Mapping hard real-time applications on many-core processors

Quentin Perret, Pascal Maurere, Eric Noulard, Claire Pagetti, Pascal Sainrat, Benoît Triquet

Mapping hard real-time applications on many-core processors

Quentin Perret
Airbus Operations S.A.S.
Toulouse, France

Pascal Maurère
Airbus Operations S.A.S.
Toulouse, France

Éric Noulard
DTIM, UFT-MiP, ONERA
Toulouse, France

Claire Pagetti
DTIM, UFT-MiP, ONERA
Toulouse, France

Pascal Sainrat
IRIT, UFT-MiP, UPS
Toulouse, France

Benoît Triquet
Airbus Operations S.A.S.
Toulouse, France

ABSTRACT

Many-core processors are interesting candidates for the design of modern avionics computers. Indeed, the computational power offered by such platforms opens new horizons to design more demanding systems and to integrate more applications on a single target. However, they also bring challenging research topics because of their lack of predictability and their programming complexity. In this paper, we focus on the problem of mapping large applications on a complex platform such as the KALRAY MPPA®-256 while maintaining a strong temporal isolation from co-running applications. We propose a constraint programming formulation of the mapping problem that enables an efficient parallelization and we demonstrate the ability of our approach to deal with large problems using a real world case study.

1. INTRODUCTION

Many-core processors are a potential technology for aerospace industry as long as there is a way to certify them according to aeronautics standards, in particular with:

- DO 178 B/C [25] which imposes that the WCET of any application can be computed;
- DO 297 [24] which states that mixed-critical applications can execute on the same resource as long as they are strongly segregated, in the sense that an application cannot interfere with others even in the presence of failures;
- CAST-32 [5] and FAA white paper [13] which detail how temporal interferences affect the safety and why mitigation means against those interferences are mandatory, when executing on multi-core COTS processors.

Because of the architecture of many-core chips, computing WCET requires to be able to bound safely the interference delays on shared resources [26] meanwhile ensuring segregation imposes that execution times must not vary whatever the co-running applications do.

1.1 Temporal isolation-based framework

The overall framework workflow is shown in Figure 1. When implementing several applications on the same target (single or multi/many-core chips), two stakeholders interact: on the one hand, the *application designers* are in charge of developing their application and on the other hand, the *platform integrator* is in charge of integrating the different applications on the shared platform.

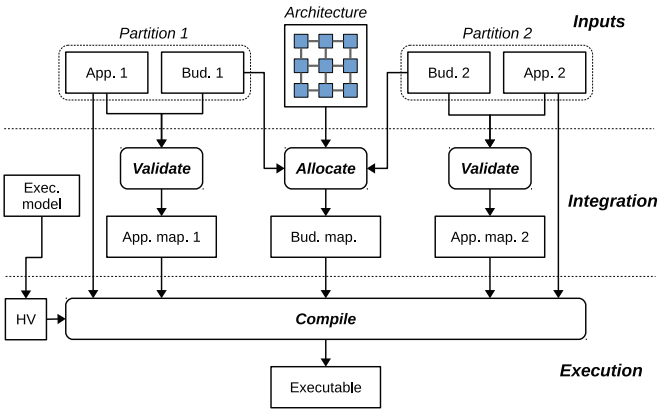


Figure 1: Workflow

In [19], we proposed an approach to implement real-time safety-critical applications on a many-core target in such a way that strict temporal guarantees are ensured whatever the behaviour of other applications sharing the resources is. The approach is based on a four-rules execution model to be followed by the developers (application designers and platform integrator) to avoid any unpredictable or non-segregated behaviours. Some of the rules are enforced by an off-line mapping and computation, while others are ensured by a bare-metal hypervisor that we developed. That initial work concerned mostly the activities of the *platform integrator*. More precisely, given an abstract view of applications in the form of a set of *application budgets*, the integrator maps off-line during the *Allocate* phase the budgets on the target, while respecting the execution model. The output is both the computed mapping and the associated code (memory allocation, boot code of the cores and schedule of budgets). Given the application internal execution in the budget, the applications can then run on the target.

1.2 Contribution

In this paper, we focus on the *application designers* activity which is modified due to the use of a many-core platform. The purpose is to help them provide their budget. More specifically, each application (*App*) is defined as a set of multi-periodic communicating tasks and is wrapped in a single partition. A partition also contains an *application budget* (*Bud*) which is an abstraction of the application that represents the needs in terms of resources (memory, CPU, communication and IOs). The budget is the interface format between the *application designer* and the *platform integrator*. Indeed, the latter does not need a complete knowledge of each application to share the platform.

Before explaining how to obtain a budget, we first recall the platform model, the execution model and the application definition in section 2. We then detail some basic formulas to derive a minimal budget (see section 3). However, it is up to the *application designer* to define its budget. Then, we propose a constraint programming approach to *validate* a given budget (see section 4). A budget is considered as valid when it enables a correct implementation of the application for both functional and non-functional requirements. During the *validate* phase, we compute a complete schedule of the application in its budget. This schedule can be distributed over several cores and includes not only the mapping of tasks on cores but also the tight management of the communication over the Network on Chip (NoC). Once a correct schedule has been found, the user can then decide to keep it or to optimize it by changing some parameters (ex: the number of cores, the length of the time-slices, ...). A second output is the internal budget code. In section 5, we illustrate the applicability and the scalability of the constraint programming approach. We then present the related work and conclude.

2. SYSTEM MODEL

2.1 Hardware platform

2.1.1 Overview

The KALRAY MPPA®-256 (Bostan version) integrates 288 cores distributed over 16 *Compute Clusters* and 4 *I/O clusters* interconnected by a dual *Network on Chip* (or *NoC*), as shown in Figure 2. The 16 *compute clusters* are dedicated to run user applications while the *I/O clusters* serve as interfaces with the out-chip components such as the DDR-SDRAM. All cores are identical 32-bits VLIW processors integrating complex components such as a private MMU (Memory Management Unit) enabling both virtual addressing and memory protection.

2.1.2 Compute and I/O Clusters

Each compute cluster is composed of 16 cores (named *PEs*), 1 specific core used for resource management (*RM*), 2MiB of Static Random Access Memory (or SRAM) exclusively accessible by the elements of the cluster and 1 DMA engine to transfer data between clusters over the NoC. The local SRAM is organised in 16 memory banks that can be accessed in parallel without inter-core interferences.

Each I/O cluster integrates 4 RMs, 4 DMAs (capable of sending data through 4 different NoC access points) and several additional peripherals such as a DDR-SDRAM controller or an Ethernet controller.

