Multi-IMA Partition Scheduling with Synchronized Solo-Partitions for Multi-Core Avionics 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. Although multi-core systems are receiving wide attention from the avionics industry, 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. In connection with meeting this challenge, several issues arise such as speed differences between single-core systems and multi-core systems and synchronization for exclusive I/O transactions. To address these challenges, we have developed an optimized Multi-IMA partition scheduling algorithm which considers exclusive regions to achieve the synchronization between partitions across cores. We show that the problem of finding the optimal Multi-IMA 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 supports the independent development of the various real-time avionics functions (having various safety-assurance levels) and enables them to run within partitions that are temporally and spatially isolated from one another. This partitioning mechanism has helped ease the certification process for avionics systems.

Over the past decade, microchip designers found it increasingly difficult to dissipate the waste heat that would be generated if more and more transistors were squeezed into each unit of area of silicon; this challenge has led to a leveling off of peak clock speeds [3]. As a result, designers have assembled devices into multiple microprocessor cores as a way of achieving continued increases in computational power. Within the avionics industry, multi-core systems are attractive because they offer the potential for more easily achieving size, weight, and power (SWaP) requirements.

Migrating avionics systems from single-core systems to a multi-core system, however, is not a trivial task. The fundamental challenge to integrating an existing set of pre-certified single-core avionics IMA systems into a multi-IMA multi-core system is to ensure that the temporal and spatial isolation of the partitions will be maintained (and to achieve this assurance without incurring huge recertification costs).

Several issues arise in connection with meeting this fundamental challenge. One such issue is the *difference in speed* between a single-core processor and a multi-core processor; in general, one core of a multi-core system has lower clock speed than that of a single-core system due to temperature and power consumption constraints. Accordingly, a direct mapping, such as a one-to-one mapping, would not work if the speed difference is significant or existing partitions have irregular and inconsistent sizes. Another issue is *synchronization* which arises in the migration to multi-core. Synchronization is required for exclusive transactions such as I/O, and helps achieve *I/O virtualization*. For example, some hardware devices such as graphic cards or storage units do not admit multiple accesses. In such cases, the IMA partition schedules should be synchronized by *exclusive executions* so that no data loss or corruption would occur.

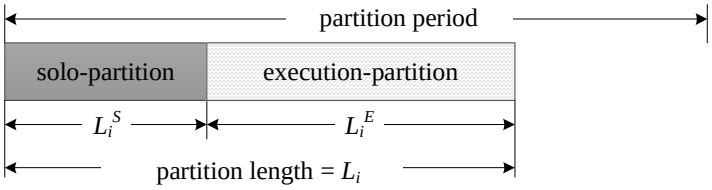


Fig. 1. The structure of an IMA partition with a solo-partition.

In this report, we investigate the problem of scheduling IMA 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. By considering this, we can address obstacles in transitioning from single-core to multi-core. We formalize the problem as a *multi-IMA partition scheduling optimization* problem and address the aforementioned challenges. Our investigation into the problem includes a discussion of the theoretical aspects of the problem, the development of scalable algorithms and the presentation of data from supporting experiments.

In our model, a partition consists of two logical regions – *solo-partition* followed by *execution-partition*, as depicted in Fig. 1. Solo-partition is not a new concept in a single core system and 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 ready in its solo-partition, an execution-partition follows its solo-partition. In the multi-core setting, we require a solo-partition to be an *exclusive region* in order to achieve synchronization between partitions across cores. In other words, only one solo-partition is allowed to be scheduled at a time instant. In contrast, execution-partitions do not need to satisfy the exclusiveness that solo-partitions do. 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 the IMA partition model described above, we formalize the following multi-IMA scheduling optimization problems.

