Multi-Constraint Multi-Processor Resource Allocation

Amir R. B. Behrouzian[1], Dip Goswami[1], Twan Basten[1,2], Marc Geilen[1], Hadi Alizadeh Ara[1]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]TNO Embedded Systems Innovation, Eindhoven, The Netherlands
email: {a.r.baghban.behrouzian, d.goswami, a.a.basten, m.c.w.geilen and s.h.seyyed.alizadeh}@tue.nl

***Abstract*—This work proposes a Multi-Constraint Resource Allocation (MuCoRA) method for applications from multiple domains onto multi-processors. In particular, we address a mapping problem for multiple throughput-constrained streaming applications and multiple latency-constrained feedback control applications onto a multi-processor platform running under a Time-Division Multiple-Access (TDMA) policy. The main objective of the proposed method is to reduce resource usage while meeting constraints from both these two domains (i.e., throughput and latency constraints). We show by experiments that the overall resource usage for this mapping problem can be reduced by distributing the allocated resource (i.e., TDMA slots) to the control applications over the TDMA wheel instead of allocating consecutive slots.**

Keywords—Resource allocation, multi-processors, throughput constraints, latency constraints, control applications, streaming applications, synchronous dataflow.

I. INTRODUCTION

Many application domains including automotive and healthcare systems require to run multiple applications simultaneously with stringent requirements on performance to ensure correct functionality. Often, the applications impose different domain-specific constraints for the correct functional behavior at the system-level. Example includes interventional X-Ray (iXR) systems where precision motion control applications for patient support execute together with compute-intensive image/video processing applications. In the one hand, the precision of the motion controller is critical for patient's safety and on the other hand, the image quality after the processing is crucial for the clinical purposes. Together, design of such systems requires to meet different types of constraints. A common practice is to consider the different types of constraints in an isolated fashion to reduce the design complexity. While such design flow is suitable for integration for large industrial systems and reduces the design effort, it increases the overall cost of the system due to poor resource efficiency. This particularly becomes relevant when it comes to embedded implementations since resource (e.g., computation and communication) is often limited and shared in such settings. In this context, multiprocessing compute platforms are widely used in almost all such industrial domains [1]. Appropriate treatment of different domain-specific constraints in the design process can potentially improve the overall design efficiency. Here, an important design question is the mapping of application tasks from multiple functional domains with different types of constraints onto the *shared* resource (e.g., processors in the case of compute resource). In this work, we consider the mapping problem of combination of feedback control applications with requirement on Quality of Control (QoC) and streaming applications with image/video quality requirement. Fig. 1 shows the setup under consideration.

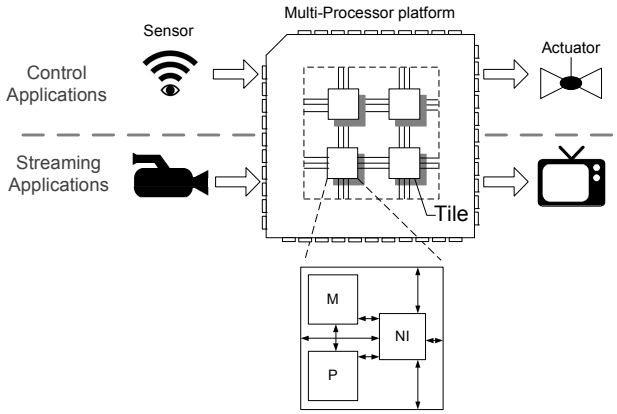

Fig. 1. Setup under consideration: latency-constrained feedback control applications sharing processors with throughput-constrained streaming applications. P: processor, M: memory and NI: network interface

Generally, the quality requirements of streaming applications can be translated into the minimum *throughput* constraints. Similarly, the QoC requirement of a feedback control application can be translated into the constraint on the maximum *sampling period* [2]. Further, depending on the model of the feedback control loops, the constraint on the sampling period can be expressed as a *maximum latency* or a *deadline* allowed by the corresponding tasks. Thus, for the setup shown in Fig. 1, the mapping problem deals with a combination of applications with throughput and latency constraints. System-level functionality requires to meet both constraints at the application level. In this process, not only performance of individual applications needs guarantees, but performance of one application should not directly influence that of others. Achieving such system-level requirements on predictability or timing is particularly challenging on shared platforms due to interference between applications through resources.

One well-known approach to achieve the above-mentioned temporal predictability on a shared resource is by *budget schedulers* that guarantee a fixed access time for every scheduled application (task) in a given time interval [4]. Therefore, applications would have access to the resource in a time-isolated fashion providing higher predictability. Time-Division Multiple-Access (TDMA) is a budget scheduler that defines a periodic working cycle for resources [5]. Each cycle is divided into a number of *time slots*. Usually, each time slot is assigned to one task (of an application). Minimization of

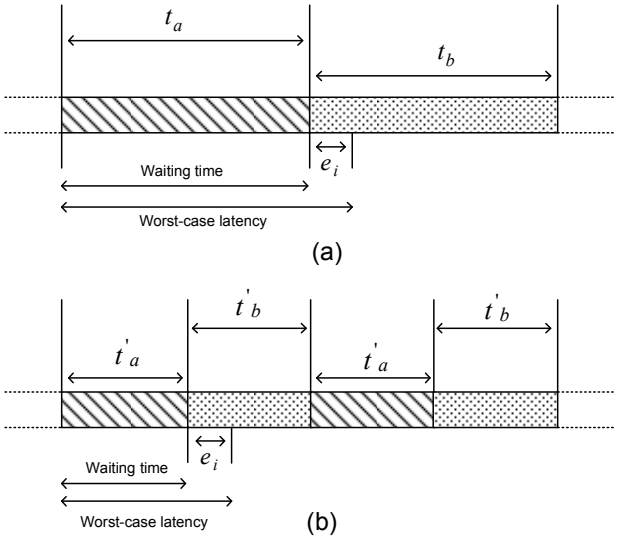

Fig. 2. Example: each work cycle of a processor is divided into (a) two, and (b) four slots each assigned to either steaming applications or control applications. Streaming applications are processed during times t_a and t'_a while control applications are processed during times t_b and t'_b.

resource usage then implies that the minimum number of time slots is allocated to an application that guarantees the required performance.

