Optimized Scheduling of Multi-IMA Partitions with Exclusive Region for Synchronized Real-Time Multi-Core Systems

Jung-Eun Kim*, Man-Ki Yoon*, Sungjin Im†, Richard Bradford§, and Lui Sha*
*Department of Computer Science, University of Illinois at Urbana-Champaign, {jekim314,mkyoon,lrs}@illinois.edu
†Department of Computer Science, Duke University, sungjin@cs.duke.edu
§Rockwell Collins, rmbradfo@rockwellcollins.com

***Abstract*—Integrated Modular Avionics (IMA) architecture has been widely adopted by the avionics industry due to its strong temporal and spatial isolation capability for safety-critical real-time systems. The fundamental challenge to integrating an existing set of single-core IMA partitions into a multi-core system is to ensure that the isolation of the partitions will be maintained without incurring huge redevelopment and recertification costs. To address this challenge, we developed an optimized partition scheduling algorithm which considers exclusive regions to achieve the synchronization between partitions across cores. We show that the problem of finding the optimal partition schedule is NP–complete and present a Constraint Programming formulation. In addition, we relax this problem to find the minimum number of cores needed to schedule a given set of partitions and propose an approximation algorithm which is guaranteed to find a feasible schedule of partitions if there exists a feasible schedule of exclusive regions.**

I. Introduction

The avionics industry has widely adopted the Integrated Modular Avionics (IMA) architecture [1], [2], which enables real-time functions to run within partitions that are temporally and spatially isolated from one another. The partitioning principle has also supported the avionics development and certification processes. Meanwhile, over the past decade, trends in microchip technology have led designers to assemble devices into multiple microprocessor cores for achieving continued increases in computational power. Multi-core systems are attractive for avionics applications because they offer the potential for more easily achieving size, weight, and power (SWaP) requirements.

Migrating an existing set of pre-certified single-core avionics IMA systems into a multi-IMA multi-core system, however, is not a trivial task. The fundamental challenge is to ensure that the temporal and spatial isolation of the partitions will be maintained without incurring huge recertification costs. One issue with meeting this challenge is that a multi-core system has lower clock speed than that of a single-core system due to temperature and power consumption constraints, then a direct mapping, such as a one-to-one mapping, may not work. Another issue is *synchronization* which is required for exclusive transactions such as I/O. For example, some hardware devices such as graphic cards or storage units do not admit multiple accesses.

In this paper, we investigate the problem of scheduling IMA-type partitions on multi-core systems. To the best of our knowledge, this is the first work that considers an exclusive region on multi-core IMA partitions. We formalize the problem as a *multi-IMA partition scheduling optimization* problem and address the aforementioned challenges. In our model, a partition consists of two logical regions: *solo-* and *execution-partition* (Fig. 1). The solo-partition concept already has a special role in IMA-based avionics systems, mainly that of performing I/O transactions (also known as *the device management partition* in [2]). Since data to be used in an execution-partition should be made ready during its solo-partition, an execution-partition follows its solo-partition. In a multi-core setting, we require a solo-partition to be an *exclusive region* to synchronize partitions across cores. That is, only one solo-partition is allowed to be scheduled at a time instant. By contrast, execution-partitions do not need to satisfy exclusiveness. Hence a partition can be feasibly scheduled on a core as long as it does not interfere with the solo-partitions of other partitions and does not overlap with other partitions assigned to the same core. In addition, each partition is *strictly periodic* and *nonpreemptive*. This supports temporal and spatial isolation among partitions. With this model, we formalize the following multi-IMA scheduling optimization problems.

a) Straight Mapping (StraightMapping): Given a set of single-core IMA systems, their partitions, and a multi-core system, we are required to construct a set of local core schedules that satisfy the exclusiveness of solo-partitions using one-to-one mapping from a single-core system to a core of the multi-core system. In other words, all partitions from a single core must be scheduled on the same core of the multi-core system (they can be rescheduled within the same core) as illustrated in Fig. 2 (a). We show that this problem is *strongly NP–complete* even when all solo-partitions have at most a *unit* size; without solo-partitions, the one-to-one mapping always gives a feasible schedule. This would imply that it does not admit an efficient algorithm even if all input parameters are small. We believe that our results show that exclusive solo-partitions add another layer of complexity to multi-IMA scheduling problems. On the positive side, we formulate StraightMapping with *Constraint Programming* (CP). Our experiments show that CP is an efficient approach for finding a feasible schedule in most cases.

b) Minimizing the required number of cores (MinCores): In this problem, we can now put a partition in any core of the multi-core. The goal is to schedule all partitions with the minimum number of cores (Fig. 2 (b)). We also formulate a CP for this problem. However, since each partition has freedom to go to any core, the search space becomes substantially larger, and the CP is no longer efficient. When solo-partitions are unit-sized, we propose a scalable approximation algorithm that outperforms the CP in the solution quality and, particularly, in scalability. Recall that even unit-sized solo-partitions make

This work is supported in part by Rockwell Collins under RPS#6 45038, by Lockheed Martin under 2009-00524, by NSF under A17321, by ONR under N00014-12-1-0046, and by NSF under CCF-1016684. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors.