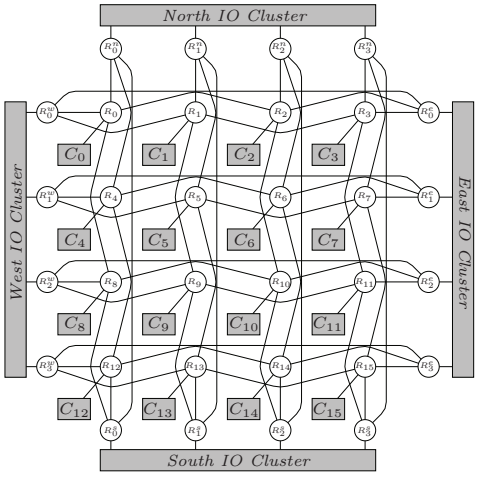


Figure 2: MPPA®-256 architecture overview

2.1.3 Network on Chip

Two 2-D torus NoCs interconnect the compute and I/O clusters. The *D-NoC* can be used for large data transfers while the *C-NoC* enables inter-cluster barriers thanks to short control messages. In both cases, the route of each packet must be defined explicitly by software.

2.2 Execution model

In [19], we have defined a four-rules execution model in order to ensure a strict temporal isolation between applications. The application designers must follow those rules to avoid any unpredictable or non-segregated behaviours. We give a brief reminder of the four rules of the execution model:

Rule 1: Spatial partitioning inside compute clusters Any PE (resp. any local SRAM bank) inside any compute cluster can be reserved by at most 1 partition. This rule ensures that a partition will never suffer from local interferences caused by other partitions.

Rule 2: Isolated accesses on the NoC Any communication on a route on the NoC does not encounter any conflict. Given two communications, either they take two non-overlapping routes or they access at different times their overlapping routes. More precisely, communications are scheduled in a TDM (Time-Division Multiplexing) manner and strictly periodic slots are defined off-line for each communication.

Rule 3: Off-line pre-defined static buffers The memory areas to be sent over the NoC must be defined off-line. This allows to both reduce the combinatorial explosion of WCET estimation through static analysis and ease the measurement-based validations.

Rule 4: Isolated accesses to DDR-SDRAM bank Any bank of the external DDR-SDRAM memory can be shared by two or more partitions as long as they never access it simultaneously. This allows to greatly mitigate the inter-partitions interferences at the external bank level.

Some of the rules are enforced by an off-line mapping and computation; while others are ensured by a bare-metal hypervisor that we developed. Further low-level details on the implementation of this hypervisor are provided in [19].

2.3 Application model

An application is a tuple $\langle \tau, \delta \rangle$ where:

1. $\tau = \{\tau_1, \ldots, \tau_n\}$ is a finite set of periodic tasks. A task τ_i is defined as $\tau_i = \langle S_i, P_i, T_i \rangle$:
   - $S_i = \{\tau_i^1, \ldots, \tau_i^{n_i}\}$ is the set of sub-tasks in τ_i with n_i the number of sub-tasks composing τ_i. All the sub-tasks in S_i are assumed to be activated simultaneously at the activation of τ_i and to have the same implicit deadline equal to T_i. Additionally, the sub-tasks are defined as $\tau_i^j = \langle C_i^j, M_i^j, I_i^j, O_i^j \rangle$ with: C_i^j the WCET of the sub-task; M_i^j the memory footprint of the sub-task (size of code, static and read-only data); I_i^j and O_i^j respectively the input and output buffers[1] read and written by τ_i^j
   - $P_i \subset S_i \times S_i$ is the finite set of precedence relations constraining the sub-tasks of τ_i. More precisely, $(\tau_i^x, \tau_i^y) \in P_i$ means that τ_i^x must be completed before τ_i^y can start. This kind of precedence relations are assumed to exist only between sub-jobs exchanging some data. Moreover, the precedence constraints imposed by P_i have no cycles and can be seen as a Directed Acyclic Graph (or *DAG*).
   - T_i the period of the task.
2. $\delta = \{\delta_1, \ldots, \delta_m\}$ the set of data with $\delta_k = \langle m_k, prod, cons \rangle$ with m_k the size of the data; $prod : \delta \mapsto S$ which provides the sub-job producing the data (with $S = \bigcup_i S_i$); and $cons : \delta \mapsto S^n$ which provides the set of sub-tasks consuming the data.

The model does not allow precedence constraints between sub-tasks having different parent tasks. However, no such constraints are imposed for the data exchanges. The only constraint for data is to be consumed over the freshest-value semantics in a *deterministic* manner[2].

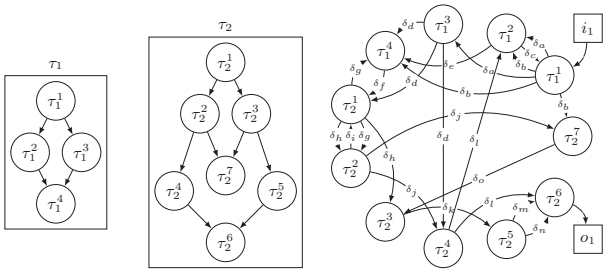
Example 1. *We consider an application composed of 2 tasks τ_1 and τ_2 respectively having 4 and 7 sub-tasks. The 11 sub-tasks exchange 15 data $\{\delta_a, \ldots, \delta_o\}$ and read from 1 input buffer i_1 and write into 1 output buffer o_1. Tasks τ_1 and τ_2 respectively have periods $T_1 = 24$ and $T_2 = 48$. The precedence constraints are depicted in Fig. 3a as two DAG (one for each task) and the production and consumption of data by tasks and the I/O buffers are depicted in the form of a multi-digraph in Fig. 3b. The parameters of the sub-tasks are provided in Table 1 (where* ck *stands for clock cycle).*

[1]The I/O buffers only represent the interactions with out-of-chip components such as the reception or emission of Ethernet frames.

[2]Here *deterministic* refers to the data production and consumption ordering. We consider that two executions of the same schedule must always ensure the same order of production and consumption of all data.

Sub-task	C_i^j (in ck)	M_i^j (in KiB)	I_i^j	O_i^j
τ_1^1	500	519	i_1	∅
τ_1^2	500	397	∅	∅
τ_1^3	700	642	∅	∅
τ_1^4	400	262	∅	∅
τ_2^1	1,000	287	∅	∅
τ_2^2	400	799	∅	∅
τ_2^3	500	542	∅	∅
τ_2^4	800	764	∅	∅
τ_2^5	900	490	∅	∅
τ_2^6	500	399	∅	o_1
τ_2^7	600	12	∅	∅

Table 1: Example of sub-tasks parameters



(a) Precedence constraints (b) Data exchanges

Figure 3: Example of sub-tasks dependencies.

2.4 Definition of application's budget

In [19], we have defined the notion of application budget, that is the interface between application designers and the integrator. The application designer must detail to the integrator the amount of resources needed by its software. We remind those definitions below for the paper to be self-contained.