In the above context, a *resource allocation* strategy defines a policy for resource sharing among applications with timing predictability and independence among them. For worst-case performance guarantees, a resource allocation strategy often considers worst-case timing behavior of an application to determine the budget it needs to meet the performance constraints (even in the worst case). Budget allocation can either be performed by allocating specific slots to the applications [3] or it can be performed by allocating a percentage of slots to an application without specific slot-to-application mapping. Based on the timing constraints at the application level (i.e. throughput/latency constraints), different resource allocation strategies can be used [6],[7].

Key observation and our approach: Traditionally, resource allocation for applications with different types of constraints on shared resources is performed by time-isolated resource-to-application mapping [7]. Fig. 2 (a) and (b) show two examples of such allocation in view of the setup illustrated in Fig. 1. In Fig. 2(a), the t_a interval within a work cycle is allocated to the streaming applications while t_b is allocated to the control applications. While such scheduling policy provides application-level independence, it can be noticed that no sensing update for the control applications is possible during t_a and the sampling period of the control loops must be longer than this duration. That is, the minimum sampling period of the control (i.e., latency-constrained) applications would be the sum of t_a and the execution time of the application e_i (i.e., $(t_a + e_i)$ in Fig. 2(a)). Towards meeting a given latency constraint, one needs to allocate enough resources in order to take into account the above worst-case scenario, i.e., t_a should be small enough such that $(t_a + e_i)$ meets the latency constraint.

As opposed to Fig. 2 (a) in which there is only one partition each for the streaming and control applications, Fig. 2 (b) shows an allocation with distributed resource budget to control applications. The resources allocated to the latency- and throughput-constrained applications are still temporally isolated like the example in Fig. 2 (a). With the distributed allocation shown in Fig. 2 (b), the sampling period for the control applications would be shorter than that of example in Fig. 2 (a). That is, $(t_a + e_i) > (t'_a + e_i)$. This further implies a control application with shorter sampling period can be scheduled with such distributed allocations. Since a shorter sampling period can potentially be translated to a higher QoC, this observation is particularly relevant for the feedback control loops. While this observation suggests that the resource should be as distributed as possible for the control applications leading to a high number of time slots, each time slots needs to accommodate certain *overhead* for *context switching* from one application to the other. Such overhead restricts the granularity of the time slots allowed by the system since a higher number of slots in a work cycle imposes a higher overhead (leading to a poor resource efficiency). In this work, we assume that the time slots are long enough to ignore such overhead. Moreover, these choice of parameters of a work cycle is a design decision often taken at the early design phase and this work deals with a given set of work cycle parameters. In this work, we further consider the worst-case scenario for throughput-constraint applications. That is, the resource is allocated to meet the throughput constraint for any distribution. Thus, our observation for allocating resource to control applications can be integrated with any method that allocates budget (i.e., does not perform application-to-slot mapping) to streaming applications with throughput guarantees. For this purpose, we rely on the state-of-the-art method reported in [6] and the existing tooling support SDF^3 [11].

Our Contributions: This work proposes a resource allocation flow for co-mapping multiple streaming applications (with throughput constraints) and multiple control applications (with sampling period/latency constraints) on a multi-processing platform (as illustrated in Fig. 1). Considering both constraints in a single schedule and mapping with efficient resource usage are two main contributions of this work. Processors run under a TDMA policy with a given work cycle. The proposed method determines both (i) task-to-processor mapping (ii) and task-to-time slot mapping within a processor such that both throughput and latency constraints are met. The implementation considered the control applications can be modeled by a single task while Synchronous Data Flow (SDF) graphs are used to model the streaming applications. We refer to our method as Multi-Constraint Resource Allocation (MuCoRA).

In Section II, related work is discussed. Section III describes the setup under consideration. The multi-processor architecture is presented in Section IV. Section V illustrates the overall problem formulation. The proposed resource allocation strategy is described in Section VI. Section VII presents the experimental results and evaluation. Section VIII concludes.

II. Related Work

In view of the setup, we consider in this work (Fig. 1), the existing literature can be classified into two categories: (i) resource allocation and mapping methods for the feedback control applications and general latency-constrained applications (ii) resource allocation/mapping methods to meet or optimize throughput of streaming applications. While our work addresses a combination of (i) and (ii) and limited/no existing work address such problem, the literature on both (i) and (ii) are relevant. In the following, we present the state-of-the-art in these directions.

Over the last decade, there has been considerable amount of work on control/architecture co-design. The idea is to allocate/optimize shared communication/computation resource for implementing one/multiple control loops. This is in line with direction (i). The problem is addressed considering different abstractions both for system dynamics (i.e., control point of view) and the architecture model. The method in [13] formulates a dynamic program to find the optimal scheduling for multiple control tasks running under non-preemptive policy and similarly, [14] presents a method to find the optimal sampling periods (i.e., latency constraints) in a similar setup as [13]. In [15], the resource requirements of control loops are translated into a compute-bandwidth constraint (rather than latency constraint). In [16], [2] , scheduling control applications for both communication and computation resource is considered. In these classes of work, several scheduling policies in communication (e.g,, TDMA, Fixed-Priority Non-Preemptive) and computation (e.g., deadline monotonic, EDF, TDMA) are considered to derive the stability and performance region for a given combination of scheduling policies. The event-based controller proposed in [17] performs sampling (i.e., sensing, computing and actuating) only at the occurrence of events (e.g., crossing an error threshold) and saves communication bandwidth by avoiding unnecessary sampling. The mapping and scheduling problem for a combination of real-time and control applications is considered for automotive architectures in [2]. It should be noted that traditional real-time scheduling methods [9] are not directly applicable to schedule feedback control applications since it needs to explicitly consider the impact of jitter unlike deadline-driven real-time applications.