978-3-9815370-0-0/DATE13/©2013 EDAA

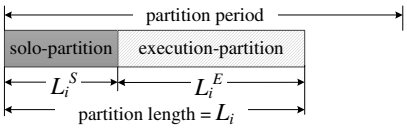

Fig. 1. The structure of an IMA partition with a solo-partition.

the problem non-trivial. The assumption of solo-partitions being unit-sized is justified by the fact that, in practice, solo-partitions are considerably smaller than execution-partitions. Our algorithm is guaranteed to yield a feasible schedule if there exists a feasible one with the solo-partitions. It builds on several algorithmic ideas from the problem of packing periodic tasks into periodic idle intervals and the Bin Packing problem [3], [4].

A. Related Work

The Pinwheel problem [5] is a special class of hard real-time scheduling that guarantees the occurrence of each symbol (task) within any sequence of a given length of consecutive intervals. In [6], Han *et al.* proposed a similar task model called a distance-constrained task system in which the distance between the *finishing times* of consecutive executions are upper-bounded by a threshold. These two classes of models, however, differ from our partition scheduling model in that the distance between the *starting times* of two consecutive executions of an IMA partition is exactly the same as its period due to the nature of cyclic and nonpreemptive scheduling, which is called *strictly periodic*.

A more closely related work to our scheduling problem is [4], [7]; Korst *et al.* addressed the problem of scheduling a set of strictly periodic, nonpreemptive tasks on the minimum number of processors. We adopt the mathematical property of this model and extend it to our multi-IMA partition scheduling with exclusive regions. [8] considered a similar problem of finding a schedule of strictly periodic tasks, maximizing the relative distances between them. The authors proposed a Game Theory based algorithm for uniprocessor scheduling and extended it to a multiprocessor case. This work was extended in [9] to support harmonic and near-harmonic periods in IMA-based architectures. In [10], [11], Lee *et al.* considered an IMA system in which multiple processors are connected over Avionics Full-Duplex Switched Ethernet (AFDX). The authors addressed the problem of generating a cyclic schedule for both partitions and bus channels, guaranteeing the timing requirements of tasks and messages. Recently, Tămaş–Selicean *et al.* [12] proposed an optimization problem of mixed-criticality, cost-constrained partitioned resources in a distributed architecture. The authors developed a Tabu-search algorithm that finds the assignments of tasks and partitions as well as their schedules to minimize the development cost. [4], [7] address minimization of the required number of cores, but do not consider execution exclusiveness. The other works do not consider either of these issues.

Lastly, [2] described a special-purpose partition called a *device management partition* for processors that communicate over a shared bus. The problem of deriving a global schedule that excludes simultaneous executions of device management partitions motivated our work.

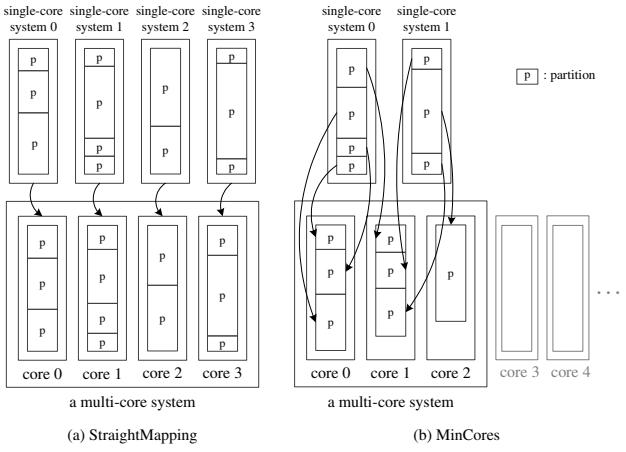

Fig. 2. Overview of migrating concept of StraightMapping and MinCores.

TABLE I
LIST OF SYMBOLS.

Parameter	Description	Parameter	Description
$\mathbf{C}, \mathbf{\Pi}$	set of cores, partitions	T_i	period of π_i
$N^{\mathbf{C}}, N^{\mathbf{\Pi}}$	number of cores, partitions	ψ_i	offset of π_i
C_i, π_i	core i, partition i	L_i	length of π_i
π_i^S, π_i^E	solo-/execution-partition of π_i	L_i^S, L_i^E	length of π_i^S and π_i^E
$C(\pi_i)$	core where π_i is in	MC	major cycle

II. System Overview

A. System Model

We consider a set of $N^{\mathbf{\Pi}}$ partitions denoted by $\mathbf{\Pi} = \{\pi_0, \pi_1, \cdots, \pi_{N^{\mathbf{\Pi}}-1}\}$. Each partition i is represented by $\pi_i = (T_i, L_i^S, L_i^E, \psi_i)$; the first execution of π_i occurs at offset slot ψ_i ($0 \leq \psi_i < T_i$) and then executes with a period T_i for a length L_i [1], which is the sum of solo-partition length, L_i^S, and execution-partition length, L_i^E. We denote each partition type as π_i^S and π_i^E, respectively. Accordingly, π_i executes during the time interval $[\psi_i + kT_i, \psi_i + kT_i + L_i)$ for all $k \in \mathbb{Z}^* \leq (\frac{MC}{T_i} - 1)$. No two solo-partitions in the system are allowed to execute concurrently. However, there is no such constraint on execution-partitions.