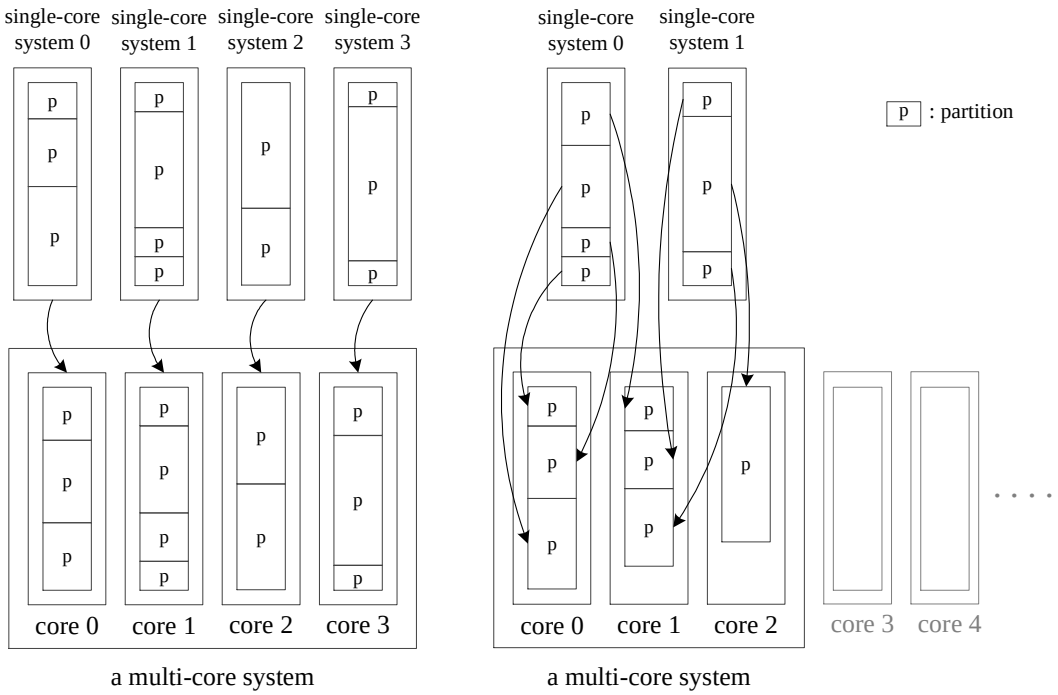
a) Straight Mapping (StraightMapping): Given a set of single-core IMA systems, their partitions, and a multi-core system, we are required to construct a set of local core schedules that satisfy the exclusiveness of solo-partitions using one-to-one mapping from a single-core IMA system to a core of the 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).

One-to-one mapping is the logical first step that one would try in migrating from single-core to multi-core. However, it may be the case that no one-to-one mapping yields a feasible schedule. 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 that finds a feasible straight mapping schedule in most cases.

Since straight mapping is not always possible, and further motivated by our hardness result, we consider the following relaxed problem.

b) Minimization of the required number of cores (MinCores): In this problem, we can now put a partition in any core of the multi-core system. The goal is to schedule all partitions with the minimum number of cores. See Fig. 2 (b).

We also formulate a CP for this problem. However, since each partition has more freedom in that it can go to any core, the search space becomes substantially larger, and the CP is no longer efficient. When solo-partitions have a unit size, 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 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 [4], [5].



(a) StraightMapping (b) MinCores

Fig. 2. Abstracted overview of migrating concept of StraightMapping and MinCores.

A. Organization of the report

The remaining sections are organized as follows: Sec. II describes the system model and then formalizes the multi-IMA scheduling optimization problem. In Sec. III, we explain the details of the StraightMapping problem, showing that it is strongly NP-complete, and present its CP formulation. In Sec. IV, we propose an approximation algorithm for MinCores problem, providing theoretical discussion. The experimental results are given in Sec. V. And then, in Sec. VI, we summarize the related work. Sec. VII concludes this report.

II. System Overview

In this section, we first provide the system model we consider in this report, and then formally describe the scheduling optimization problems of multi-IMA partitions with exclusive regions.

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 of T_i for a length of 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

$$[\psi_i + kT_i, \psi_i + kT_i + L_i), \tag{1}$$

for all $k \in \mathbb{Z}^* \leq (\frac{MC}{T_i} - 1)$. As previously stated, any two solo-partitions on the system are not allowed to execute at the same time. However, there is no such constraint on execution-partitions.

We then consider a multi-core system consisting of $N^{\mathbf{C}}$ homogeneous cores, $\mathbf{C} = \{C_0, C_1, \cdots, C_{N^{\mathbf{C}}-1}\}$. We also consider a set of $N^{\mathbf{C}}$ single-core systems, each of which is denoted as C_i^s and consists of a set of IMA

[1] A migration of a partition would require an adjustment of partition length due to the clock speed difference. This is beyond the scope of our report, however interested readers can refer to the literature on hierarchical scheduling [6]–[8].

TABLE I
LIST OF SYMBOLS.

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

partitions. For the problem of one-to-one migration of a set of single-cores to a multi-core system, the partitions in C_i^s are to be migrated to C_i as they were in each single-core system.