Along direction (ii), a large body of work has been reported on resource allocation strategy for mapping of throughput-constrained applications onto multi-processors [6], [19], [23]. [6] proposes resource allocation techniques for task binding and scheduling directly on an SDF graph of the throughput constraint applications. In [23], the throughput constraint streaming applications are modeled and mapped to the multiprocessor by an acyclic model in which an application task graph allows to run each tasks once. [19] encodes the problem of mapping and scheduling of dataflow applications on a multi-processor platform in the form of logical constraints and present it to a Satisfiability Modulo Theory (SMT) solver. Given several cost constraints (e.g. memory usage, throughput, static power consumption), the SMT solver uses a combination of search techniques and constraint propagation to find a set of Pareto optimal solutions. A key point in a resource usage efficient mapping method is how accurately the throughput is estimated. The more pessimistic throughput estimation is, the more resources are required. The worst-case throughput calculation is complicated. [21] proposes throughout analysis based on Homogeneous SDF (HSDF) graph. Therefore, throughput analysis needs to transform SDF graph of applications to HSDF graph, which leads to a large graph size. [20] propose a throughput analysis method which works directly on an SDF graph. Executing an SDF graph, this method makes and analyses its dynamic state-space. [22] improve throughput calculation for tighter throughput guarantees using max-plus algebra combined with worst-case resource availability curves.

The mapping and scheduling to answer both (i) and (ii) directly are not yet explored. However, some work try to address isolation of applications in shared resources. Based on Worst-Case Execution Time (WCET) of intra-application tasks, running on multi-processors, [24] proposes a bus scheduling method to share Network on Chip (NoC) among several processors. The approach in [25] uses virtualization method for scheduling several application in different shared resources (e.g. processors, interconnects and memory) among applications. In this method, using preemptive time-division multiplexing (TDM) among applications, each resources is divided into smaller virtual resources. This allows independent design, verification and execution.

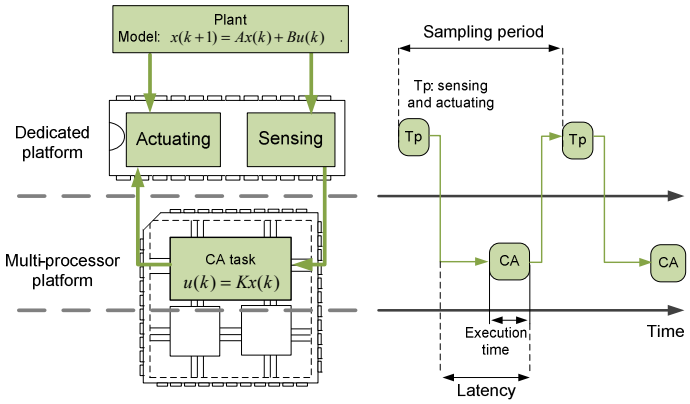

Fig. 3. Implementation and timing diagram of feedback control loop. Sensing and actuating tasks are performed in dedicated hardware; the CA task uses the multi-processor platform.

III. Setup under consideration

The setup under consideration is illustrated in Fig. 1. In the following, we describe the models of the control applications and streaming applications used in our analysis.

A. Control application

A feedback controller regulates behavior of a dynamic system – called *plant*. The plant is considered to be a linear time invariant (LTI) system modeled by a set of difference equations of the following form.

$$x(k+1) = Ax(k) + Bu(k) \qquad (1)$$

where A and B are the system matrices, $x(k)$ is the plant state and $u(k)$ is the control input. A feedback control loop performs three sequential operations: sensing, computing, and actuating. Fig. 3 shows implementation and timing diagram of these

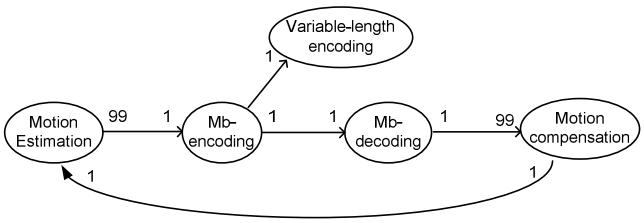

Fig. 4. SDF graph of H.263 encoder

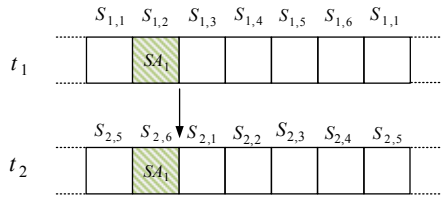

Fig. 5. Worst-case senario for data communication between tiles.

operations. The sensing operation obtains system *state* $x(k)$ from the plant periodically with period h. The period h is referred to as the *sampling period* (see Fig. 3). Next, the compute operation computes the control input $u(k)$. The compute operation is performed by the Control Application (*CA*) task. The execution of the *CA* task must be completed before the actuation operation starts. The time needed for computation of the *CA* task on a processor is referred to as *execution time* of the *CA* task. The *CA* task passes the control input *u(k)* to the actuation operation which applies to the control input to the plant. As shown in Fig. 3 sensing and actuating operations are performed together by a task T_p that runs on dedicated hardware in a time-triggered fashion with period h. The *CA* task runs on the multi-processor platform. We define the time interval from the point that the samples arrive to computation unit till the moment the *CA* task finishes as *latency,* which should be less than or equals to the sampling period *h* to guarantee that all sampled data are processed for actuation.