We then consider a multi-core system consisting of $N^{\mathbf{C}}$ homogeneous cores, $\mathbf{C} = \{C_0, C_1, \cdots, C_{N^{\mathbf{C}}-1}\}$, and a set of $N^{\mathbf{C}}$ single-core systems, each of which is denoted as C_i^s, consisting of a set of partitions. In StraightMapping, the partitions in C_i^s are to be migrated to C_i as they were in each single-core system. In the MinCores problem, $N^{\mathbf{C}}$ is not a fixed number. We denote the minimum required number of cores for $\mathbf{\Pi}$ found by our approximation algorithm in Sec. IV as $A(\mathbf{\Pi})$ and the number of cores of the optimal solution as OPT($\mathbf{\Pi}$).

Throughout this paper, partition periods are assumed to be harmonic.[2] In the *MinCores* problem, we also assume each solo-partition has a unit length, i.e., $L_i^S = 1$. Although these seem to be restrictive assumptions, we will see that the optimization problem is still difficult to solve with such constraints.

B. Problem Description

StraightMapping

Given $\mathbf{C}$ and $\mathbf{\Pi}$, we try to find a set of feasible partition offsets, $\{\psi_i | i = 0, \cdots, N^{\mathbf{\Pi}} - 1\}$, satisfying the following constraints:

$$\forall_i, C_i^s = C(\pi_i), \forall_{i,j,i\neq j}, k \in \mathbf{Z}^* \leq (\tfrac{MC}{T_i} - 1), \text{and } l \in \mathbf{Z}^* \leq (\tfrac{MC}{T_j} - 1),$$

$$\textit{Local Feasibility}\text{: if } C(\pi_i) = C(\pi_j),$$
$$[\psi_i + kT_i, \psi_i + kT_i + L_i) \cap [\psi_j + lT_j, \psi_j + lT_j + L_j) = \varnothing. \quad (1)$$

[1] A migration of a partition would require an adjustment of partition length due to the clock speed difference between single-core and multi-core system. This is beyond the scope of our paper, however interested readers can refer to the literatures on hierarchical scheduling [13]–[15].

[2] One can convert non-harmonic periods to harmonic ones as in [6], [10], [11].

Global Feasibility: if $C(\pi_i) \neq C(\pi_j)$,
$[\psi_i + kT_i, \psi_i + kT_i + L_i^S) \cap [\psi_j + lT_j, \psi_j + lT_j + L_j^S) = \varnothing$. (2)

MinCores

Given $\mathbf{\Pi}$, the *MinCores* problem finds a set of feasible partition offsets, $\{\psi_i\}$, and their core assignments satisfying both (1) and (2), while minimizing the required number of cores, thus,

$$\text{Minimize } A(\mathbf{\Pi}). \quad (3)$$

III. Straight Migration from Single-Core Systems to a Multi-Core System

In this section, we explain how to generate schedules of the given IMA partitions migrated straight from one single-core system to one core on a multi-core system. Scheduling partitions is ultimately about determining the offsets of partitions, i.e., $\{\psi_i\}$, satisfying Constraints (1) and (2). In [4], the authors expressed the constraint that there can be no overlap between two strictly periodic tasks if the following condition is satisfied:

$$L_i \leq (\psi_j - \psi_i) \bmod g_{i,j} \leq g_{i,j} - L_j,$$

where L_i and L_j are the execution times, and $g_{i,j}$ represents the greatest common divisor of the periods T_i and T_j. We apply this condition to our partition scheduling problem; any two partitions on the same core and any two solo-partitions on different cores must satisfy the condition above. This problem is a decision problem to find a set of feasible partition offsets satisfying the two constraints. We thus formulate it with Constraint Programming.

A. Proof of NP–Completeness

Prior to the CP formulation, we first prove that the Multi-IMA partition scheduling problem with exclusive solo-partitions described in Sec. II-B is NP-complete in the strong sense even if we are given only *two* cores and all solo-partitions have length of at most *one*. It is known that the problem of testing if a set of partitions is schedulable on a single core is strongly NP-complete [4], even without considering solo-partitions. The reader may wonder if their proof immediately implies a similar hardness result for our problem. One subtle difference is that in our setting, we may already know a feasible schedule for each core in which all partitions on the core are feasibly scheduled without considering the exclusiveness across cores of solo-partitions. Our proof is stronger in that we use very restricted solo-partition lengths; the difficulty of our problem stems mainly from solo-partitions – without solo-partitions, the problem becomes trivial.

Theorem 1. *The problem of Multi-IMA partition scheduling with exclusive solo-partitions which optimally finds the partition schedule to make (i) each π_i schedulable according to its L_i^S, L_i^E, and T_i in each local core $C(\pi_i)$, i.e. no overlap with π_j for all $j \neq i$ such that $C(\pi_i) = C(\pi_j)$, (ii) all π_i^S exclusively synchronized throughout all cores, i.e., no overlap with π_j^S for all $j \neq i$, is strongly NP–hard. Further, it remains the case even if all solo-partitions have a length of at most one, and for each core we are given a schedule where all and only the partitions assigned to the core are feasibly scheduled.*

Due to the space constraints we omit the proof. The proof can be found in [16].

B. Constraint Programming (CP) Formulation

We present the CP formulation for the multi-IMA partition schedule problem of StraightMapping described in Sec. II-B.