Definition 1. *An* application budget *is defined as a tuple $\langle \mathcal{P}, \mathcal{B}, \mathcal{I}, \mathcal{C} \rangle$ where*

1. *$\mathcal{P} = \{PN_1, \ldots, PN_m\}$ is a finite set of PNs. A* Partition Node *(or PN) is a pair $\langle N_c, N_b \rangle$ which represents the need of processing resource (N_c is a number of cores) and memory (N_b is a number of local memory banks) allocated to an application inside one cluster;*
2. *$\mathcal{B}$ represents a number of external DDRx-SDRAM banks;*
3. *$\mathcal{I} = \{I/ON_1, \ldots, I/ON_j\}$ is a finite set of I/ONs. An* I/O Node *(or I/ON) represents an access point to the external memory. It is materialized as a processor located on an I/O cluster.*
4. *$\mathcal{C} = \{PC_1, \ldots, PC_p\}$ is a finite set of PCs. A* Partition Communication *(or PC) represents a need of communication between two PNs located on two different clusters (or a PN and an I/ON) in the form of a directed strictly periodic NoC access slot. Moreover, a PC is defined as a tuple $\langle src, dst, \langle T, C, O \rangle \rangle$ where:*
   - src $\in \mathcal{P} \cup \mathcal{I}$ *(resp.* dst $\in \mathcal{P} \cup \mathcal{I}$*) is the source (destination) of the communication. We assume that there exists at most one PC linking* src *and* dst *(no two different ways to communicate).*

- $\langle T, C, O \rangle$ *is the strictly periodic slot with period* T*, duration* C *and offset* O*.*

3. BUDGETING

The budget of an application is a set of PNs storing and executing all the tasks, a number of DDR banks, a set of I/ONs, a set of PCs interconnecting the PNs and I/ONs.

3.1 Assumptions

3.1.1 No code fetch

On the KALRAY MPPA®-256, the PEs cannot directly access the external DDR-SDRAM, they must always use the DMA engines instead. In particular, the process of loading code from the external DDR-SDRAM (in case an application does not fit into the local SRAM) must be achieved explicitly by software. However, we will assume in the rest of the paper that no code fetch from the external DDR-SDRAM ever occurs. Several works already considered code fetching such as [3, 12], we prefer to focus on the multi-cluster parallelization and the efficient use of the NoC instead.

3.1.2 Execution model and hypervisor

One local SRAM bank is reserved by the hypervisor in each cluster (see [19] for further details). Any of the 15 remaining local memory banks can be used by applications.

A global tick (denoted as the *Systick*) of period T_{sys} activates periodically the hypervisors of all clusters simultaneously. Thus, the duration and the period of any PC must be multiple of the *Systick*.

3.1.3 Valid budget

A budget is valid if the application can execute safely with the resources offered by the budget. More precisely,

DEFINITION 2 (VALID BUDGET). *For an application* $\mathcal{A} = \langle \tau, \delta \rangle$*, a budget* $\langle \mathcal{P}, \mathcal{B}, \mathcal{I}, \mathcal{C} \rangle$ *is valid if there exist:*

- *a mapping of all sub-tasks* $\tau_i^j \in S$ *to* $\mathcal{P}$*. More precisely, each sub-task is associated to a PN=* $\langle N_c, N_b \rangle$*, executes on one of the* N_c *core and its memory footprint is stored in the* N_b *banks;*
- *a local schedule on each core so that the sub-tasks respect their deadlines and their precedence constraints;*
- *a mapping of each sub-task's I/O to the* $\mathcal{B}$ *DDR banks;*
- *a mapping for each data on a PC from the producer's PN to the consumers PNs.*

EXAMPLE 2 (VALID BUDGET AND ASSOCIATED SCHEDULE). *We consider the task set defined in Example 1. As shown on Fig. 4, a valid budget for this task set is composed of three PNs* p_0*,* p_1 *and* p_2*; five PCs* c_{10}*,* c_{21}*,* c_{01}*,* c_{12} *and* c_{20} *linking the PNs where* c_{ij} *is the PC going from* p_i *to* p_j*, and one I/ON accessible from two PCs* c_{io2} *and* c_{1io}*. All PNs have a similar configuration with* $N_c = 1$ *and* $N_b = 15$*. All the PCs linking the PNs also have an identical configuration with the same duration* L_c *and the same period* T_c*. The PCs giving access to the I/ON have a two time longer period.*

3.2 Minimal budget

In our approach, the definition of the application's budget is left to the application designer and can be chosen for any arbitrary reason. To help the user in its budget estimation, we list below several necessary conditions that any budget should meet to have a chance of being valid.

3.2.1 Memory limitations

Since we do not accept any code fetching, the application code and data must be statically allocated in the PNs local memory. Thus, we can deduce a lower bound on the number of PNs (that is $|\mathcal{P}|$) required to completely store an application. Assuming that the maximum available memory space in each compute cluster is M_a, the number of PNs is bounded by:

$$|\mathcal{P}| \geq \left\lceil \frac{\sum_{\forall \tau_i^j \in S} M_i^j}{M_a} \right\rceil$$

3.2.2 Processing power limitations

An other bound on $\mathcal{P}$ can be deduced from the utilization ratio of the application defined as:

$$U = \sum_{\forall \tau_i^j \in S} \frac{C_i^j}{T_i}$$

Since the number of cores on which the application is scheduled must be greater than U, we can deduce:

$$\sum_{pn \in \mathcal{P}} N_c(pn) \geq \lceil U \rceil$$

On the KALRAY MPPA®-256, $1 \leq N_c \leq 16$ for all PNs because there are 16 PEs in each cluster. Thus, the absolute minimal number of PNs required is $\lceil U/16 \rceil$.

EXAMPLE 3. *The memory footprints of the tasks defined in Example 1 are provided in Table 1. We assume the memory space reserved for storing the data to be exchanged to weight* $M_d = 15$ *KiB. So, the maximum amount of memory available for storing the tasks in each cluster is* $M_a = 15 * 128 - M_d = 1905KiB$ *(16 banks, 1 reserved for the hypervisor, 128KiB in each bank). So, we can deduce from the memory limtation that* $|\mathcal{P}| \geq 3$*. The utilization ratio of the task set of Example 1 is 1.85. From the processing power limitation we can deduce:*

$$\sum_{pn \in \mathcal{P}} N_c(pn) \geq 2$$

.

