Partitioned Scheduling of Recurrent Real-Time Tasks

Pontus Ekberg
Uppsala University
pontus.ekberg@it.uu.se

Sanjoy Baruah
Washington University in Saint Louis
baruah@wustl.edu

***Abstract*—The partitioned scheduling of periodic and sporadic task systems upon multiprocessor platforms (both identical and heterogeneous) is considered. The computational complexity of a large number of such partitioned schedulability problems is examined. New lower and upper bounds on complexity are presented for several problems. Some problems are pigeonholed into their precise complexity classes in this way. A list of problems for which exact classification remains open is compiled.**

***Index Terms*—Partitioned Scheduling; Sporadic and Periodic Tasks; Unrelated (Heterogeneous) Multiprocessors; Partitioned Simultaneous Congruences; Complexity Classification**

I. Introduction

Under the partitioned paradigm of multiprocessor scheduling for recurrent tasks, each task is pre-assigned to an individual processor and all jobs generated by a task may only execute upon the processor to which the task has been assigned. Schedulability analysis for partitioned scheduling – determining whether a given task system can be partitioned upon a specified platform in a manner that guarantees to always meet all deadlines – is easily shown to be NP-hard under most non-trivial assumptions regarding the task system or the platform, by reduction from the PARTITION or BIN PACKING problem [1, Problems SP12 and SR1]. Hence most prior work on schedulability analysis for partitioned scheduling has focused upon the design of good approximation algorithms for solving these problems (see, e.g., [2]–[9]).

Exact partitioned schedulability and ILPs. The research reported in this manuscript, in contrast, studies partitioned scheduling from the perspective of obtaining *exact* algorithms for doing partitioned schedulability analysis; given the inherent intractability of the problem, such algorithms are necessarily likely to have exponential worst-case running times. Still, in some cases it is possible to formulate a partitioned schedulability problem as an integer linear program (ILP), and to leverage the tremendous performance of optimized ILP solvers to achieve running times that are acceptable in practice for reasonably large problem instances. We detail exact ILP formulations that are known to exist for some important partitioned schedulability problems, and apply results from computational complexity theory [10] to show that it is unlikely that many other important problems can be represented as ILPs in polynomial time.

Classifying computational complexity. We further seek to identify the precise computational complexity of those problems that seem unlikely to be equivalent to ILPs. Our efforts in this direction may be looked upon as a furthering of the effort initiated by Eisenbrand and Rothvoß [11], [12] over a decade ago, and continued by Ekberg and Yi [13]–[16] since then, at providing a deeper theoretical understanding of schedulability issues for some of the foundational workload models in real-time computing. From the works cited above we now know that most schedulability analysis problems that the real-time systems community cares about are NP-hard or coNP-hard since many generalize one of the problems shown to be NP- or coNP-complete in the above works, while only a select few (such as feasibility of implicit-deadline [17] task systems) problems are in P.

However, any binary classification that simply puts each schedulability problem either among the "difficult" problems due to being NP- or coNP-hard or among the "easy" problems in P provides an over-simplified view of matters. Indeed, we believe most in the real-time systems community would agree that there seems to be a *qualitative difference* between the difficulty of analysing, say, rate-monotonic schedulability of implicit-deadline sporadic tasks on uniprocessors (which we know is NP-complete [15]) and doing the same for, say, global fixed-priority schedulability of arbitrary-deadline asynchronous periodic tasks on multiprocessors (which seems much harder still). Simply stating that both are difficult because they are NP-hard would not differentiate their hardness; while it is technically correct to state that both are NP-hard, it likely provides us with a very incomplete picture. Fortunately, computational complexity theory offers a rich structure of complexity classes for problems that seem *harder* than any problem in NP or coNP. In this work we will pinpoint some of the basic partitioned schedulability problems at higher levels of the *polynomial hierarchy* [18]–[20], showing that they are, in fact, qualitatively harder than their uniprocessor counterparts.

Contributions. Our major contribution is *a methodical categorization of the computational complexity of partitioned scheduling problems* (summarized in Fig. 2). For some problems we detail polynomial-time reductions to (the decision version of) ILP. Such reductions provide constructive proof that these problems are in NP. Since all the studied problems are obviously NP-hard as well (as mentioned above), this establishes that the problems that can be reduced to ILP are in fact NP-complete. For other problems we apply results from computational complexity theory to show that they belong to complexity classes that are widely believed to be strictly harder than NP, and so unlikely to be representable as ILPs in

polynomial time. We pinpoint the precise complexity classes of some of these harder-than-NP-or-coNP problems; as a further contribution we identify gaps between new and known lower and upper bounds on complexity for those partitioning problems that we have not been able to pinpoint exactly. In order to maximize the applicability of our results, all upper bounds and algorithms (ILPs) that we present have been designed to be valid for heterogeneous (unrelated) multiprocessors, while all lower bounds are shown valid also for identical multiprocessors. This means that the problems that are exactly pinpointed remain so in both of these settings.