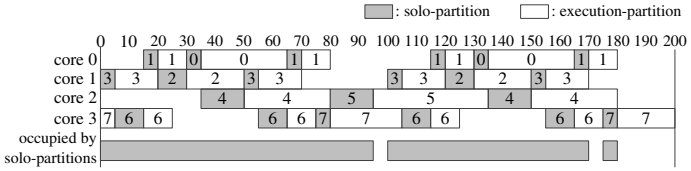

Fig. 3. An example partition schedule with exclusive solo-partitions.

1) Decision variables: ψ_i, offset of π_i, $0 \leq \psi_i \leq (T_i - 1)$.

2) Constraint 1 – Local feasibility (1)*:*

$$\forall\pi_i \forall\pi_j (i \neq j \text{ and } C(\pi_i) = C(\pi_j)), (\psi_j - \psi_i) \bmod g_{i,j} \geq L_i,$$
$$(\psi_j - \psi_i) \bmod g_{i,j} \leq g_{i,j} - L_j.$$

3) Constraint 2 – Global feasibility (2)*:*

$$\forall\pi_i \forall\pi_j (i \neq j \text{ and } C(\pi_i) \neq C(\pi_j)), (\psi_j - \psi_i) \bmod g_{i,j} \geq L_i^S,$$
$$(\psi_j - \psi_i) \bmod g_{i,j} \leq g_{i,j} - L_j^S.$$

We linearize the constraints by replacing $\big((\psi_j - \psi_i) \bmod g_{i,j}\big)$ with $\big((\psi_j - \psi_i) - g_{i,j} \cdot X_{i,j}\big)$, where $X_{i,j}$ is a new real-valued variable bounded in $[\lfloor \frac{1-T_i}{g_{i,j}} \rfloor - 1, \lfloor \frac{T_j - 1}{g_{i,j}} \rfloor + 1]$ if $T_i, T_j \geq 2$.

Example: Fig. 3 shows a schedule generated by CP for an example set of four cores each of which has two partitions. Note that, although no core is fully utilized, any increase in L_i^S of any partition makes the schedule infeasible, which motivates the MinCores problem described in the following section.

IV. Minimization of The Required Number of Cores

As seen in the previous section, some input sets cannot be feasibly scheduled on the pre-assigned cores because of, e.g., the lower clock speed of each core on a multi-core processor than a single-core one or simply due to the exclusive regions. Thus, we address the problem of finding the minimum number of cores that can schedule a given set of partitions. We extend the CP formulation (Sec. III-B) and present an approximation algorithm that can be scalable with respect to the partition count.

A. Extended CP Formulation

We add the following variables and constraints to the CP formulation presented in Sec. III-B.

1) Added decision variables:

- $\lambda_{i,k} = 1$ if π_i is assigned to C_k, otherwise 0.
- $\mu_k = 1$ if any π_i is assigned to C_k, otherwise 0.

2) Constraint 3 - Partition assignment: A partition must be assigned to a core.

$$\forall_i, \quad \sum_k \lambda_{i,k} = 1.$$

3) Constraint 4 - Core occupancy: If any partitions are assigned to core C_k, this indicates the core is being used ($\mu_k = 1$).

$$\forall_k, \quad \textbf{if} \quad \sum_i \lambda_{i,k} \geq 1, \quad \textbf{then} \quad \mu_k = 1.$$

4) Modified Constraint 1 and 2: Since partitions can be assigned to arbitrary cores, the condition $C(\pi_i) = C(\pi_j)$ of Constraint 1 in Sec. III-B needs to be modified to $\lambda_{i,k} = \lambda_{j,k}$. For Constraint 2, we can remove the condition $C(\pi_i) \neq C(\pi_j)$.

5) Objective function: The optimization objective is to minimize the sum of μ_k, i.e., the total number of used cores.

$$\text{Minimize} \quad \sum_k \mu_k.$$

The presented CP, however, is not scalable with respect to the number of partitions – we need as many as $\frac{N^{\mathbf{\Pi}} \cdot (N^{\mathbf{\Pi}} - 1)}{2} \cdot N^k$

Algorithm 1 $A(\mathbf{\Pi})$

1: $S^{\mathbf{\Pi}}$: The sorted list of $\mathbf{\Pi}$ in decreasing order of $\frac{L_i}{T_i}$
2: $\pi_{\sigma(i)}$: i^{th} partition in $S^{\mathbf{\Pi}}$
3: $\mathbf{\Delta}$: The set of generated periodic intervals
4: $\mathbf{C}$: The set of generated cores
5: $\mathbf{C} = \mathbf{C} \cup \{C_0\}$
6: **for all** $\pi_{\sigma(i)}$, $i = 0, \cdots, N^{\mathbf{\Pi}} - 1$ **do**
7: **for all** C_j, $j = 0, \cdots, |\mathbf{C}| - 1$ **do**
8: $\psi \leftarrow$ FINDOFFSET$(C_j, \pi_{\sigma(i)}, \mathbf{\Delta}, S^{\mathbf{\Pi}})$
9: **if** $\psi \geq 0$ **then**
10: Assign $\pi_{\sigma(i)}$ on C_j at offset ψ {$\mathbf{\Delta}$ was updated in FINDOFFSET}
11: Break the loop
12: **end if**
13: **end for**
14: **if** $\pi_{\sigma(i)}$ is not assigned **then**
15: Create a new core C_j.
16: $\mathbf{C} = \mathbf{C} \cup \{C_j\}$
17: Do Line 8–12
18: **end if**
19: **end for**
20: **return** $|\mathbf{C}|$

of Constraint 1, where N^k is the maximum possible number of cores. Accordingly, the problem size grows exponentially.