4. BUDGET VALIDATION

Given a budget, we propose to evaluate its validity by finding a valid schedule of the considered application in the budget with a constraint programming-approach. Such a solution is quite standard to compute off-line mapping and schedule. It also allows to take into account other constraints coming from the specificities of a many-core platform (i.e. the local SRAM limitations or the non-negligible NoC-related delays) and the limitations imposed by the execution model (i.e. only strictly periodic communications

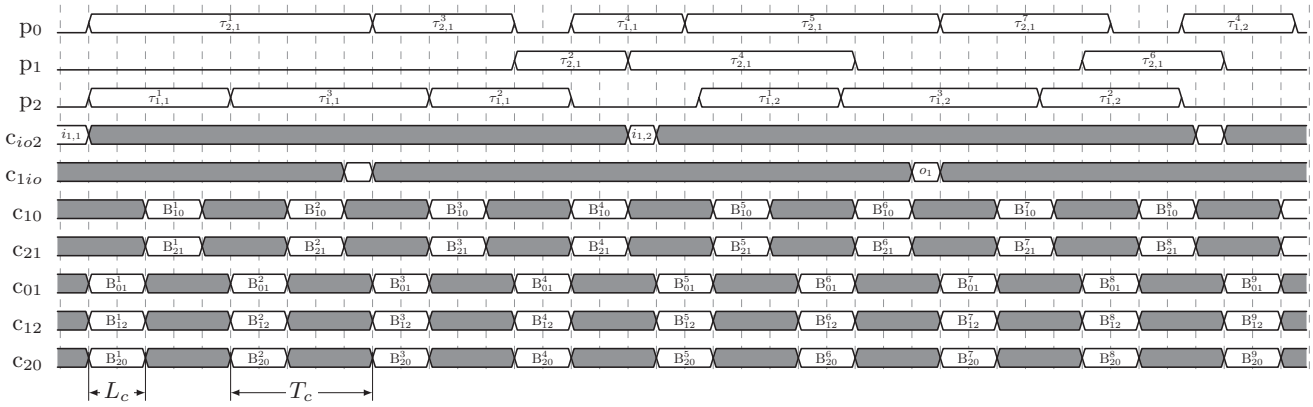


Figure 4: Schedule of the application given in example 1 on the budget of example 2. The communication slots used: $B^2_{20} = \{\delta_{b,1}\}$; $B^4_{20} = \{\delta_{d,1}, \delta_{e,1}\}$; $B^7_{20} = \{\delta_{b,2}\}$; $B^8_{20} = \{\delta_{d,2}, \delta_{e,2}\}$; $B^4_{21} = \{\delta_{d,1}\}$; $B^8_{21} = \{\delta_{d,2}\}$; $B^3_{01} = \{\delta_{g,1}, \delta_{h,1}\}$; $B^7_{01} = \{\delta_{m,1}, \delta_{n,1}\}$; $B^5_{10} = \{\delta_{i,1}\}$; $B^6_{10} = \{\delta_{j,1}\}$; $B^7_{12} = \{\delta_{l,1}\}$.

between clusters). To the best of our knowledge, there is no scheduling technique able of managing all those constraints simultaneously in the literature.

4.1 Modelling framework

Many approaches in the literature have been proposed to map dependent task sets on multi/many-core architectures using an ILP formulation of the problem [3,11,20]. However, those approaches often face major scalability issues when applied to large applications making them unusable in an industrial context without considering sub-optimal heuristics. In this paper, we use a different modelling framework that has proven to be practically useful in order to overcome the same scalability issues that we faced when we developed the preliminary ILP-based formulation of our industrial case study. In the following sections, we will formulate the scheduling problem using the notion of *Conditional Time-Intervals* that has been introduced into IBM ILOG CP Optimizer since version 2.0. In order to ease the reading, we provide a short introduction to some of the concepts associated with this scheduling framework. Further motivation behind this approach and the formal semantics of the operations on Interval variables can be found in [15] and [16].

An *interval* variable i represents an activity of finite duration $l(i)$ whose start date $s(i)$ must be computed by the solver. Each interval i is defined in a pre-definite time window $[a(i), e(i)]$ where $a(i)$ and $e(i)$ respectively are the activation and the deadline of i. This means that a correct schedule must always ensure that $s(i) \geq a(i)$ and $s(i)+l(i) \leq e(i)$. i is said to be *optional* if its presence in the final solution is not mandatory. Among all the interval-related constraints implemented within IBM ILOG CP Optimizer, we use:

- *start* constraints: the start date $s(i)$ of the interval i can be constrained;
- *presence* constraints: for an optional interval i, $p(i) \in [0,1]$ denotes whether 1) i is *present* (i.e. $p(i) = 1$), in which case i is part of the final solution and all constraints on it must be met, or 2) *absent* (i.e. $p(i) = 0$) meaning that i is ignored and not considered in any constraint in which it was originally involved;
- *precedence* constraints: an interval i_1 can occur before a second interval i_2:

$$i_1 \rightarrow i_2 \Leftrightarrow s(i_1) + l(i_1) \leq s(i_2)$$

- *alternative* constraints: for a set of optional intervals $I = \{i_1, \ldots, i_n\}$, it is possible to express that only one interval is present and all the others are absent:

$$\oplus(I) \Leftrightarrow \sum_{i \in I} p(i) = 1$$

- *cumulative function* constraints: A cumulative function represents the usage of a resource by different activities over time as the sum of the individual contribution of these activities. Cumulative functions can be used to constrain the usage of a resource to fit into a specific envelope representing the resource capabilities. Especially, the pulse function $\sqcap(i,h)$ increments the usage of resource by h at the beginning of an interval i and decrements it at the end of i if i is present.

Example 4 (Avoid overlapping with cumulative functions). *Assuming a set of intervals to be scheduled $I = \{i_1, \ldots, i_n\}$ that all use a single resource that is accessible by only one activity at a time, we can enforce a non-overlapping constraints between these intervals by using cumulative functions with* $\sum_{i \in I} \sqcap(i, 1) \leq 1$.

Overall, this formulation with conditional intervals allows a natural modelling of problems in which the activities to be scheduled can be processed by different resources. In such problems, each activity can be associated with several optional intervals (one for each resource) and constrained so that only one of the intervals is present for each activity in the final solution.

4.2 Problem Formulation

4.2.1 Problem inputs

We consider an application $\mathcal{A} = \langle \tau, \delta \rangle$, a budget $\langle \mathcal{P}, \mathcal{B}, \mathcal{I}, \mathcal{C} \rangle$ and a specific platform. Since *Conditional Time-Intervals* apply on jobs and since tasks can have different periods, we

flatten the tasks as jobs on the hyperperiod and the schedule will follow a repeating pattern:

$$T_H = \underset{\tau_i \in \tau}{\text{lcm}} (T_i)$$

We define the *job* $\tau_{i,k}$ of task τ_i as the k-th activation of τ_i in one hyperperiod and $\tau_{i,k}^j$ as the k-th activation of the sub-task τ_i^j. By doing so, we transform the initial multi-periodic problem into a job scheduling problem. S will represent the set of sub-jobs.

The precedence constraints are assumed to exist only between sub-tasks of the same parent task. Therefore, we duplicate each constraint on sub-tasks to many constraints on sub-jobs.

Since each data can have at most one producing sub-task, it is assumed to be over-written by each sub-job of this sub-task. When the data must be read by a sub-task allocated to a different cluster, it is sent from the producing cluster to the consuming cluster during a communication slice following its production. The k-th data δ_x produced by $\tau_{i,k}^j$ is denoted as $\delta_{x,k}$.