Organization. We provide a brief review of some concepts and results from computational complexity theory in Section II, and demarcate the scope of the problems considered in this paper by formally laying out our workload and platform model in Section III. In Section IV we demonstrate that two basic partitioning problems are in fact in NP, by giving ILPs for solving these problems exactly. Section V is primarily focused on *lower bounds* – on showing that some problems are hard for certain complexity classes and therefore cannot, in all likelihood, be represented as ILPs in polynomial time. This section includes a proof that a natural generalization of the Simultaneous Congruences Problem, which has been widely used (see, e.g., [21]–[23]) to prove intractability for uniprocessor scheduling problems, is complete at a higher level of the polynomial hierarchy; we are optimistic that this result will play a role in helping determine the precise complexity of several partitioned multiprocessor scheduling problems. In Section VI we establish upper bounds upon the computational complexity of some partitioned schedulability problems, showing that they are no harder than the hardest problems of certain complexity classes. Upon combining the lower and upper bounds we find that the complexity of some of the studied schedulability problems have been exactly pinpointed, while others still have a gap between the bounds. We summarise this current state of knowledge for partitioned multiprocessor schedulability of recurrent task systems in Section VII, and close out in Section VIII with some brief concluding remarks.

II. Some Background from Complexity Theory

We now provide a brief introduction to concepts of computational complexity theory that are used in this manuscript.[1] We will make reference to the following complexity classes:

- P is the set of problems that can be *solved* by algorithms with running time polynomial in the size of their inputs.
- NP is the set of problems with solutions that can be *verified* by algorithms with running time polynomial in the size of their inputs; coNP is the set of problems whose complements are in NP.
- Σ_2^{P} is the set of problems that can be verified in polynomial time by an algorithm that has access to an *oracle* for some NP-complete problem, where an oracle can be thought of as a "black box" that is able to solve a specific decision problem in a single step. Similar to how coNP relates to NP, we have Π_2^{P} that is the set of problems whose complements are in Σ_2^{P}.
- Σ_3^{P} is the set of problems that can be verified in polynomial time by an algorithm that has access to an oracle for some problem that is complete for Σ_2^{P}. Again, Π_3^{P} is the set of problems whose complements are in Σ_3^{P}.

The above idea is generalized for any $k \in \mathbb{N}$: Σ_k^{P} and Π_k^{P} are defined assuming access to an oracle that is complete for $\Sigma_{k-1}^{\mathsf{P}}$ (we will only see the first three levels, explicitly listed above, in this paper). This infinite hierarchy of complexity classes is called *the polynomial hierarchy*. The relationship amongst these complexity classes is shown in Figure 1 as a Hasse diagram depicting the subset relationship (i.e., an arrow $A \rightarrow B$ in Figure 1 pointing from A to B denotes that complexity class A is a subset of complexity class B: each problem that falls in complexity class A is also contained in complexity class B). It is widely believed that each such depicted containment is proper – i.e., each depicts the $\subsetneq$ relationship.[2]

An alternate view on the polynomial hierarchy. Instead of defining the levels in the polynomial hierarchy by using *oracles* from the levels below, one can equivalently define the polynomial hierarchy by the logical structure of the problems contained in it. We find both views very useful, and therefore briefly outline this alternate view as well.

In the logical view, a problem A in NP with input x can be posed as a question of the form $A(x) = \exists w : \psi(x, w)$, where w is a polynomial-sized *witness* and $\psi(x, w)$ is a function in P that verifies that x belongs to the problem A with the help of the witness w. For example, if our NP-problem A is SATISFIABILITY, then x is a boolean formula, while w can be a truth assignment of variables and ψ the function that simply verifies if the truth assignment w satisfies x.

Similarly, a problem A in coNP with input x can be posed as a question of the form $A(x) = \forall w : \psi(x, w)$, where $\psi(x, w)$ is a function in P.

It can be noted that the distinguishing feature of NP is that *solutions* (or witnesses), which show that some input x belongs to the problem, are easily verified. On the other hand, the distinguishing feature of coNP is that *counterexamples*, which show that some input x *does not* belong to the problem, are easily verified.

The higher levels of the polynomial hierarchy can then be defined in the same vein by nesting alternating quantifiers, with more quantifiers further up in hierarchy. A problem $A(x)$ in Σ_2^{P} can be posed in the form $A(x) = \exists w_1 \forall w_2 : \psi(x, w_1, w_2)$, while a problem $A(x)$ in Π_2^{P} can be posed in the form $A(x) = \forall w_1 \exists w_2 : \psi(x, w_1, w_2)$. In general, problems at the k-th level of the polynomial hierarchy can be posed as questions with k alternating quantifiers, resulting in class Σ_k^{P} if the first quantifier is $\exists$ and in class Π_k^{P} if the first quantifier is $\forall$.