B. Approximation Algorithm

The key idea of the approximation algorithm for MinCores is to pack partitions into the smallest number of cores possible while guaranteeing that the remaining partitions can be schedulable by tracking the availability of free spaces for solo-partitions.

Considering π_i in decreasing order of $\frac{L_i}{T_i}$ is inspired by the well-known heuristic called the *First Fit Decreasing* (FFD) for the Bin Packing problem. The input of the Bin Packing problem consists of n items of respective sizes $a_1, a_2, \cdots, a_n \in [0, 1]$. The problem to pack all items into the minimum number of bins, each having a unit size, is known to be NP-complete. The FFD algorithm is known to use at most $11/9\text{OPT}(\mathbf{\Pi}) + 1$ bins, where OPT($\mathbf{\Pi}$) denotes the number of bins of the optimal solution [3]. The FFD algorithm sorts all items in decreasing order according to their sizes. And then, for each item in consideration, it attempts to add it to each opened bin favoring earlier opened bins. If unsuccessful, it opens another bin and places the item into that bin. Packing items in an arbitrary order results in a worse guarantee in that it uses at most 2OPT($\mathbf{\Pi}$) bins.

Since partitions are nonpreemptive, the utilization $\frac{L_i}{T_i}$ of π_i is not sufficient for determining the feasibility of a set of partitions. There are examples of partitions that cannot be scheduled together although the sum of utilization is very small [5], [6]. However, utilization could still be a useful metric for designing a heuristic. Thus, by viewing each partition as an item in the Bin Packing problem and its utilization as the item size, the FFD suggests the ordering in Line 1,6, and 7 of Algorithm 1.

There can, however, be conflicts between solo-partitions if partition offsets are not carefully chosen. The importance of selecting a proper offset can be seen if partitions have arbitrary offsets. In such a case, the remaining partitions might not be able to be assigned even if an infinite number of cores were available. Thus, in our algorithm, when trying to assign a new partition to a core, we find a proper offset that can ensure that all the remaining partitions will be guaranteed to be assigned (Line 8 in Algorithm 1, which is detailed in Algorithm 2). In order to do so, we maintain guaranteed slots for the remaining solo-partitions by a set of *periodic intervals* [4]. A periodic interval (PI), δ_j,

Algorithm 2 FINDOFFSET $(C_j, \pi_i, \mathbf{\Delta}, S^{\mathbf{\Pi}})$

1: S^{Δ}: Sorted list of Δ in decreasing order of T^{δ}
2: **for all** δ_k in S^{Δ} **do**
3: **if** CANFIT(π_i, δ_k) **then**
4: **for all** $\psi \in [0, T_i - 1]$ s.t. $\psi = \psi_k^{\delta} \pmod{T_k^{\delta}}$ **do**
5: **if** ISLOCALLYFEASIBLE(π_i, ψ, C_j) **then**
6: $\mathbf{\Delta}^* \leftarrow \mathbf{\Delta}/\{\delta_k\} \cup$ UPDATEPI(π_i, ψ, δ_k)
7: **if** CANGUARANTEE$(\mathbf{\Delta}^*, S^{\mathbf{\Pi}}, i)$ **then**
8: $\mathbf{\Delta} \leftarrow \mathbf{\Delta}^*$
9: **return** ψ
10: **end if**
11: **end if**
12: **end for**
13: **end if**
14: **end for**
15: **return** -1 {No offset has been found.}

can be represented by $(T_j^{\delta}, L_j^{\delta})$ meaning that consecutive free slots of L_j^{δ} appear periodically at every T_j^{δ} slots in the schedule. For example, if only one partition with $T_i = 10$ and $L_i = 2$ has been scheduled, we have one periodic interval with $T_j^{\delta} = 10$ and $L_j^{\delta} = 8$. π_i can be assigned to δ_j if and only if [4]

$$L_i + (T_j^{\delta} - L_j^{\delta}) \leq gcd(T_i, T_j^{\delta}) \quad \text{or} \quad L_j^{\delta} = T_j^{\delta}. \tag{4}$$

A periodic interval, however, can only tell us of its existence, not where it is actually located. Thus, in order to represent the available slots precisely, we extend the expression of each periodic interval to a three-tuple, $(T_i^{\delta}, L_i^{\delta}, \psi_i^{\delta})$ in which a free space of length L_i^{δ} begins at ψ_i^{δ} and appears at every T_i^{δ}.

The algorithm keeps track of empty slots that can be used for solo-partitions, as a compact set of PIs of a *unit length*. Let $a_1, a_2, \cdots, a_l$ be the set of periods of all partitions in increasing order. Initially the empty slots are expressed as a_1 PIs: $(a_1, 1, h)$, $0 \leq h < a_1$. When π_i^S is scheduled in δ_j at ψ_i, δ_j splits into multiple periodic intervals (Algorithm 3 & Fig. 4). The key idea is to maintain the set Δ of current PIs to be minimal. We say that a Δ is *minimal* if all PIs in Δ have a unit length and no subset of Δ can be replaced with a PI with a smaller period. For example, suppose all periods are powers of two. Then $\Delta = \{(4, 1, 0), (4, 1, 1), (4, 1, 2)\}$ is not minimal, since the PIs $(4, 1, 0)$ and $(4, 1, 2)$ are equivalent to a PI $(2, 1, 0)$. In contrast, $\Delta = \{(4, 1, 1), (4, 1, 2)\}$ is minimal. It is easy to see that the minimal expression of PIs is unique.

