number of PEs	WCET (ms)				
	5	10	15	20	25
2	0.04	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
3	0.06	$1 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
4	0.08	$1 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
5 and more	0.09	$2 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$

TABLE V
Number of generated MAFs for 10 partitions when $D_{c_i,max} = 20ms$

Number of PEs	WCET (ms)				
	5	10	15	20	25
2	8752	7	2	2	1
3	11103	11	2	2	1
4	16445	15	2	2	1
5	16946	18	2	2	1
6 and more	16947	19	2	2	1

TABLE VI
Analysis time (s) for 10 partitions when $D_{c_i,max} = 40ms$

Number of PEs	WCET (ms)				
	5	10	15	20	25
2	0.06	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
3	235	$2 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
4	4956	$6 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
5	12866	$8 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
6	15463	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
7	15560	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
8	22520	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
9	21625	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
10	21054	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$

TABLE VII
Number of generated MAFs for 10 partitions when $D_{c_i,max} = 40ms$

Number of PEs	WCET (ms)				
	5	10	15	20	25
2	1415	10	2	2	2
3	450923	24	2	2	2
4	2415999	52	2	2	2
5	2908636	62	2	2	2
6	2967705	93	2	2	2
7	1456829	93	2	2	2
8	1457616	93	2	2	2
9	1457662	93	2	2	2
10	1457663	93	2	2	2

A second test consists in slackening the end-to-end temporal constraint $D_{c_i,max}$. If we compare Tables IV and V corresponding to the analysis with $D_{c_i,max}$=20 ms and Tables VI and VII corresponding to the analysis with $D_{c_i,max}$=40 ms, we note an explosion of the number of generated MAFs and the analysis time when the WCET is low. More branches are covered because more scheduling schemes are possible and could be valid.

A third test consists in modifying the number of partitions. The WCET for all the partitions is equal to 5 ms and the end-to-end delay constraint $D_{c_i,max}$ for all the chain is 20

TABLE IX
NUMBER OF GENERATED MAFS WHEN $C_i = 5ms$ AND $D_{c_i,max} = 20ms$

Number of PEs	Number of partitions					
	10	12	14	16	18	20
2	8752	8752	8752	8752	8752	8752
3	11103	14424	822684	822684	822684	822684
4	16445	108876	913731	963771	77703186	77703186
5	16946	129861	1456356	10214991	82803486	83589411
6	16947	130864	1518394	12491602	140640952	1055180557
7	16947	130865	1520158	12643182	148237480	$> 1.2 * 10^9$
8	16947	130865	1520159	12646020	148561189	$> 1.2 * 10^9$
9	16947	130865	1520159	12646021	148565468	$> 1.2 * 10^9$
10	16947	130865	1520159	12646021	148565469	$> 1.2 * 10^9$

TABLE VIII
ANALYSIS TIME (S) WHEN $C_i = 5ms$ AND $D_{c_i,max} = 20ms$

Number of PEs	Number of partitions					
	10	12	14	16	18	20
2	0.04	0.04	0.04	0.05	0.05	0.06
3	0.06	0.08	5.8	6.3	6.8	7.0
4	0.08	0.7	7.6	9.9	841.3	869.4
5	0.09	0.8	12.6	155.8	2061.6	4631.9
6	0.09	0.8	12.9	177.4	4877.0	$> 10^5$
7	0.09	0.8	12.9	176.2	4189.8	$> 10^5$
8	0.09	0.8	12.9	175.9	4038,4	$> 10^5$
9	0.09	0.8	12.9	172.8	3967,7	$> 10^5$
10	0.09	0.8	12.9	171.0	4068,0	$> 10^5$

ms. Firstly, we can state with Table VIII that the analysis time reaches a maximum threshold from a number of processing elements, e.g. 6, 7 and 8, ... processing elements for respectively 10, 12, 14, ... partitions. According to Table IX, the number of generated MAFs reaches a limit. Secondly, we note that the analysis time is multiplied by 10 when we add 2 new partitions: as the number of possibilities increases, a larger number of branches in the allocation tree is covered.

This algorithm gives encouraging results on a larger architecture, but it reaches its limits due to a temporal explosion of this worst-case analysis when a larger number of partitions with small WCET related to their period are processed, or delays are not constrained. Those limits are due to the large number of possible scheduling schemes which increases the size of the search tree.

VI. CONCLUSIONS AND PERSPECTIVES

In this paper, we propose an integrated mapping approach taking into account altogether scheduling of partitions on each processing element as well as end-to-end timing constraints. We explain on an illustrative example how a depth search in a tree of potential partial allocations can be handled. Then, we show on a helicopter case study how this mapping algorithm can help choosing the optimal number of processing element in a distributed IMA architecture. A scalability analysis shows some limits of such a comprehensive algorithm on large configurations: computing the MAF for each processing element is costly and a heuristic approach should be proposed.

Ongoing work deals with the definition of such a heuristic approach in order to be able to find optimal or near-optimal allocations in the context of complex systems. Moreover, taking into account local I/O constraints when mapping partitions is another important objective as well as analysis of a possible oversampling of emitting partitions (increasing their period to overcome exceeded end-to-end communication delays).

REFERENCES

[1] P. Bieber, F. Boniol, M. Boyer, E. Noulard, and C. Pagetti, "New challenges for future avionic architectures," *Journal Aerospace Lab*, vol. 4, May 2012.

[2] J. Moore, *Digital Avionics Handbook, Second Edition - 2 Volume Set*. CRC Press 2000, 2001, ch. Advanced distributed architectures, pp. 33–1–33–12.

[3] *ED-112A: Minimum operational performance specification for crash protected airborne recorder systems*, EUROCAE, EUROCAE Std., September 2013.

[4] A. Al Sheikh, "Resource allocation in hard real-time avionic systems. scheduling and routing problem," Ph.D. dissertation, EDSYS, September 2011.

[5] C. Ekelin and J. Jonsson, "A lower-bound algorithm for minimizing network communication in real-time systems," in *Parallel Processing, 2002. Proceedings. International Conference on*, 2002, pp. 343–351.

[6] H. Bauer, J. Scharbarg, and C. Fraboul, "Worst-case end-to-end delay analysis of an avionics afdx network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 1220–1224.

[7] X. Li, J. Scharbarg, and C. Fraboul, "Worst-case delay analysis on a real-time heterogeneous network," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, June 2012, pp. 11–20.

[8] M. Lauer, J. Ermont, F. Boniol, and C. Pagetti, "Latency and freshness analysis on ima systems," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, Sept 2011, pp. 1–8.

[9] N. Badache, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, "End-to-end delay analysis in an integrated modular avionics architecture," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Sept 2013, pp. 1–4.

[10] M.-Y. Nam, R. Pellizzoni, L. Sha, and R. Bradford, "Asiist: Application specific i/o integration support tool for real-time bus architecture designs," in *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*, June 2009, pp. 11–22.

[11] *Avionics application software interface, part 1 - Required services, ARINC specification 653P1-3*, Aeronautical radio, Inc., Aeronautical radio, Inc. Std., November 2010.

[12] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," *Proceedings of the IEEE Real-Time System Symposium - Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Barcelona, Spain, November 30, 2008*, 2008.

[13] E. Deroche, J.-L. Scharbarg, and C. Fraboul, "Performance evaluation of a distributed ima architecture," in *Work-in-Progress Proceedings, Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, July 2015, pp. 17–20.