Let $CA = \{CA_1, CA_2, ...\}$ be the set of all *CA* tasks corresponding to different control applications. A control application $CA_i \in CA$ is characterized by a tuple (e_i, h_i) with $e_i \in \mathbb{N}$ (positive natural numbers) the execution time (in time units, further discussed in Section IV) and $h_i \in \mathbb{N}$ the sampling period (in time units) of the CA_i task.

B. Streaming applications

We use the SDF graph to model the streaming applications and for throughput analysis as suggested in a number of recent works [6][7]. We illustrate our proposed method considering some streaming applications such as the H.263 encoder [8] (see Fig. 4). In an application SDF graph (*application graph*), nodes (*actors)* communicate by sending data (*tokens*) over edges (*channels)*. Upon firing of an actor, input tokens are removed from its input channels and after a certain amount of time *(execution time)* new tokens are produced on its output channels; the numbers of consumed and produced tokens are given by the annotations (*rates*) attached to the channel ends in the graph. The execution time of some of the actors and their input/output rate may vary (e.g. for variable-length encoding actor in the example shown on Fig. 4) from once execution to another. By conservative overestimation it is possible to have a fix number for them. Let $SA = \{SA_1, SA_2, ...\}$ be the set of all streaming applications. Each application $SA_i \in SA$ ($i \in \mathbb{N}$) has a throughput constraint of λ_i actor firings per time unit, for some dedicated actor.

IV. Architecture Model

In this work, a tile-based template [6] is used as the multi-processor platform. Fig. 1 shows an example architecture platform with four connected tiles that process multiple control and streaming applications. Each tile contains a processor (P), a local memory (M), and a network interface (NI) that connects the memory to the local processor, and interconnections between tiles. Let $T = \{t_1, t_2, ...\}$ denotes the set of tiles under consideration.

All tiles run a TDMA scheduling policy. Under the TDMA policy, the resource access schedule repeats cyclically according to a cycle called a *time wheel*. The time wheel consists of multiple equally sized parts, called *time slots*. It should be noted that the context switching overhead between applications is negligible compared to the size of each time slot (as explained in the introduction). Many architectures [3] allow such choice of the TDMA configuration. Each time slot can be allocated to a different application during which the application can run without any interference from other applications. Each tile $t_i \in T$ has a time wheel consisting of w_i equally sized time slots. Without loss of generality, we consider one time slot to be unit *time*. For each tile $t_i \in T$, we define S_i as the set of all time slots as $S_i = \{S_{i,1}, S_{i,2}, ..S_{i,w_i}\}$.

We consider the communication between tiles to be *asynchronous.* The presented analysis on timing behavior of data communicated between tiles is based on the worst-case timing scenario. In other words, any data travelling from one tile to another, is assumed to arrive at the destination tile at the moment such that it will have to wait the most until it reaches the next assigned time slot. Fig. 5 illustrates the worst-case timing scenario for data that is communicated between two tiles t_1 and t_2 with $w_1 = w_2 = 6$. As shown in Fig. 5, $S_{1,2}$ in t_1 and $S_{2,6}$ in t_2 are assigned to application SA_1. Therefore, the worst-case scenario for data sent from t_1 to t_2 is to arrive just after $S_{2,6}$ finishes. In this scenario, the data has to wait for $w_2 - 1 = 5$ time units before its processing starts.

V. Problem Statement

We can now formalize the precise problem description for this paper. For a set of inputs as follows:

I1. a set of streaming applications SA with $SA_i \in SA$ modeled by an SDF graph,

I2. a set of control applications CA with $CA_i \in CA$ characterized by tuples (e_i , h_i),

I3. a set of tiles T, each tile $t_i \in T$ running a TDMA policy with a time wheel size of w_i time slots,

we address the following resource allocation problem:

P1. binding actors (for SA) and tasks (for CA) to the tiles and

P2. assigning time slots (i.e., $S_{i,j}$) in a tile to applications (i.e., SA and CA) bound to that particular tile

such that the following constraints are satisfied:

C1. throughput constraints λ_i of the streaming applications $SA_i \in SA$,

C2. latency constraints h_i of control applications $CA_i \in CA$.

Given I1-I3, a solution to P1 and P2 which satisfies both C1 and C2 for all the applications (meeting both performance requirements) is a valid solution to the above resource allocation problem.

VI. MuCoRA: Resource Allocation Flow

Resource allocation is done in two stages (see Fig. 6): (i) binding the applications to tiles (i.e., P1) (ii) assigning the time slots to the applications (i.e., P2). In the following, we describe the two stages for the control and the streaming applications.

A. Control application mapping

Given I2 and I3, we address P1 and P2 for control applications in this subsection. Resource allocation for the control applications consists of two stages. In the first stage, each control application is bound to only one tile (i.e. P1). First, this avoids latency overhead due to asynchronous interaction among the tiles. Second, this avoids connection delay caused by the interconnects. Further, the control tasks are computationally light making them easier to run on a single tile. The binding solution of control applications $CA_i \in CA$ must guarantee that the latency constraint C2 is met.