We now explain in detail how to find the offset ψ_i of π_i on core C_j (Algorithm 2). We try each periodic interval that has been created up until that point in decreasing order of periods. Thus, the periodic interval set, Δ, is sorted in decreasing order of T_i^{δ}. Once Δ is sorted, we pick each δ_k and check if π_i^S can fit into the interval by using Eq. (4) with $L_i = L_i^S$. By definition of minimal PIs, it fits if and only if $T_i \geq T_k^{\delta}$, because periods are harmonic. Note that π_i^S must have an offset $\psi_i \in [0, T_i)$ to fit in δ_k such that $\psi = \psi_k^{\delta} \pmod{T_k^{\delta}}$. However, it could not be available to π_i since even if π_i^S can be allocated at ψ, π_i could be in conflict with another partition π_ℓ in the same core (recall that δ_k is the periodic interval for solo-partitions in our algorithm). Thus, we check if π_i and all partitions already assigned to C_j satisfy Eq. (1), assuming π_i has an offset ψ (Line 5). If not, we try another offset ψ. If all ψ fail in Line 4, we move to the next PI in the sorted list δ, S^{Δ}. Now, consider a case in which π_i^S can fit in a $\delta_k \in \Delta$ and further placing π_i at offset ψ makes no conflict with other partitions already assigned to C_j. The final ascertaining of ψ is whether or not all the remaining partitions

Algorithm 3 UPDATEPI (π_i, ψ, δ_j)

```
1: Let T_i be a_p and T_j^δ be a_h
2: Δ ← ∅
3: if (T_i == T_j^δ) & (ψ == ψ_j^δ) then
4:     return Δ
5: else if T_i > T_j^δ then
6:     for m = h to p − 1 do
7:         for all ψ s.t. ψ = ψ_j^δ (mod a_m) and ψ ≠ ψ_j^δ (mod a_{m+1}) do
8:             Δ = Δ ∪ (a_m, 1, ψ).
9:         end for
10:        { (a_{m+1}/a_m) − 1 periodic intervals are created.}
11:    end for
12: end if
13: return Δ
```

yet to be assigned are guaranteed to find their offsets after π_i^S is placed in δ_k with offset ψ.

Feasibility test: To describe how to implement CANGUARANTEE (Line 7 in Algorithm 2), it suffices to show how to examine if a set S of solo-partitions can be scheduled into a minimal set E of PIs of a unit length. There always exists a feasible schedule for a set of strictly periodic tasks whose periods are harmonic, lengths are unit-sized, and total utilization is less than or equal to 1, provided that they are sorted in increasing order of periods [5], [6]. We however need a stronger test that also works when all time slots are not available. To the best of our knowledge, our feasibility test seems to appear only implicitly in previous work.

To formally define our feasibility test, recall that $a_1, a_2, \cdots, a_l$ are the periods in increasing order. We say that $v(S) = (s_1, s_2, \cdots, s_l)$ is the canonical vector of S when s_i is the number of solo-partitions in S whose period is exactly a_i. Similarly, we say that $v(E) = (e_1, e_2, \cdots, e_l)$ is the canonical vector of set E. Let l' be the smallest integer in $\{1, 2, \cdots, l\}$ such that $s_{l'} \neq 0$ or $e_{l'} \neq 0$. We say that $v(E) \succeq v(S)$ if $l' = l$ and $e_{l'} \geq s_{l'}$, or if $l' < l$, $e_{l'} \geq s_{l'}$ and $(0, 0, \cdots, 0, \frac{a_{l'+1}}{a_{l'}}(e_{l'} - s_{l'}) + e_{l'+1}, e_{l'+2}, \cdots, e_l) \succeq (0, 0, \cdots, 0, s_{l'+1}, s_{l'+2}, \cdots, s_l)$. Note that $\succeq$ is recursively defined.

Theorem 2. *Consider any set S of solo-partitions and any minimal set E of periodic intervals of a unit length. Then the solo-partitions in S can be scheduled into the periodic intervals in E if and only if $v(E) \succeq v(S)$.*

Proof Sketch. Let $1 \leq l' \leq l$ be the smallest integer such that $s_{l'} \neq 0$ or $e_{l'} \neq 0$. We prove the theorem using induction on the value of l' in decreasing order. In the initial case that $l' = l$, the theorem trivially holds. Suppose the theorem is true for all $l' > h$. We show that the theorem holds also when $l' = h$.

($\Rightarrow$) A partition π_1 cannot be scheduled in a unit length PI of larger period. Hence it must use one of the PIs of the same period (no PIs of smaller periods are available by definition of $l' = h$). Clearly each (solo-)partition π_i of period a_h in S uses one PI of the same period. Hence it follows that $e_h \geq s_h$. Now after serving s_h partitions of period a_h, the remaining PIs of period a_h can be turned into $\frac{a_{h+1}}{a_h}(e_h - s_h)$ PIs of period a_{h+1}. The induction hypothesis completes the proof of this direction.