In the problem of finding the minimum required number of cores for a given $\mathbf{\Pi}$, a partition in C_i^s is not necessarily migrated to C_i. Depending on the optimization result, the partition can end up being in C_j where $i \neq j$. However, every partition must be migrated to any one of $C_j \in \mathbf{C}$. In this particular 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 report, partition periods are assumed to be harmonic.[2] Additionally, in *MinCores* problem, we 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, $\{\psi_i | i = 0, \cdots, N^{\mathbf{\Pi}} - 1\}$, that satisfy the following constraints:

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

$$[\psi_i + kT_i, \psi_i + kT_i + L_i) \cap [\psi_j + lT_j, \psi_j + lT_j + L_j) = \varnothing, \text{if } C(\pi_i) = C(\pi_j). \tag{2}$$

$$[\psi_i + kT_i, \psi_i + kT_i + L_i^S) \cap [\psi_j + lT_j, \psi_j + lT_j + L_j^S) = \varnothing, \text{if } C(\pi_i) \neq C(\pi_j). \tag{3}$$

If a feasible schedule does not exist, it must be possible to determine that it is infeasible.

MinCores

Given $\mathbf{\Pi}$, *MinCores* problem finds a set of feasible partition offsets, $\{\psi_i | i = 0, \cdots, N^{\mathbf{\Pi}} - 1\}$, and their core assignments that satisfy both Constraints (2) and (3), while minimizing the required number of cores, thus,

$$\text{Minimize } A(\mathbf{\Pi}). \tag{4}$$

III. STRAIGHT MIGRATION FROM SINGLE-CORE SYSTEMS TO A MULTI-CORE SYSTEMS

In this section, we explain in detail 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., $\{\psi_i\}$, that satisfy Constraints (2) and (3). In [5], the authors expressed the constraint that there can be no overlap between two strictly periodic tasks, which can be applied and transformed to our

[2]When the given periods are non-harmonic, one can convert non-harmonic periods to harmonic ones as in [9]–[11].

multi-IMA partition scheduling problem. First of all, to be *locally feasible*, any two partitions in the same core cannot overlap,

$$\forall \pi_i \forall \pi_j (i \neq j \text{ and } C(\pi_i) = C(\pi_j)), L_i \leq (\psi_j - \psi_i) \bmod g_{i,j} \leq g_{i,j} - L_j, \tag{5}$$

where $g_{i,j}$ represents the greatest common divisor of T_i and T_j. In addition, to be *globally feasible*, no two solo-partitions on different cores can execute at the same time,

$$\forall \pi_i \forall \pi_j (i \neq j \text{ and } C(\pi_i) \neq C(\pi_j)), L_i^S \leq (\psi_j - \psi_i) \bmod g_{i,j} \leq g_{i,j} - L_j^S. \tag{6}$$

Note that this problem is not an optimization problem but rather a decision problem to find a set of feasible partition offsets satisfying the above two constraints. To address this problem, 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 [5], 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. If solo-partitions are allowed to have non-uniform lengths that are greater than one, their proof can be easily adapted to our setting. However, it is necessary that our proof differs from theirs in that we are allowed to use very restricted solo-partition lengths. Note that the difficulty of our problem stems mainly from solo-partitions; without solo-partitions, the problem becomes trivial.

We show its strong NP-completeness via a reduction from the *3-Partition Problem*, which is a well-known strongly NP-complete problem asking whether a given set of integers can be partitioned into the same sum of triples [12]. It would imply that the Multi-IMA partition scheduling problem does not admit an efficient algorithm even if all parameters of the input are small. The 3-Partition Problem is formally defined as follows: the input consists of a set S of $n = 3k$ positive integers and the target sum E of each triple or subset. This problem asks whether S can be partitioned into k $(= n/3)$ disjoint subsets $S_0, S_1, \cdots, S_{k-1}$, each of which includes exactly three integers that add up exactly to E.

Theorem 1. *The problem of Multi-IMA partition scheduling with exclusive solo-partitions which optimally finds the partition schedule to make*

- *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 $C(\pi_i) = C(\pi_j)$,*
- *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.

Proof: Consider any instance for the 3-Partition problem. Let $a_0, a_1, \cdots, a_{3k}$ be the given $3k$ positive integers. For the IMA partition system, we create two cores, C_0 and C_1, and $k(E+5)+1$ partitions. There are four groups of partitions, and each group has its own role. The partitions are described as follows. For notational convenience, we also use the notation π' in addition to π. We also drop the offset from the tuple (only in this proof), i.e., $\pi_i = (T_i, L_i^S, L_i^E)$.

- $\pi'_0 = (2(E+1), 0, E+1)$.
- $\pi'_i = (2k(E+1), 1, 0)$, $1 \leq i \leq k(E+1)$.
- $\pi_i = (2k(E+1), 1, a_i - 1)$, $0 \leq i \leq 3k-1$.
- $\pi_i = (2k(E+1), 1, E+1)$, $3k \leq i \leq 4k-1$.