We use a load balancing policy for task-to-tile binding similar to [6]. Resource usage for the control applications corresponds to the number of time slots that is required to meet C2. We define the load function of control application CA_i as

$$l_{CA_i} = \frac{e_i}{h_i}. \qquad (1)$$

It determines the minimum fraction of a time wheel that is required for a control application to meet C2. It is noteworthy that a total load of less than 1 is a necessary condition for schedulability on a single tile, i.e. [9]

$$\sum l_{CA_i} < 1.$$

To balance the load of control applications among tiles, we sort the applications in decreasing order of their loads. To balance the load among tiles we bind applications in that order

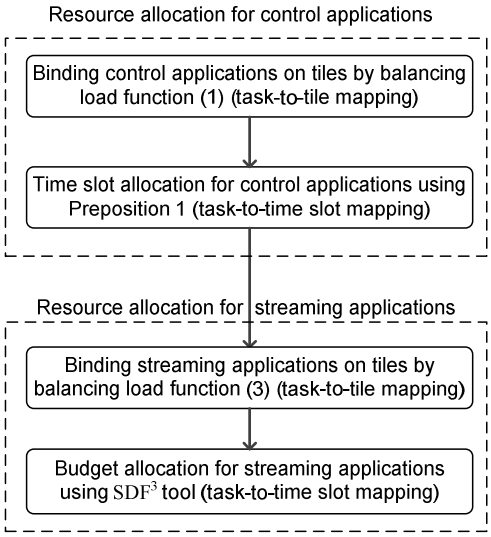

Fig. 6. Resource allocation flow for mapping several control and streaming applications on a multi-processor platform.

to the tiles. For example, assume binding of four control applications with load function values of

$$[l_{CA_1}, l_{CA_2}, l_{CA_3}, l_{CA_4}] = [0.6, 0.4, 0.3, 0.1].$$

The best case for binding these applications to two tiles according to the resource load balancing policy is to bind CA_1 and CA_4 to one tile and CA_2 and CA_3 to the other tile. This addresses P1 for the control applications CA. In the following, we describe how time slots are assigned to the control applications towards addressing P2.

Definition 1. For any time slot $S_{i,j}$ (i.e. slot number j in tile number i), we define an occupation variable $X_{i,j,m}$ that takes value 1 when slot $S_{i,j}$ is assigned to application CA_m and takes value 0 if not.

For each application mapped on a tile we initially allocate all available time slots, i.e., $X_{i,j,m} = 1$ for all j which are not allocated to other applications and next, we attempt to unassign the time slots that are *unnecessary*. That is, we *check* if an assigned slot can be unassigned without violating constraint C2.

Proposition 1. Assigned time slot $S_{i,j}$ (i.e. slot number j in tile number i) to a control application CA_m can be unassigned without violating C2 if the following inequality is satisfied for all $j < n < j + h_m - 1$,

$$\sum_{k=n-h_m+1}^{n} X_{i,k,m} \geq e_m + 1, \qquad (2)$$

where e_m and h_m are execution time and sampling period of the application CA_m respectively.

Proof. For a schedule that already satisfies latency constraint C2 of an application CA_m, time slot $S_{i,j}$ which is assigned to

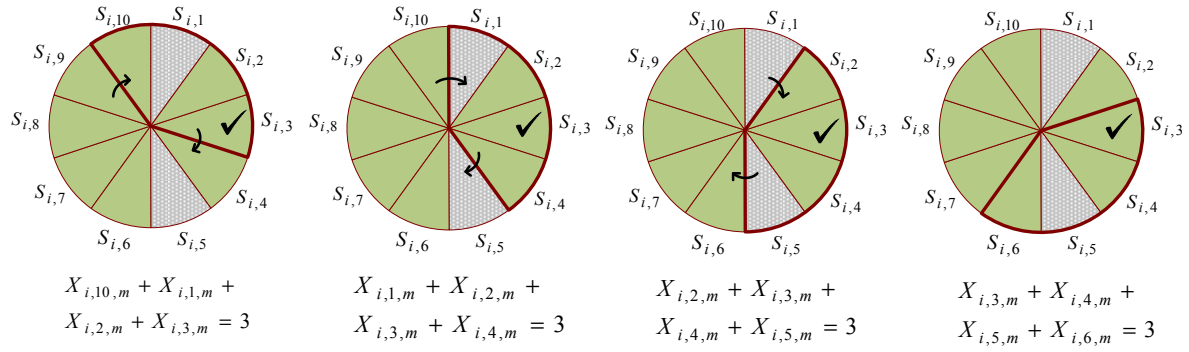

Fig. 7. All combinations of 4 consecutive time slots including $S_{i,3}$ are checked to find if $S_{i,3}$ can be unassigned of application CA_m, with (e_m, h_m)=(2,4), w_i=10

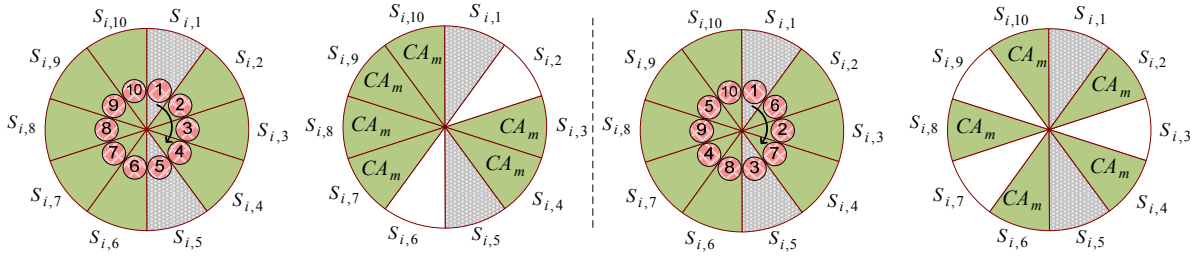

Fig. 8. All combination of 4 consecutive time slots including $S_{i,1}$ are checked to find if $S_{i,1}$ can be unassigned of application CA_m.

application CA_m, can be unassigned if there is no combination of h_m consecutive slots, including $S_{i,j}$, for which the sum of occupation variables is less than e_m+1. This comes from the idea that a TDMA schedule satisfies latency constraint C2 of an application CA_m only if e_m time slots are allocated to CA_m in every h_m consecutive time slots. Equation (2) computes the total execution time in every h_m slots. This completes the proof.