($\Leftarrow$) The proof is similar to the one above and is omitted. □

Example: Fig. 4 shows the schedule for an example set of five partitions generated by the presented algorithm. Note that π_3 takes slot 5 on C_0 by updating δ_1; utilizing δ_4 would require an additional core since it cannot fit into any core with slot 12.

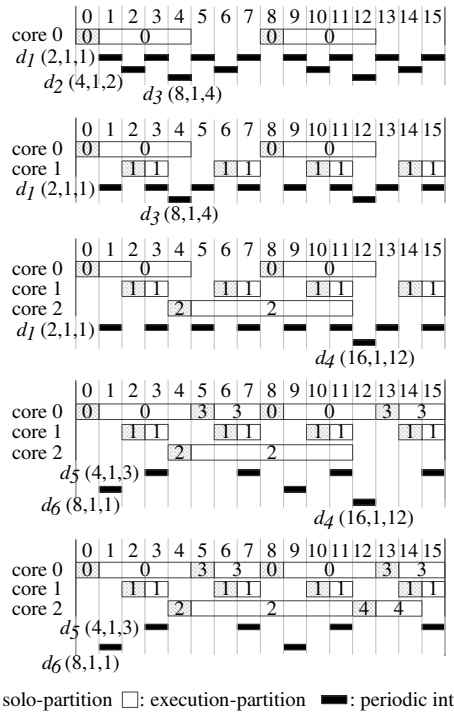

Fig. 4. An example of partition assignment and scheduling for five partitions generated by the proposed approximation algorithm.

V. EVALUATION

In this section[3], we show the impact of solo-partition on the feasibility of Multi-IMA partitions through the result of StraightMapping with CP. Then, we evaluate the proposed approximation algorithm of MinCores with varying partition packing order and offset searching. In the evaluation of MinCores, we do not compare the solving time of CP with that of our approximation algorithm since those are incomparable. The algorithm runs in milliseconds, while CP runs in tens of minutes.

Firstly, Fig. 5 shows the result of StraightMapping obtained by the CP formulation. The experimental parameters are as follows: 4 cores; 3–5 partitions per core; execution-partition length is in {1,2,$\cdots$,30}; period is in {64,128,256,512}; 40 random input sets for each solo-partition length group. For each input set, CP outputs one of the following: a) *solution found*: a feasible schedule for a given set of partitions pre-assigned on each core is found; b) *timeout*: we put a 30–minute time limit on the execution of each input set. We observed that when there exists a feasible solution, the solving time was a few minutes and no more than 10 minutes. Thus, when a CP execution terminates by timeout, we suspect that no solution for the given input set exists; c) *infeasible input set*: the input set itself cannot satisfy some constraints. The

[3]We executed all the experiments on Intel(R) Core(TM)2 Duo CPU 2.66 GHz with 4GB RAM. For CP, we used IBM ILOG CPLEX CP Optimizer.

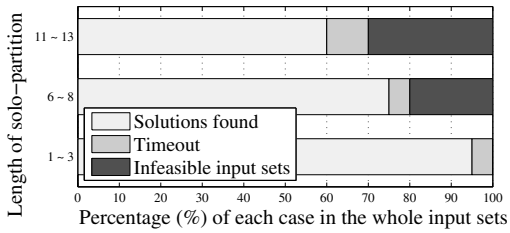

Fig. 5. Result of *StraightMapping* obtained by CP in Sec. III-B.

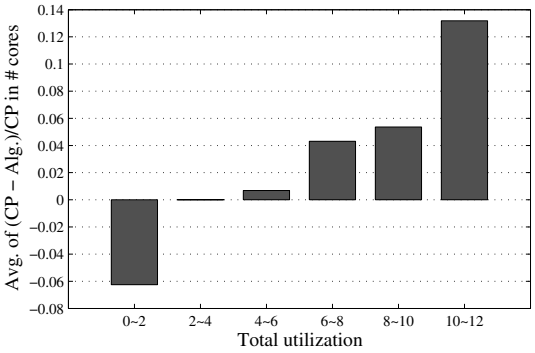

Fig. 6. Average relative differences between the required number of cores found by our algorithm and the solution of CP with various total utilizations.

result shows that, in the stable conditions, except for the length of solo-partitions, the proportion of infeasible sets increases only as the solo-partition lengths increase. That is having solo-partitions is the bottleneck affecting the schedulability of the whole system.

Next, we evaluated our approximation algorithm in Sec. IV-B by comparing the minimum number of cores it can find for a given set with what CP finds. For this evaluation, we generated 200 random input sets, each of which consists of 5–50 partitions. The length of execution-partition and the period were chosen from $\{5,6,\cdots,50\}$ and $\{64,128,256,512\}$, respectively. Also, the time limit of CP was set to 30 minutes. In the case of a timeout, the solution of CP is the best solution found up until the time limit. In Fig. 6, the y–axis shows the average relative differences between the required number of cores found by our algorithm and the solution of CP. We grouped input sets according to the total utilizations, i.e., the sum of $\frac{L_i}{T_i}$ over partitions. A higher utilization is mainly attributed to the number of partitions in each set. Thus, as the total utilization increases, CP could not find any better solution than the one our approximation algorithm could find because of the exponentially increasing search space. This shows that our algorithm is scalable.