Core C_0 is given to partitions π'_i, $0 \leq i \leq k(E+1)$, and core C_1 is given to partitions π_i, $0 \leq i \leq 4k-1$. Note that E can be assumed to have a size that is polynomial in the input size since the 3-Partition is strongly NP-hard. Hence, the above instance has a polynomial size. This completes the description of the instance for StraightMapping. Note that there is a trivial, feasible schedule when we consider only the partitions assigned to each core.

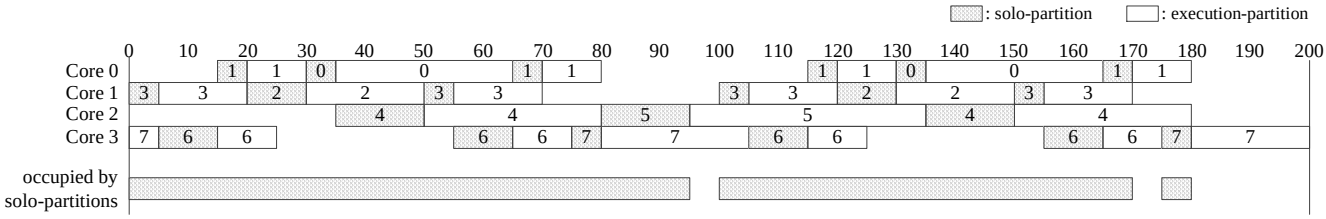


Fig. 3. An example partition schedule with exclusive solo-partitions.

The partitions in C_0 are used to enforce the constraints of the 3-Partition. Partitions π'_i fully utilize C_0 with their solo-partitions and execution-partitions, since the total utilization of π'_i is exactly one. The only feasible schedules for C_0 are to pack the unit-length solo-partitions π'_i, $1 \leq i \leq k(E+1)$ into the empty time slots that are left after scheduling π'_0. Observe that there is one periodic interval of length $(E+1)$ and period $2(E+1)$ that can be used by the solo-partitions in C_1. Alternatively, we can equivalently think that there are k periodic intervals of length $(E+1)$ and period $2k(E+1)$ and they are equally spaced by $E+1$ time slots away. Since all π_i have period $2k(E+1)$, we will focus on the time interval $[0, 2k(E+1))$ and the k intervals of length $E+1$ that are free for solo-partitions. Assume without loss of generality that $I_h = [2h(E+1), 2h(E+1)+E]$, $0 \leq h \leq k-1$ are such intervals.

As is the case in C_0, core C_1 is fully utilized by its partitions π_i. As mentioned above, since all partitions π_i have period $2k(E+1)$, we will focus on the exactly one appearance during $[0, 2k(E+1))$ of solo and execution parts of each partition π_i. Consider any interval I_h. We can, without loss of generality, consider I_0. We claim that one of π_i, $3k \leq i \leq 4k-1$ must be scheduled starting exactly at time E. Note that no partition π_i can start at time $E+1 \leq t \leq 2E+1$, since these time slots are used by the solo-partitions of π'_i. Hence during $[E+1, 2E+1]$, core C_1 can be used only by a partition π_i that starts no later than time E. Note that π_i, $3k \leq i \leq 4k-1$ has a length strictly larger than π_i, $0 \leq i \leq 3k-1$ and that a π_i of total length $E+2$, $3k \leq i \leq 4k-1$ can cover the entire interval $[E+1, 2E+1]$ only when it starts at time E; recall that C_1 cannot be wasted at any time. Therefore we are left with k disjoint intervals of length E that are separated from each other. Now we are forced to pack the remaining partitions π_i of total length a_i, $0 \leq i \leq 3k-1$ into these intervals. Each π_i, $0 \leq i \leq 3k-1$ has a length corresponding uniquely to a number a_i in the given instance for the 3-Partition problem. Hence there exists a feasible schedule if and only if the 3-Partition instance is a Yes-instance.

This completes the proof that our problem is strongly NP-complete, since it is trivial to see that it is in NP. ■

B. Constraint Programming (CP) Formulation

In this section, we present the CP formulation for the multi-IMA multi-core partition schedule problem of StraightMapping described in Sec. II-B.

1) Decision variables:

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

2) Constraint 1 – Local feasibility (5)*:*

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

3) Constraint 2 – Global feasibility (6)*:*

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

We can 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 $T_i, T_j \geq 2$.