Fig. 7 illustrates the above proposition by checking if slot $S_{i,1}$ of tile t_i that is already assigned to control application CA_m with (e_m, h_m) = (2,4) can be unassigned. In this example, slots $(S_{i,1}, S_{i,5})$ are already assigned to another applications and cannot be assigned to CA_m. So, $X_{i,1,m}=0$ and $X_{i,5,m}=0$.

Next, all combinations of four consecutive time slots including $S_{i,1}$ are checked to ensure that there is always more than one time slot assigned to the application (see Fig. 7). If so, $S_{i,1}$ can be unassigned (i.e. $X_{i,1,m}=0$) while there is at least one time slot allocated to CA_m which guarantees satisfaction of the latency constraint C2.

The minimum e_m slots in each h_m consecutive slots should be allocated to application CA_m to meet C2. Thus, the minimum load of an control application in each time wheel of tile t_i is obtained by

$$l_{CA_i} = \left\lceil \frac{e_i}{h_i} w_i \right\rceil. \quad (3)$$

As illustrated in Fig. 2, a distributed allocation of resource for control applications improves the resource efficiency requiring less time slots. Therefore, we aim *to minimize the maximum distance between any two consecutive slots that are assigned to a control application*. We assign the slots in such a way that the maximum number of slots between every two assigned time slots is

$$\left\lceil \frac{h_m}{e_m} \right\rceil. \quad (4)$$

In the above process, some slots might already be occupied by other applications and it is not possible to assign those slots to the control application. In those cases, more slots are needed to guarantee the performance, i.e., meet the latency constraint C2. To maximize the distribution in resource allocation, the order of slots that are considered for unassignment is chosen periodically with a period as Eq. (4).

Fig. 8 shows two orders of selecting time slots in example in Fig. 7 for unassignment. In Fig 8 (a), consecutive available slots are considered for unassignment. Fig. 8 (b) shows the resulting allocation. In Fig 8 (c), the time slots are chosen with a period of 2 obtained by Eq. (4). The order of selecting the slots would be

$$\{S_{i,1}, S_{i,3}, S_{i,5}, S_{i,7}, S_{i,9}, \ldots\}$$

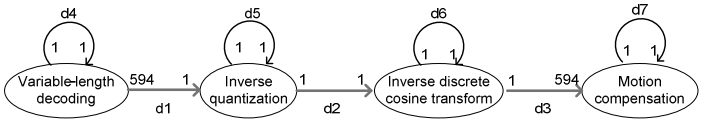

Fig. 9. SDF graph of H.263 decoder.

$\{S_{i,1}, S_{i,5}\}$ are not allowed to be assigned to CA_m. By checking all other slots, we find that $\{S_{i,3}, S_{i,7}, S_{i,9}\}$ can be unassigned. Fig. 8 (d) shows the resulting allocation. Although both schedules (Fig. 8 (b) and Fig. 8 (d)) satisfy the latency constraints of application CA_m, the second schedule in Fig. 8 (d) requires less time slots than the one shown in Fig. 8 (b). Maximum distance between two consecutive assigned time slots in the second solution is 2 as per Eq. (4). Thus, the results are in line with the observation described in introduction.

B. Mapping of streaming applications

We use the existing method of [6] as implemented in the SDF^3 tool [11] to allocate resources to streaming applications.

Binding actors to tiles. This step binds every actor of a streaming application SDF graph to a tile. The actors that have large impact on the throughput are bound first. The impact of an actor is too expensive to compute exactly. Therefore it is estimated for each cycle in the SDF graph in which is participates, by the total computation time of the actor divided by the pipelining parallelism in that cycle. The pipelining is, in turn, estimated by the total number of initial tokens in the cycle, normalized by the actor firing rates on the corresponding edge on which they reside. Since an actor may be part of multiple cycles in the SDF graph, the most critical cycle is considered an estimate for the impact of the actor.

After sorting the actors according to their impact on throughput they are bound to the tiles. Binding tries to achieve load balancing, where the load of a tile is estimated as the total execution time of firings of actors bound to the tile divided by the total execution time of all actor firings. The main strategy is a greedy algorithm that binds an actor to the least loaded tile. The SDF^3 tool balances the load of tiles not only in terms of computation, but also in terms of memory usage and bandwidth requirements, both of which we do not consider in this paper. SDF^3 allows us however to set weighting factors for these aspects, which then enables us to perform load balancing in terms of computation only.

Constructing firing order. After binding the actors of an application to the tiles, the static order of their firing is obtained using a list-scheduler.

Budget allocation. A binary search algorithm is used in combination with a throughput analysis algorithm to find the minimum number of time slots required on each tile to satisfy the required throughput rate. The throughput analysis algorithm that is used is conservatively pessimistic, both in view of which of the time slots in the time wheel will be allocated, and in terms of the misalignment of slots on different tiles due to asynchronous execution of their TDMA wheels. Thus, problem P1 is solved for the streaming applications and problem P2 is trivially satisfied by any slot allocation we may choose. After obtaining the budget (i.e. number of time slots) that is required for streaming application by SDF^3, we allocate the time slots that are not allocated to control application. The rest of time slots that are not allocated to none of the applications remain empty. They can be used to map more applications.