4.2.2 Decision variables

There are two decision variables:

j: Each sub-job $\tau_{i,k}^j \in S$ is associated with $|\mathcal{P}|$ optional interval variables. $\forall pn \in \mathcal{P}, \tau_{i,k}^j \in S$, $j(\tau_{i,k}^j, pn)$ is present if $\tau_{i,k}^j$ is allocated on the PN pn. All the sub-jobs intervals have a fixed length equal to the WCET of the sub-job and are defined in a fixed time window $[(k-1) \times T_i; k \times T_i]$.

d: Each data $\delta_{x,k}$ is associated with $|\mathcal{C}|$ optional interval variables. $\forall pc \in \mathcal{C}, \delta_{x,k} \in \delta$, $d(\delta_{x,k}, pc)$ is present if the data $\delta_{x,k}$ is sent over the PC pc. The duration of each data interval is fixed and equal to the duration of its corresponding PC. The time window in which a data interval is defined is equal to the one of its producing sub-job.

REMARK 1 (DURATION OF DATA INTERVALS). *Note that the duration of data intervals does not represent the actual time required to send the data over the NoC. The link between the communication time and the number of data allocated to the same PC is made in section 4.2.7. Note moreover that during a hyperperiod, there may be several slots for a specific PC (exactly $\lceil T_H/T \rceil$). Thus the start time of an interval d indicates in which slot the data is sent.*

4.2.3 PN utilization constraints

Each sub-job is executed only once in only one PN.

$$\forall \tau_{i,k}^j \in S, \oplus(\{j(\tau_{i,k}^j, pn) \mid \forall pn \in \mathcal{P}\}) \quad (1)$$

If different sub-jobs of the same sub-task could execute on different clusters, it would imply 1) to duplicate the sub-task's code and data on all of these clusters; 2) to enforce coherency between the static data of all duplicates thanks to NoC communications. So, in order to reduce the pressure on the NoC, the local SRAM and for simplicity reasons, we impose that all sub-jobs of a sub-task execute on the same PN:

$$\forall \tau_{i,k}^j \in S, \forall pn \in \mathcal{P},\ p(j(\tau_{i,k}^j, pn)) = p(j(\tau_{i,k+1}^j, pn)) \quad (2)$$

The sub-jobs assigned to a common PN may not overlap. This can be modelled with the utilization using cumulative functions to be less than the number of core N_c:

$$\forall pn \in \mathcal{P}, \sum_{\forall \tau_{i,k}^j \in S} \sqcap (j(\tau_{i,k}^j, pn), 1) \leq N_c(pn) \quad (3)$$

The constraint 3 is a necessary and sufficient condition for being able to execute all sub-tasks allocated to a PN non-preemptively on $N_c(pn)$ cores since the *Interval Graph* associated to each PN is *chordal* and can thus be colored using $N_c(pn)$ colors [9].

4.2.4 Local memory constraints

Each sub-task τ_i^j has a memory footprint M_i^j that represents the amount of local SRAM used by τ_i^j in the cluster to which it is attached. Then, we ensure that the local memory available in each PN M_a is sufficient to store the sub-tasks mapped on this PN:

$$\forall pn \in \mathcal{P}, \sum_{\tau_i^j \in S} p(j(\tau_{i,0}^j, pn)) \times M_i^j \leq M_a \quad (4)$$

4.2.5 Precedence constraints

The precedence constraints imposed by the application model can be separated in two categories. A precedence constraint (τ_i^j, τ_i^l) is said to have:

- *forward* data $\Leftrightarrow \exists \delta_x \in \delta, (\tau_i^j = prod(\delta_x)) \wedge (\tau_i^l \in cons(\delta_x))$,
- *backward* data $\Leftrightarrow \exists \delta_x \in \delta, (\tau_i^j \in cons(\delta_x)) \wedge (\tau_i^l = prod(\delta_x))$.

With forward data, the freshest value semantics imposes that τ_i^l cannot start before it receives the data produced by τ_i^j. If both sub-tasks are executed on the same cluster, the data produced by τ_i^j will be visible to τ_i^l immediately after its production since the address space is shared. In this case, the precedence constraint can be met simply by starting τ_i^l after the completion of τ_i^j:

$$\forall (\tau_{i,k}^j, \tau_{i,k}^l) \in P_{i,k}, \forall pn \in \mathcal{P},\ j(\tau_{i,k}^j, pn) \rightarrow j(\tau_{i,k}^l, pn) \quad (5)$$

However, if the two sub-tasks are not executed on the same cluster, the data will need to be sent through the NoC after its production to become visible by τ_i^l. Thus, first, the data must be assigned to a PC slot after the end of the producing sub-job:

$$\forall pc \in \mathcal{C}, \forall \delta_{x,k} \in \delta,\ j(prod(\delta_{x,k}), src(pc)) \rightarrow d(\delta_{x,k}, pc) \quad (6)$$

And then the consumer τ_i^l cannot start before the completion of the communication slot during which the data is sent:

$$\begin{aligned} &\forall (\tau_{i,k}^j, \tau_{i,k}^l) \in P_{i,k}, \forall pc \in \mathcal{C}, \forall \delta_{x,k} \in \delta, \\ &\quad (\tau_{i,k}^j = prod(\delta_{x,k})) \wedge (\tau_{i,k}^l \in cons(\delta_{x,k})) \\ &\quad \Longrightarrow d(\delta_{x,k}, pc) \rightarrow j(\tau_{i,k}^l, dst(pc)) \end{aligned} \quad (7)$$

With backward data, the precedence constraint imposes that the data consumed by $\tau_{i,k}^j$ is always the one that was produced by $\tau_{i,k-1}^l$. To enforce the respect of this constraint, if the two sub-tasks are on the same cluster we reuse the constraint 6, otherwise we impose that $\tau_{i,k}^j$ completes before the beginning of the communication slice during which the data

produced by $\tau^l_{i,k}$ is sent, so that $\tau^j_{i,k}$ could never consume a too fresh data:

$$\begin{aligned}\forall(\tau^j_{i,k},\tau^l_{i,k}) \in P_{i,k}, \forall pc \in \mathcal{C}, \forall \delta_{x,k} \in \delta,\\ (\tau^j_{i,k} \in cons(\delta_{x,k})) \wedge (\tau^l_{i,k} = prod(\delta_{x,k}))\\ \Longrightarrow j(\tau^j_{i,k}, dst(pc)) \rightarrow d(\delta_{x,k}, pc)\end{aligned} \quad (8)$$

4.2.6 Data mapping constraints

Any data produced by a sub-job and consumed by any other sub-job hosted on a different cluster is sent during a communication slice after the completion of the producing sub-job. To do so, we constrain the data produced by the first sub-job of a sub-task to be present on a PC if at least one consuming sub-task is present on the destination PN:

$$\begin{aligned}&\forall pc \in \mathcal{C}, \forall \delta_x \in \delta,\\ &\left(p(j(prod(\delta_{x,0}), src(pc))) \wedge \right.\\ &\left. \sum_{\tau^j_{i,0} \in cons(\delta_{x,0})} p(j(\tau^j_{i,0}, dst(pc))) \geq 1 \right) = p(\delta_{x,0}, pc)\end{aligned} \quad (9)$$

Additionally, we impose to all sub-data to be placed on the same PC:

$$\forall pc \in \mathcal{C}, \forall \delta_{x,k} \in \delta,\ p(d(\delta_{x,k}, pc)) = p(d(\delta_{x,k+1}, pc)) \quad (10)$$

And finally, we enforce all the data sent to be aligned in time with the activation of their PC assuming that $T(pc)$ and $O(pc)$ respectively are the period and the offset of the PC.

$$\forall pc \in \mathcal{C}, \forall \delta_{x,k} \in \delta,\ s(d(\delta_{x,k}, pc)) \bmod T(pc) = O(pc) \quad (11)$$

4.2.7 PC utilization constraints

The amount of data sent during each communication slice must be restricted in order to be compatible with the hardware capabilities and to be implementable on the real target.