Lastly, in Fig. 7, we evaluated our approximation algorithm by adding randomness in order to justify the theoretical effectiveness. We compared the original algorithm described in Sec. IV-B and the following three variations: a) *random sorting*: partitions are packed into cores in a random order. b) *random offset*: when a partition π_i finds its offset on a core, it randomly tries offsets within $[0, T_i)$ until it either finds a feasible one or fails T_i times. c) *random sorting and offset*: both of the above combined. For this evaluation, we used the same experimental parameters as in Fig. 6. For each variation, we take the average of 100 executions for an input. Each x-axis shows the number of partitions, and each y-axis is the relative differences between the solutions found by our algorithm and each variation. The results show that the added randomness increased the required number of cores. This is worse in (c); in the worst-case, it used 50% more cores than our algorithm, and could not output the same solution for $N^{\mathbf{\Pi}} > 20$. Although there are some cases where the variations found better solutions than our algorithm, these results show the theoretical importance of the proposed algorithm.

VI. Conclusion

In this paper, we developed an optimized Multi-IMA partition scheduling algorithm considering exclusive regions, *solo-partitions*, to synchronize partitions across cores on a multi-core system. We showed that the problem of finding an optimal

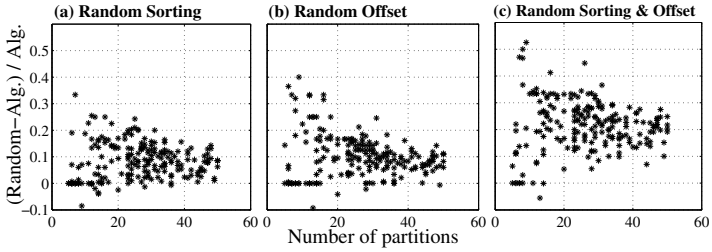

Fig. 7. Relative differences between our algorithm and three random variations in partition packing order and offset searching.

Multi-IMA partition schedule for StraightMapping problem is NP-complete and presented a CP formulation. In addition, we relaxed the problem to find the minimum number of cores needed to schedule a given set of partitions. The proposed approximation algorithm is guaranteed to find a feasible schedule if there exists a feasible schedule of solo-partitions. To the best of our knowledge, this is the first work for the problem of scheduling multi-IMA partitions with exclusive regions on a multi-core system. An interesting direction for future work would be to extend the problem to non-uniform sized solo-partitions. This would help solo-partitions find wider application.

References

[1] *ARINC Specification 651: Design Guidance for Integrated Modular Avionics*, ser. ARINC report. Airlines Electronic Engineering Committee (AEEC) and Aeronautical Radio Inc, Nov. 1991.

[2] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," *NASA Langley Technical Report Server*, Mar. 1999.

[3] M. Yue, "A simple proof of the inequality FFD(L) $\leq$ 11/9 OPT(L) + 1, $\forall$L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica (English Series)*, vol. 7, no. 4, pp. 32–331, 1991.

[4] J. H. M. Korst, E. H. L. Aarts, and J. K. Lenstra, "Scheduling periodic tasks," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 428–435, 1996.

[5] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: a real-time scheduling problem," in *Proc. of the 22nd Annual Hawaii International Conference on System Sciences*, 1989, pp. 693–702.

[6] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Transactions on Computers*, vol. 45, pp. 814–826, 1996.

[7] J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels, "Periodic assignment and graph colouring," *Discrete Appl. Math.*, vol. 51, pp. 291–305, Jul. 1994.

[8] A. Al Sheikh, O. Brun, P.-E. Hladik, and B. J. Prabhu, "A best-response algorithm for multiprocessor periodic scheduling," in *Proc. of the 23rd Euromicro Conference on Real-Time Systems*, 2011, pp. 228–237.

[9] ——, "Strictly periodic scheduling in IMA-based architectures," *Real-Time Syst.*, vol. 48, no. 4, pp. 359–386, Jul. 2012.

[10] Y.-H. Lee, D. Kim, M. Younis, J. Zhou, and J. Mcelroy, "Resource scheduling in dependable integrated modular avionics," in *Proc. of the International Conference on Dependable Systems and Networks (FTCS-30 and DCCA-8)*, 2000, pp. 14–23.

[11] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Proc. of the 19th Digital Avionics Systems Conference (DASC)*, 2000.

[12] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures," in *Proc. of the 32nd IEEE Real-Time Systems Symposium*, 2011, pp. 24–33.

[13] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proc. of the 4th ACM international conference on Embedded software*, 2004, pp. 95–103.

[14] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proc. of the 15th Euromicro Conference on Real-Time Systems*, 2003, pp. 151–158.

[15] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proc. of the 24th IEEE Real-Time Systems Symposium*, 2005, pp. 389–398.

[16] J.-E. Kim, M.-K. Yoon, S. Im, R. Bradford, and L. Sha, "Multi-IMA Partition Scheduling with Synchronized Solo-Partitions for Multi-Core Avionics Systems," Dept. of Computer Science, University of Illinois at Urbana-Champaign, Technical report, May 2012, https://www.ideals.illinois.edu/handle/2142/35280.