C. Example

Fig. 3 shows a schedule generated by CP for an example input set with four cores each of which has two partitions. Note that, although no core is fully utilized in this example, any increase in L^S_i 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, e.g., because of the lower clock speed of each core of a multi-core processor than a single-core one or simply due to the exclusive regions. Thus, in this section, we address the problem of finding the minimum number of cores that can schedule a given set of partitions. For this problem, we first extend the CP formulation in the previous section and present an approximation algorithm that can be scalable with respect to the number of partitions.

A. Extended CP Formulation

The CP formulation for MinCores problem can be extended from the one presented in Sec. III-B by adding the following variables and constraints:

1) Added decision variables:

- $\lambda_{i,k} = \begin{cases} 1 & \text{if } \pi_i \text{ is assigned to } C_k, \\ 0 & \text{otherwise.} \end{cases}$
- $\mu_k = \begin{cases} 1 & \text{if any partition is assigned to } C_k, \\ 0 & \text{otherwise.} \end{cases}$

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 one or more partitions are assigned to core C_k, this indicates that the core is being used, i.e., $\mu_k = 1$.

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

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 simply 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. Accordingly,

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

The presented CP, however, is not scalable with respect to the number of partitions because each partition can be assigned to an arbitrary core. Thus, we need as many as $\frac{N^{\mathbf{\Pi}} \cdot (N^{\mathbf{\Pi}}-1)}{2} \cdot N^k$ of Constraint 1, where N^k is the maximum possible number of cores. Accordingly, the problem size grows exponentially with the number of partitions.

B. Approximation Algorithm

The key idea of the approximation algorithm for MinCores problem presented in this section 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 space for solo-partitions.

Considering partitions π_i in decreasing order of 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 goal is to pack all items into the minimum number of bins, each having a unit size. This problem 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 [4]. The FFD algorithm simply sorts all items in decreasing order according to their size. 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. We note that packing items in an arbitrary order results in a worse guarantee in that it uses at most 2OPT($\mathbf{\Pi}$) bins.

Recall that partitions must be scheduled nonpreemptively. Generally in this case, the utilization L_i/T_i of partitions π_i is not sufficient for determining the feasibility of a set of partitions. There is, in fact, an easy example of two

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 core C_j at offset ψ
11: $\{\mathbf{\Delta}$ was updated in FINDOFFSET$\}$
12: Break the loop
13: **end if**
14: **end for**
15: **if** $\pi_{\sigma(i)}$ is not assigned **then**
16: Create a new core C_j.
17: $\mathbf{C} = \mathbf{C} \cup \{C_j\}$
18: Do Line 8–13
19: **end if**
20: **end for**
21: **return** $|\mathbf{C}|$

partitions that cannot be scheduled together although the sum of utilization is very small [9], [13]. However, utilization could still be a useful metric for designing a heuristic. We use the obvious necessary condition for a set of partitions being schedulable on a single core: the total utilization of all the partitions should not exceed one. 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 choose 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* [5]. A periodic interval (PI), δ_j, can be represented by (T_j^δ, L_j^δ) 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 $L_j^\delta = 8$. Partition π_i can be assigned to δ_j if and only if [5]

$$L_i + (T_j^\delta - L_j^\delta) \leq gcd(T_i, T_j^\delta) \quad \text{or} \quad L_j^\delta = T_j^\delta. \tag{7}$$

A periodic interval, however, can only tell us of its existence, not where it is actually located. Thus, in order to precisely represent the available slots, we extend 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 the empty time 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 solo-partition is scheduled in δ_j with offset ψ_i, δ_j splits into multiple periodic intervals based on the rules in Algorithm 3, which are also illustrated in Fig. 4; finding the offset ψ_i will be discussed soon. 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.

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.}

We now explain in detail how to find the offset ψ_i of partition π_i on core C_j as described in 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. (7) with $L_i = L_i^S$. In fact this is easy: 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. (5), assuming π_i has an offset of 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, let us 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 yet to be assigned are guaranteed to find their offsets after π_i^S is placed in δ_k with offset ψ. In the following, we explain how to implement the function CANGUARANTEE in Line 7.

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. It should be noted that 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

Algorithm 3 UPDATEPI (π_i, ψ, δ_j)

1: Let T_i be a_p and T_j^{δ} be a_h
2: $\Delta \leftarrow \varnothing$
3: **if** $(T_i == T_j^{\delta})$ & $(\psi == \psi_j^{\delta})$ **then**
4: return Δ
5: **else if** $T_i > T_j^{\delta}$ **then**
6: **for** $m = h$ to $p - 1$ **do**
7: **for all** ψ s.t. $\psi = \psi_j^{\delta} \pmod{a_m}$ and $\psi \neq \psi_j^{\delta} \pmod{a_{m+1}}$ **do**
8: $\Delta = \Delta \cup (a_m, 1, \psi)$.
9: **end for**
10: $\{ \frac{a_{m+1}}{a_m} - 1$ periodic intervals are created.$\}$
11: **end for**
12: **end if**
13: **return** Δ

order of periods [9], [13]. We however need a stronger test that works also 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, we need to define more concepts. 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. Likewise, we say that $v(E) = (e_1, e_2, \cdots, e_l)$ is the canonical vector of set E when e_i is the number of PIs in E whose period is exactly a_i. 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.