Since our execution model and hypervisor provide a property of interference avoidance on the NoC [19], we can derive the maximum number of flit that can be sent during one communication slice simply by assuming that the latency of a NoC link is Δ_L, that the latency of one router is Δ_R and that the maximum number of routers on a NoC route is n^{max}_R. Indeed, based on the models we introduced in [18], we can deduce that the maximum time required for N_{flit} to completely cross a NoC route is $(n^{max}_R + 1)\Delta_L + n^{max}_R \Delta_R + N_{flit}$. So, we can formulate the maximum number of flits that can be transferred during one communication slice of length L_c as:

$$N^{L_c}_{flit} = L_c - (n^{max}_R + 1) \times \Delta_L - n^{max}_R \times \Delta_R$$

Secondly, in order to compute the number of flits required to send a data δ_x, we define $n^{\delta_x}_{bytes}$ and n^{flit}_{bytes} respectively as the number of bytes in δ_x and the number of bytes in one NoC flit. So, the number of payload flits required to transfer the data is $n^{\delta_x}_{flit} = \left\lceil n^{\delta_x}_{bytes} / n^{flit}_{bytes} \right\rceil$. Assuming n^{pkt}_{flit} to be the maximum number of payload flits in one NoC packet, we can derive the number of packets required to send δ_x as $n^{\delta_x}_{pkt} = \left\lceil n^{\delta_x}_{flit} / n^{pkt}_{flit} \right\rceil$. Then, with n^{head}_{flit} the number of flits in one NoC packet header and n^{bbl}_{cycles} the number of empty flits lost in the bubble separating two consecutive packets, we can derive the total number of flits $N^{\delta_x}_{flit}$ required to send δ_x as:

$$N^{\delta_x}_{flit} = n^{\delta_x}_{flit} + n^{\delta_x}_{pkt} \times n^{head}_{flit} + (n^{\delta_x}_{pkt} - 1) \times n^{bbl}_{flit}$$

Finally, in order to simplify the upper-bounding of the amount of data that can be sent during one communication slice, we take the conservative assumption that all data are placed into non-contiguous memory areas. So, assuming that N^{gap}_{flit} is the number of empty flits lost in the gap incurred when the DMA jumps to a non contiguous memory address, we can constrain the amount of data sent into a single communication slice with constraint 12.

$$\begin{aligned}&\forall pc \in \mathcal{C},\\ &\textstyle\sum_{\delta_x \in \delta} \sqcap(d(\delta_x, pc), N^{\delta_x}_{flit} + N^{gap}_{flit}) \leq N^{L_c}_{flit}\end{aligned} \quad (12)$$

Remark 2 (NoC-aware optimization of the memory mapping). *It would be preferable, in order to achieve the best performance, to include the positioning of data with respect to each other into the scheduling problem. Indeed, an optimized memory mapping could enable to group several data into bigger memory chunks that could be sent over the NoC more efficiently. However, such an approach would greatly increase the complexity of the scheduling problem. We will investigate the possibilities of optimal, or at least heuristic, NoC-aware optimization of the memory mapping in future work.*

In order to fulfill implementability constraints, we must also take into account the fact that our DMA micro-code is able of autonomously sending only a finite number of non-contiguous memory areas denoted N^{DMA}_{bufs}.

$$\forall pc \in \mathcal{C},\ \sum_{\delta_x \in \delta} \sqcap(d(\delta_x, pc), 1) \leq N^{DMA}_{bufs} \quad (13)$$

4.2.8 Determinism constraints

Two sub-jobs are allowed to exchange data without being constrained by a precedence relation. In this case, if a data is produced or received during the execution of a consuming sub-job, the order of consumption of this data could be non-deterministic and even change over time depending on the real execution times of sub-jobs. In this paper, we consider this non-determinism in the data production-consumption ordering as not acceptable since the semantics of such an execution is not clear and would break the repeatability of executions (even with the same set of input values) that is required for test purposes. We avoid this non-determinism by forbidding the overlapping between the data interval and its consuming sub-jobs.

$$\begin{aligned}&\forall \delta_{x,k} \in \delta, \forall \tau^j_{i,l} \in cons(\delta_{x,k}), \forall pc \in \mathcal{C}\\ &\quad (prod(\delta_{x,k}), \tau^j_{i,l}) \notin P_k \wedge (\tau^j_{i,l}, prod(\delta_{x,k})) \notin P_k\\ &\quad \Rightarrow \sqcap(d(\delta_{x,k}, pc), 1) + \sqcap(j(\tau^j_{i,l}, dst(pc)), 1) \leq 1\end{aligned} \quad (14)$$

Similarly, if the producing and the consuming sub-jobs are present on the same PN, they must not overlap:

$$\begin{aligned}&\forall \delta_{x,k} \in \delta, \forall \tau^j_{i,l} \in cons(\delta_{x,k}), \forall pn \in \mathcal{P}\\ &\quad (prod(\delta_{x,k}), \tau^j_{i,l}) \notin P_k \wedge (\tau^j_{i,l}, prod(\delta_{x,k})) \notin P_k\\ &\quad \Rightarrow \sqcap(j(prod(\delta_{x,k}), pn), 1) + \sqcap(j(\tau^j_{i,l}, pn), 1) \leq 1\end{aligned} \quad (15)$$

4.2.9 Managing IOs

In our approach, the communication with out-of-chip components are handled as read and write requests into the external DDR-SDRAM. Indeed, this can simulate the reception and emission of AFDX frames for example.

Since, all transactions with the external memory are handled by the compute clusters' DMAs on the KALRAY MPPA®-256, they can be taken into account exactly like the application's data. Thus, we assume all the IO buffers to be represented as conditional intervals together with the data. The only difference resides on the PCs on which IO intervals can be placed which are the PCs linking the PNs and the I/ONs. By doing so, all the constraints for data intervals on PCs can be applied directly for managing the IO intervals. In future work, we will integrate the fine grained modelling of the DDR access time that we developed in [18] to the formulation of constraint 12 when applied on PCs targeting I/ONs.