TABLE I. CONTROL APPLICATIONS: UNIT IS TDM TIME SLOTS

	Execution time	Sampling period
CA1	2	13
CA2	2	10
CA3	1	6
CA4	2	12
CA5	5	19
CA6	2	8
CA7	3	15

VII. EXPERIMENTAL RESULTS

System description: We consider four control loops and two streaming applications for evaluating our MuCoRA design flow. The multi-processor platform has two tiles both running under TDMA policy with a time wheel size of 20 slots. The streaming applications are an H.263 video encoder (SA_1) [8] and an H.263 decoder (SA_2) [10]. The SDF graphs of the streaming applications are shown in Fig. 4 and Fig. 9 respectively. The control loops have execution time and sampling period as shown in Table I. From the seven control loops, we choose 35 subsets of four control loops for our experiments. This shows how sensitive our results are to the parameters of the control applications. We choose the execution time and the sampling period of control applications and time wheel size to be of the same order of magnitude. The periods of the control applications are from 6 to 15 time slots while the time wheel is 20 slots. The impact of significantly longer or shorter sampling periods than the time wheel size is not explored in this work. Intuitively, the impact of the relative length of sampling periods affects the scalability while this work focuses on establishing the potential benefit of the proposed method.

Allocation with MuCoRA: We first apply MuCoRA to solve the mapping problem. Table II shows one result. CA_1 and CA_2 are mapped to Tile 1 while CA_3 and CA_4 are mapped to Tile 2. Fig. 10 (a) illustrates the resource allocation in Tile 1. Time slots $\{S_{1,1}, S_{1,8}, S_{1,14}\}$ are allocated to CA_1 (shown in Fig. 10(a) in blue), $\{S_{1,2}, S_{1,7}, S_{1,12}, S_{1,17}\}$ are allocated to CA_2 (shown in Fig. 10(a) in red), $\{S_{1,3}, S_{1,5}, S_{1,9}, S_{1,13}\}$ are allocated to SA_1, $\{S_{1,18}, S_{1,19}\}$ are allocated to SA_2 and $\{S_{1,4}, S_{1,6}, S_{1,10}, S_{1,11}, S_{115}, S_{1,16}, S_{1,20}\}$ are empty.

Evaluation: For evaluating the proposed method, we consider an allocation where only two partitions are considered – one for the feedback control applications and one for the streaming applications (similar to Fig. 2(a)) [5]. We refer to this method as partitioning method. In this case, since multiple control loops are sharing the same partition (e.g., t_b in Fig. 2(a)), we need an arbitration policy to schedule the applications in the control

TABLE II. RESOURCE ALLOCATION FOR MUCORA AND PARTITIONING METHOD.

Tile	Mapping method	CA_1	CA_2	CA_3	CA_4	H.263 Enc.(SA1)	H.263 Dec.(SA2)	$SA_{1\text{ and }2}$ occupation	Control application occupation	Empty portion of tile
1	MuCoRA	X	X			X	X	30%	35%	35%
	Partitioning method							30%	60%	10%
2	MuCoRA			X	X	X	X	45%	15%	40%
	Partitioning method							45%	50%	5%

partition. For this purpose, we adapt the EDF (Earliest Deadline First) policy of [18]. For the mapping of streaming applications, we use SDF^3 described in Section VI.B. SDF^3 assigns a budget and does not consider the exact slot allocation. Therefore, the resource budget for the streaming applications is identical to MuCoRA. The size of the partition for streaming applications equals the computed budget. The partition size for control applications is derived from the worst-case arrival of control samples relative to the partition, which is just after the partition for control applications. The size of control application partition is therefore obtained such that the samples are executed before deadline. Let w denotes the size of time wheel, t_s and t_c denote the size of the partition of streaming and control application respectively (see Fig 10.). Then, for this example, the empty portion of the time wheel is obtained as follows:

$$t_e = \min\{(l_1 - e_1 - e_2 - t_s), (l_2 - e_2 - t_s)\},$$

where, l_1 and l_2 are deadline of CA_1 and CA_2 and e_1 and e_2 are execution time of CA_1 and CA_2. Next, t_c is obtained by $t_c = w - t_s - t_e$. For the example, as MuCoRA, the partitioning method maps CA_1 and CA_2 to Tile 1 and CA_3 and CA_4 to Tile 2. Fig. 10 illustrates the working cycle in Tile 1. As shown in Table II, the partition for streaming applications takes 30% of the time wheel) and the one for control applications 60%. The resource budget considers the worst-case arrival of samples, at the end of the partition assigned to control applications; this is shown by arrows in Fig. 10(b). All samples then have to wait until the end of the partition allocated to the streaming applications.

Comparison and discussion: We compared results of both methods for all possible 35 combinations of four out of seven control loops shown in Table I. For both methods, we measure the resource usage by the percentage of assigned time slots of the time wheel after mapping both categories of applications. The resource efficiency is higher if the percentage of the unassigned time slots is higher. Table II shows the empty and allocated portions of two tiles. For these particular examples, MuCoRA comes up with an allocation with 18% more empty space on average over all combinations. Table II provides one example of the 35 experiments. The experiments show that MuCoRA is robust to the changes in control parameters, with a minimum saving of 15% across all experiments.