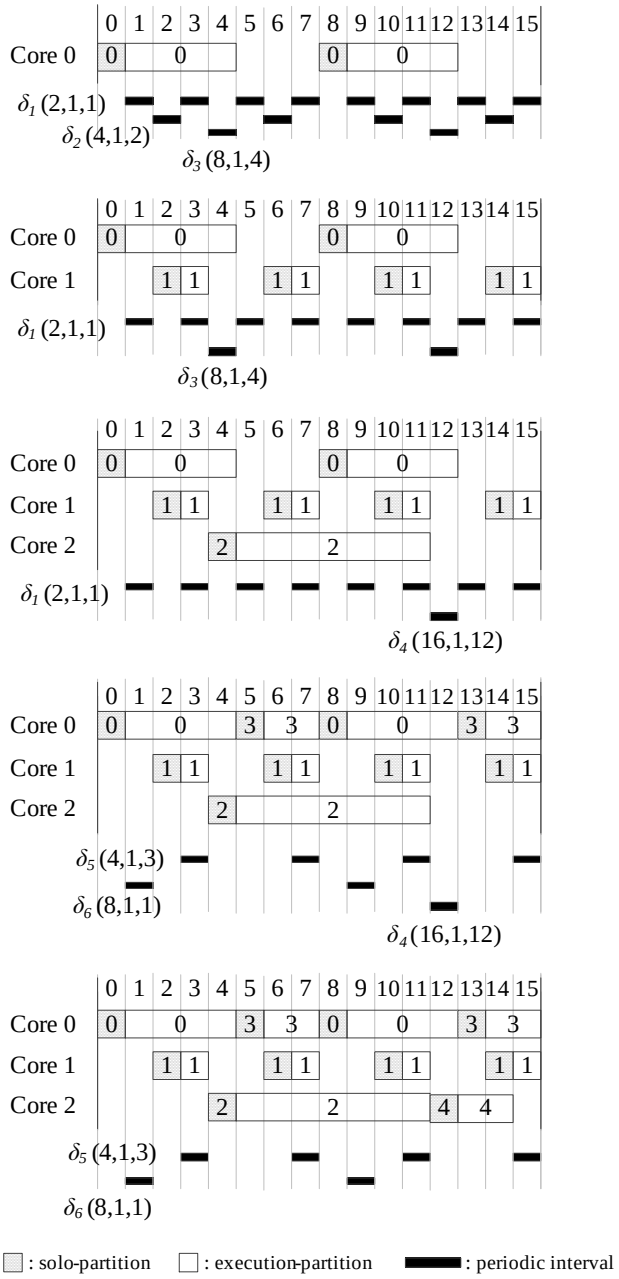


Fig. 4. An example of partition assignment and scheduling for five partitions generated by our approximation algorithm.

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 want to 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 thus omitted. □

C. Example

Fig. 4 shows how to generate a schedule for an example set of five partitions with the presented algorithm. Note that after π_0 is assigned, π_1 finds its offset at slot 2 of δ_2, not slot 4 of δ_3, since its period is 4. Also note that π_3 takes slot 5 on C_0 by updating δ_1. If it were to utilize slot 12 of δ_4, an additional core would be required since π_3 cannot fit into any existing core with the offset of 12.

V. Evaluation

In this section, 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. 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 [14]. 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's running time is in millisecond order while CP's is in few dozens of minutes order.

Firstly, Fig. 5 shows the result of StraightMapping obtained by the presented constraint programming formulation. For the experimental parameters, except for the length of the solo-partition, the following parameters are fixed: four 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. During the experiment 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. We can see from the result that, in the stable conditions,

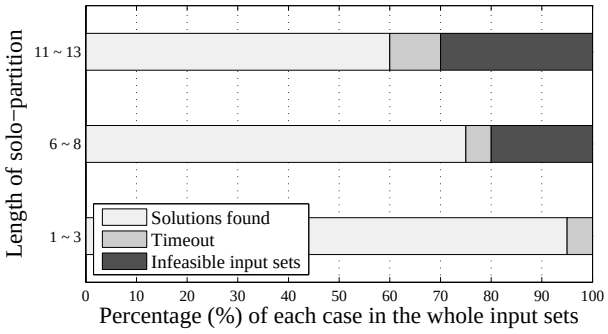


Fig. 5. Result of *StraightMapping* obtained by Constraint Programming in Sec. III-B.