5. EXPERIMENTAL RESULTS

As a case study, we used a large industrial application from Airbus. This application has several tasks with harmonic periods. The total number of sub-jobs and associated data is close to 100,000 leading to a large optimization problem with several million decision variables and constraints.

5.1 Implementation choices

5.1.1 One PE per PN

In our experiments, we limited the number of cores in each PN to 1 for the following reasons: 1) we limit the potential interferences suffered by the computing core to the local memory accesses generated by the DMA in order to simplify and improve the computation of tight WCETs of sub-tasks; 2) we focus on the mostly unexplored [3] problem of inter-cluster parallelization requiring a tight management of the NoC to exhibit the application's speed-up opportunities through parallelization despite the non-negligible NoC-related delays.

5.1.2 Prompt and symmetric PCs

In the budgets considered during the experiments, we always assume *symmetric* and *prompt* PCs connecting all the PNs of the budget. By *symmetric*, we mean that each PN has one outgoing PC to send data to any other PN and all PCs have the same duration L_c and the same period T_c. By *prompt*, we mean that the value of L_c typically is in the order of magnitude of the Systick and that T_c equal to its minimal value of $(n-1) \times L_c$ with n the number of PNs.

The *symmetry* between the PCs is chosen to simplify the expression of the budget and we argue that the *promptness* of the PCs is well suited for the targeted case study. Indeed, our industrial application basically has many data of small size. By choosing fast PCs we hope to shorten the delays involved by precedences with forward data. The evaluation of fundamentally different budgets with asymmetric and/or slow PCs will be done in future work.

5.2 Goal

Overall, the goal of these experiments is twofold. Firstly, we aim at finding a valid budget with a minimal footprint in order to both maximize the space left to other potential co-running applications and increase the utilization ratio of the reserved resources. And secondly, we want to explore the possibilities of speeding-up the application when parallelized over several clusters despite the non negligible latencies involved by the NoC.

5.2.1 Narrowing the budget

To minimize the footprint of the budget, we compute a minimal number of PNs denoted as n_{pn}^{min} using the rules of Section 3.2. Based on this, we define and evaluate 3 potential configurations with $|\mathcal{P}|$ equal to $n_{pn}^{min} + i$ for $i \in [0;2]$.

Secondly, we investigate the impact of the NoC configuration with 3 PC setups where the length of each PC is equal to $L_c^{min} + j \times$ Systick with $j \in [0;2]$ being the PC configuration number.

5.2.2 Increasing speed-up

To evaluate the speed-up possibilities, we artificially reduce the temporal horizon (i.e. the hyperperiod) on which tasks are scheduled. In this case, the maximum speedup and minimal makespan is obtained when no valid budget can be found for any further reduced temporal horizon. We define a division parameter k_{div} to compute schedules with reduced periods defined as $\widetilde{T}_i = \frac{256-k_{div}}{256} \times T_i$ and we evaluate 256 instances of the application for $0 \leq k_{div} \leq 255$. All other parameters, including the WCETs of sub-tasks are left unchanged.

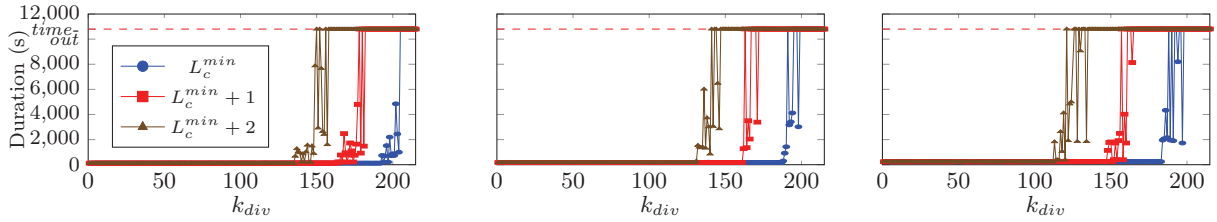
5.3 Results

Figure 5 shows the time (in seconds) required by the solver to compute a schedule meeting all the constraints as a function of the k_{div} parameter in 9 different budget configurations. We can see in figure 5a that the maximum k_{div} (representing the maximum processing load) is counter-intuitively achieved with the budget configuration offering the smallest number of PNs. However, we can also observe on all three figures that the maximum k_{div} are invariably obtained with the shortest PCs. This seems to demonstrate that the bottlenecking parameter to achieve the best performance with this case study is the latency of data exchanged through the NoC and not the processing power. The figure 6 depicts an example of PC utilization from the experiment with the maximum k_{div} over a complete hyperperiod. The figure 6a shows the number on non contiguous memory areas that are sent during each PC activation and the figure 6b shows the number of flits that are sent in each PC slot. Both figures are annotated with the maximum values imposed by the constraints 13 and 12. We observe that the limitation from constraint 13 (fig. 6a) seems to be a lot more bottlenecking than the limitation from constraint 12. This means that the limiting parameter in this case study is not the actual NoC bandwidth but rather the capabilities of DMAs to send many small non-contiguous data. As mentioned in Remark 2, a better optimized memory mapping could probably help send bigger chunks of data and thus balance the curves of figure 6 to achieve the best performances.

6. RELATED WORK

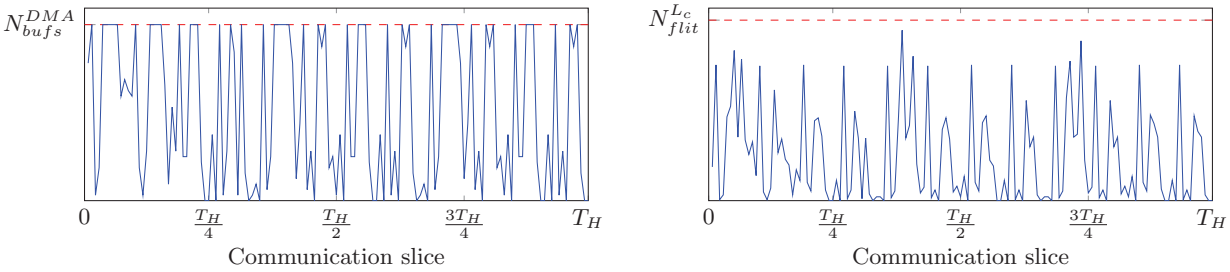
In this paper, we presented a mapping procedure enabling the efficient parallelization of one application into its budget. However, since our execution model allows the concurrent execution of several partitions, our mapping technique can be re-used on several applications needing to be mapped into their corresponding budgets. To the best of our knowledge,

[3]The problem of intra-cluster parallelization has already been addressed in the literature in [3] and the CAPACITES project [2] and will be further explored in the ASSUME project [1].



(a) $|\mathcal{P}| = n_{pn}^{min}$ (b) $|\mathcal{P}| = n_{pn}^{min} + 1$ (c) $|\mathcal{P}| = n_{pn}^{min} + 2$

Figure 5: Comparison of computation times in function of the load increase with different budgets.