VIII. CONCLUSIONS

In this work, we addressed a mapping problem of multiple control and streaming applications onto a multi-processor. While state-of-the-art resource allocation methods either consider one type of constraint or translate all constraints to one category, our

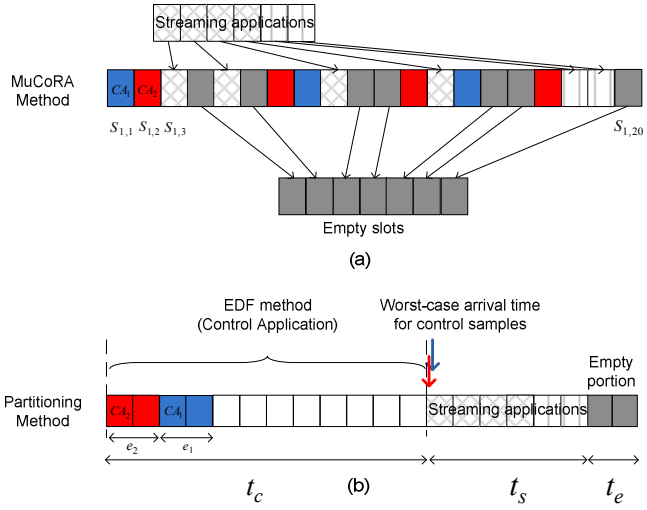

Fig. 10. Multi-constraint application mapping on Tile 1 with two methods, (a) MuCoRA and (b) partitioning method.

method treats the constraints from different domains separately. We have shown by experiments that distributing the allocated resources reduces the resource requirement by control applications and that this improves overall resource efficiency. As future work, we plan to explore the impact of the relative length between sampling periods (i.e., latency constraints) of the control applications.

ACKNOWLEDGMENT

This research was supported by the ARTEMIS joint undertaking under grant agreement no 621439 (ALMARVI).

REFERENCES

[1] K. Zhang, Y. Yao, O. Labanni, Z. Lu, X. Wu, "A New Universal-Environment Adaptive MultiProcessor Scheduler for Autonomous Cyber-Physical System", Proceedings International Conference on Information Systems (ICIS), 2012.

[2] D. Goswami, M. Lukasiewycz, R. Schneider and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software", Proceedings Design, Automation and Test in Europe (DATE), 2012.

[3] A. B. Nejad, A. Molnos, K. Goossens, "A Software-Based Technique Enabling Composable Hierarchical Preemptive Scheduling for Time-Triggered Applications", Proceedings Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013.

[4] G. C. Buttazzo, "Hard Real-Time Computing System", Springer, Virginia, 2011.

[5] B. Akesson, A. Minaeva, P. Sucha, A. Nelson, Z. Hanzalek, "An Efficient Configuration Methodology for Time-Division Multiplexed Single

Resources", Proceedings Real-Time and Embedded Technology and Application Symposium (RTAS), 2015.

[6] S. Stuijk, T. Basten, M.C.W. Geilen and H. Corporaal, "Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs" Proceedings Design Automation Conference (DAC), 2007.

[7] S. Sriram, S. Bhattacharyya, "Embedded Multiprocessors: Scheduling and Synchronization", Marcel Dekker, 2000.

[8] F. Siyoum, B. Akesson, S. Stuijk, K. Goossens, and H. Corporaal, "Resource-Efficient Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration" Proceedings Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011.

[9] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments," Journal of the Association for Computing Machinery, 20(1), 40–61, 1973.

[10] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear-Programming Based Techniques for Synthesis of Network-on-Chip Architectures," IEEE Transactions on Very Large Scale Int. Sys, 14(4), 407–420, 2006.

[11] S. Stuijk, M. Geilen, T. Basten, "SDF3: SDF For Free", Proceedings Application of Concurrency to System Design (ACSD), 2006.

[12] P. Marti, J. M. Fuertes, G. Fohler, K. Ramamritam, "Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling Issues", Proceedings Real-Time Systems Symposium (RTSS), 2002.

[13] A. Cervin, P. Alriksson, "Optimal On-Line Scheduling of Multiple Control Tasks: A Case Study", Proceeding of Euromicro Conference on Real-Time Systems (ECRTS), 2006.

[14] E. Bini, A. Cervin, "Delay-Aware Period Assignment in Control Systems", Proceeding of Real-Time Systems Symposium (RTSS), 2008.

[15] A. Aminifar, E. Bini, P. Eles, Z. Peng, "Bandwidth-efficient controller-server co-design with stability guarantees", Proceedings Design, Automation and Test in Europe (DATE), 2014.

[16] P. Naghshtabrizi, J. P. Hespanha, "Analysis of distributed control systems with shared communication and computation resource", Proceedings American Control Conferences (ACC), 2009.

[17] X. Wang and M. Lemmon, "Event-triggering in distributed networked control systems", IEEE Transactions on Automatic Control, 56(3):586–601, 2011.

[18] A. K. Mok, X. A. Feng, D. Chen, "Resource Partition for Real-Time Systems," Proceedings Real-Time and Embedded Technology and Application Symposium (RTAS), 2001.

[19] P. Tendulkar, "Mapping and Scheduling on Multi-Core Processors Using SMT Solver", PhD thesis, L'Université de Grenoble, 2014.

[20] A.H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, M. Mousavi, "Throughput Analysis of Synchronous Data Flow Graphs", Proceedings Application of Concurrency to System Design (ACSD), 2006.

[21] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms", IEEE Transactions on Design Automation of Electronic Systems, 9(4):385–418, 2004.

[22] F. Siyoum, M. Geilen, H. Corporaal, "Symbolic Analysis of Dataflow Applications Mapped onto Shared Heterogeneous Resources", Proceedings Design Automation Conference (DAC), 2014.

[23] J. Hu, R. Marculescu, "Energy- and Performance-aware mapping for regular NoC Architecture", IEEE Transactions on Computer-Aided Design of Int. Circuits and Sys. 24(4), 551-562, 2005.

[24] A. Andrei, P. Eles, Z. Peng, J. Rosen, "Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip", Proceedings VLSI Design (VLSID), 2008.

[25] K. Goossens, A. Azevedo, K. Chandrasekar et al., "Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow", ACM SIGBED, 10(3), 23-34, 2013.