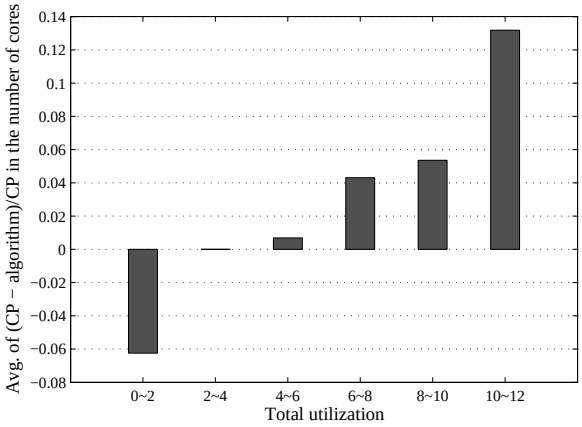


Fig. 6. Average relative differences between the required number of cores by our algorithm and CP of MinCores divided by CP according to the six total utilization range groups.

except for the length of solo-partitions, the proportion of infeasible sets increases only as the solo-partition lengths increase. This shows 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 can be found by CP. For this evaluation, we generated 200 random input sets, each of which consists of $\{5,6,\cdots,50\}$ 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 into six different range groups according to their 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 we can see from the result, 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 thus shows how well our algorithm can be scalable and efficient.

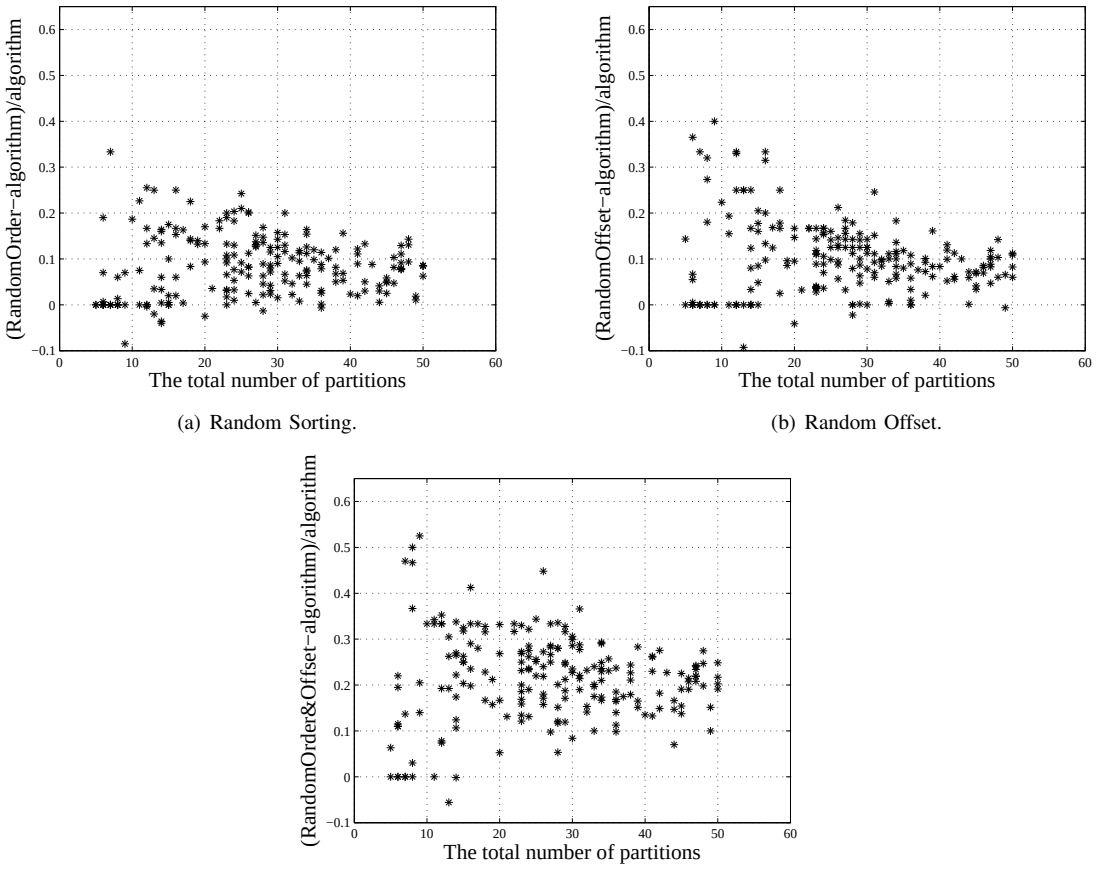
Lastly, we evaluated our approximation algorithm by adding randomness in order to justify the theoretical effectiveness of the algorithm. We compared the original algorithm described in Sec. IV-B and the following three variations:

1) Random Sorting: Partitions are packed into cores in a random order.
2) 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.
3) Random Sorting and Random Offset: Both of the above combined.