(a) PC utilization regarding the number of DMA buffers (b) PC utilization regarding the amount of data sent

Figure 6: Example of PC utilization under maximum processing load

no other approaches enabling the execution of multiple parallelized applications on the Kalray mppa®-256 have been proposed in the literature yet.

Generic DAG scheduling. The theoretical problem of scheduling real-time DAG parallel tasks has already been addressed in the literature for both mono-core [6] and more recently multi-core [17, 22, 23] processors with (G)EDF. However, the classical DAG model used in these papers does not include information on the tasks' communication latencies or on memory limitations and are thus not directly applicable on the Kalray mppa®-256.

Similarly, the static scheduling of DAG tasks on multiprocessor systems with a makepan-optimization objective has been widely studied, mostly using heuristics based on *list* algorithms [14]. Nevertheless, to the best of our knowledge, there are no published algorithm able of managing simultaneously the precedence constraints together with communication latencies, memory limitations and other constraints coming from an execution model.

Mapping on the Kalray MPPA. The problem of mapping real-time applications on the Kalray mppa®-256 has already been addressed several times in the literature [3, 8, 10, 12]. In [3], the authors propose a framework for mapping automotive applications inside a single compute cluster accessing the external DDR-SDRAM but never address the problem of multi-clustered applications. On the other hand, the NoC is taken into account in [8] and [10]. However, in both cases, the guarantees on the NoC are provided with a competition-aware approach based on Network Calculus. Our approach relies on the competition-free property of the NoC TDM schedule provided by our execution model. We argue that such an approach enables shorter guaranteed NoC latencies that are better suited to benefit from a multi-cluster parallelization [21].

Mapping on other many-core processors or distributed systems. Puffitsch et al. [20] proposed an execution model and a framework to map dependent task sets on several multi/many-core architectures. However, the message passing latencies are derived from a measured WCTT which occurs in a situation of maximum contention while our approach relies on the contention-avoidance property provided by our execution model. In [4], the authors proposed an efficient heuristic to map Synchronous Dataflow models onto a time-triggered architecture but the assumptions on the system model differ from our approach. Indeed, no support is offered to either account for multi-rate applications or handle constraints coming from an execution model. In [7], Craciunas et al. proposed both a SMT and a MIP formulation of a co-scheduling problem handling simultaneously the generation of cores and network schedules. Since the authors assume a TTEthernet network and since other constraints such as the memory limitations are taken into account, the goal of the mapping algorithm is very close to ours. The main differences reside in the system model as they assume preemptive scheduling at the tt-tasks level while we assume non preemptive scheduling at the sub-task level. Moreover, all the tt-tasks are assumed to be pre-assigned to an end-system while our approach includes the placement of sub-tasks to cores together with the core and NoC scheduling.

7. CONCLUSION

In this paper, we have proposed an approach enabling the automatic parallelization of large applications onto the MPPA architecture. This approach meets the requirements imposed by the execution model providing the property of strong temporal isolation between co-running applications thanks to spatial and temporal partitioning. We provided a formulation of the mapping problem as a CSP using the notion of time-intervals and we demonstrated the ability of the approach to deal with large problems with an industrial case study. Overall, the mapping procedure that we described in this paper together with the real world imple-

mentation of our execution model that we presented in [19] form a complete work-flow enabling an end-to-end integration of real-time applications on the KALRAY MPPA®-256.

In the future, we will propose additional guidelines to help the application designers in the process of defining their budgets and we will investigate the optimization possibilities of the memory mappings in order to improve the NoC utilization and the overall performances.

8. REFERENCES

[1] Affordable Safe And Secure Mobility Evolution (ASSUME). http://www.assume-project.eu/.

[2] Parallel Computing for Time and Safety Critical Applications (CAPACITES). http://capacites.minalogic.net/en/.

[3] M. Becker, D. Dasari, B. Nicolic, B. Åkesson, V. Nélis, and T. Nolte. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform. In *28th Euromicro Conference on Real-Time Systems (ECRTS'16)*, July 2016.

[4] T. Carle, M. Djemal, D. Potop-Butucaru, and R. De Simone. Static mapping of real-time applications onto massively parallel processor arrays. In *14th International Conference on Application of Concurrency to System Design (ACSD'14)*, pages 112–121, 2014.

[5] CAST (Certification Authorities Software Team). Position Paper on Multi-core Processors - CAST-32, 2014.

[6] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[7] S. S. Craciunas and R. S. Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems*, 52(2):161–200, 2016.

[8] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti. Guaranteed Services of the NoC of a Manycore Processor. In *2014 International Workshop on Network on Chip Architectures (NoCArc'14)*, pages 11–16, 2014.

[9] F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[10] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. D. de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, pages 1–51, 2015.

[11] R. Gorcitz, E. Kofman, T. Carle, D. Potop-Butucaru, and R. De Simone. On the Scalability of Constraint Solving for Static/Off-Line Real-Time Scheduling. In *13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'15)*, pages 108–123, 2015.

[12] A. Graillat. Code Generation of Time Critical Synchronous Programs on the Kalray MPPA Many-Core architecture. In *7th Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'16)*, 2016.

[13] X. Jean, L. Mutuel, D. Regis, H. Misson, G. Berthon, and M. Fumey. White Paper on Issues Associated with Interference Applied to Multicore Processors, 2016.

[14] Y.-K. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, Dec. 1999.

[15] P. Laborie and J. Rogerie. Reasoning with Conditional Time-intervals. In *21st International Florida Artificial Intelligence Research Society Conference (FLAIRS'08)*, 2008.

[16] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In *22nd International Florida Artificial Intelligence Research Society Conference (FLAIRS'09)*, 2009.

[17] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of Global EDF for Parallel Tasks . In *25th Euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[18] Q. Perret, P. Maurère, É. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. Predictable composition of memory accesses on many-core processors. In *8th Conference on Embedded Real Time Software and Systems (ERTS'16)*, 2016.

[19] Q. Perret, P. Maurère, É. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. Temporal isolation of hard real-time applications on many-core processors. In *22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16)*, 2016.

[20] W. Puffitsch, É. Noulard, and C. Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

[21] W. Puffitsch, R. B. Sørensen, and M. Schoeberl. Time-division Multiplexing vs Network Calculus: A Comparison. In *23rd International Conference on Real Time and Networks Systems (RTNS'15)*, pages 289–296, 2015.

[22] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet. Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems. In *21st International Conference on Real-Time Networks and Systems (RTNS'13)*, pages 287–296, 2013.

[23] M. Qamhieh, L. George, and S. Midonnet. A Stretching Algorithm for Parallel Real-time DAG Tasks on Multiprocessor Systems. In *22nd International Conference on Real-Time Networks and Systems (RTNS'14)*, pages 13–22, Oct. 2014.

[24] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations.

[25] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-178C: Software considerations in airborne systems and equipment certification, 2011.

[26] R. Wilhelm and J. Reineke. Embedded systems: Many cores - many problems. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 176–180, 2012.