For this evaluation, we used the same experimental parameters as in Fig. 6. For each input set, we take the average of the results of 100 executions. First of all, as we can see from the results, the added randomness increased the required number of cores, as expected. This is worse in the case of (c); in the worst-case, it used more than 1.5 times of the number of cores that were found by the original algorithm, and could not output the same solution in the range over 20 partitions. Although there are some cases where the variations found better solutions than the original algorithm, these results show the theoretical importance of the proposed algorithm.

VI. Related Work

On a general IMA architecture, a set of partitions runs in Time Division Multiple Access (TDMA) according to a cyclic scheduling table constructed by partition periods and pre-specified time durations. Within a partition, a set of applications are scheduled by a local scheduling policy which is, in general, a fixed-priority preemptive scheduling such as Rate Monotonic [15]. IMA-partition scheduling is a class of *strictly periodic* scheduling due to the nature of cyclic and nonpreemptive scheduling; that is, the distance between the *starting times* of two consecutive executions of a partition is exactly the same as its period, that is, *strictly periodic*. The periodicity of



(a) Random Sorting.

(b) Random Offset.

(c) Random Sorting & Random Offset.

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

tasks in hard real-time scheduling has been addressed from different aspects in previous literature. The Pinwheel problem [13] 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 [9], Han *et al.* proposed a similar task model called a distance-constrained task system (DCTS), in which the distance between the *finishing times* of consecutive executions are constrained to be no longer than a certain threshold. These two classes of task models, however, differ from our partition scheduling model in that a task can start its execution at an arbitrary time instant as long as the constraint can be met in each respective model. A more closely related work to our scheduling problem is [5], [16]; Korst *et al.* addressed the problem of scheduling a set of strictly periodic, nonpreemptive tasks on the minimum number of processors. We adopt their mathematical property of a strictly periodic and nonpreemptive task model and extend it to our multi-IMA partition scheduling with exclusive regions. [17] 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 [18] 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 IMA partitions and bus channels, considering the timing requirements of tasks and messages. Although not directly assuming an IMA model, Tămaş–Selicean *et al.* [19], [20] considered an optimization problem of mixed-criticality, cost-constrained partitioned resources in a distributed architecture. In this work, the authors developed a meta-heuristic algorithm that finds the assignments of tasks and partitions as well as their schedules that can satisfy application schedulability while minimizing the development cost.

Lastly, in [2], a closely related and meaningful discussion on synchronization issues of distributed IMA

architectures is found. This article addressed the necessity of a special purpose partition called a *device management partition* on processors that communicates over a shared bus. Due to the exclusiveness on the communication line, the necessity of a global schedule that excludes simultaneous executions of device management partitions was addressed, which motivates our work.

VII. Conclusion

In this report, we have developed an optimized Multi-IMA partition scheduling algorithm which considers exclusive regions, *solo-partitions*, to achieve synchronization between partitions across cores on a multi-core system. We have shown that the problem of finding an optimal Multi-IMA partition schedule for StraightMapping problem is NP-complete and presented a Constraint Programming formulation. In addition, we have 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 that studies 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] D. Patterson, "The trouble with multi-core," *IEEE Spectrum*, vol. 47, no. 7, pp. 28–32, Jul. 2010.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. R. Garey and D. S. Johnson, "Tutorial: hard real-time systems," J. A. Stankovic and K. Ramamritham, Eds. IEEE Computer Society Press, 1989, ch. Complexity results for multiprocessor scheduling under resource constraints, pp. 205–219.

[13] 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.

[14] "IBM ILOG CPLEX CP Optimizer," http://www-01.ibm.com/software/integration/optimization/cplex-cp-optimizer/.

[15] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[16] J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels, "Periodic assignment and graph colouring," *Discrete Appl. Math.*, vol. 51, no. 3, pp. 291–305, Jul. 1994.

[17] 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.

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

[19] 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.

[20] ——, "Optimization of time-partitions for mixed-criticality real-time distributed embedded systems," in *Proc. of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2011, pp. 1–10.