[1] The presentation of this material is intentionally kept quite informal.

[2] If the containment at some level k would not be proper, then the entire polynomial hierarchy above level k would *collapse* to that level (i.e., we would have $\Sigma_k^{\mathsf{P}} = \Sigma_w^{\mathsf{P}} = \Pi_w^{\mathsf{P}}$, for all $w > k$). In the extreme case, where $\mathsf{P} = \mathsf{NP}$, the entire hierarchy collapses to P, but any collapse further up the hierarchy would be very astonishing as well.

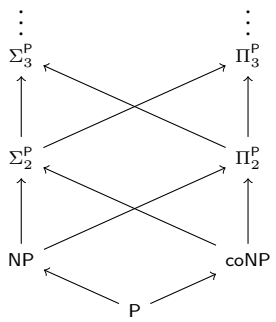

Fig. 1. Complexity classes considered in this paper

Relevance to ILP. It has been shown [24] that the problem of determining whether an ILP has a solution or not is NP-hard in the strong sense. It was subsequently shown [25] that the decision version of ILP is in NP, thereby establishing that ILP is in fact *complete* for the complexity class NP. Recall from complexity theory [10] that a problem is said to be *hard* for a complexity class of the kinds discussed above if any problem in the class can be reduced to this problem in polynomial time. Since ILP belongs to the class NP and it is widely believed that each of the subset relationships depicted in Figure 1 is a proper one (as mentioned above), we have the following.

Proposition 1. *Showing a problem to be hard for any of the complexity classes Σ_k^P or Π_k^P, for $k \geq 2$, offers strong evidence that it cannot be represented as an ILP in polynomial time.*

Another result from complexity theory that we will use concerns the complexity classes NP and coNP. It is widely believed that these two classes are not equal: that there are problems in NP that are not in coNP and vice versa.[3] Hence, we get the following.

Proposition 2. *Showing a problem to be hard for both* NP *and* coNP *offers strong evidence that it is not* in *either of these two complexity classes (and therefore cannot be represented as an ILP in polynomial time).*

It should be noted that while we have some focus on ILP in this paper, all the above still hold if we replace ILP with any other problem in NP, such as SAT.

III. Task and Platform Model

We will consider the problem of partitioning a given collection $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n recurrent tasks upon a platform of m unrelated processors. Each task τ_i is characterized by m *worst-case execution time* (WCET) parameters $\langle C_{i1}, C_{i2}, \ldots, C_{im} \rangle$ where C_{ij} denotes the maximum duration that a job of τ_i would take to complete execution upon the j'th processor; a *relative deadline* D_i; and a *period* T_i. Each task τ_i may release a potentially unbounded sequence of jobs during any given execution of the task system; each job released by τ_i is required to complete execution within a duration D_i of its release time. We will consider systems of both *sporadic* and *periodic* tasks: for sporadic tasks the period T_i represents the minimum duration between the release of successive jobs while for periodic tasks T_i represents the exact duration between the release of successive jobs. A periodic task τ_i is additionally characterized by an *initial release time* or *offset* O_i, and hence releases jobs at instants $O_i + k \cdot T_i$ for all $k \in \mathbb{N}$.

Synchronous and asynchronous periodic task systems. Periodic task systems may be further characterized as being *synchronous* if they satisfy the additional constraint that their initial release times are all equal (in which case they are typically assumed to be equal to zero, and omitted from the specifications). Periodic task systems that are not synchronous, i.e., in which all tasks do not have equal initial release times, are called *asynchronous* periodic task systems.

Implicit-, constrained- and arbitrary-deadline systems. Periodic or sporadic task systems are said to be *implicit-deadline* if $D_i = T_i$ for all tasks (the D_i parameter is typically dropped from the specification of tasks in such systems), and *constrained-deadline* if $D_i \leq T_i$ for all tasks. If no such restrictions are placed on the D_i and T_i parameters (i.e., we may have $D_i > T_i$), then the task system is said to be *arbitrary-deadline*.

EDF and FP scheduling. We will consider the partitioning of recurrent task systems when each individual processor is scheduled preemptively during run-time using either the Earliest Deadline First (EDF) scheduling algorithm, or some Fixed-Priority (FP) scheduling algorithm. When considering FP scheduling, we will assume that tasks are indexed according to priorities: τ_i has greater priority than τ_{i+1} for all $i, 1 \leq i < n$.

We now briefly state the following well-known results regarding EDF and FP scheduling of recurrent task systems upon preemptive uniprocessors. Let C_i denote the WCET of $\tau_i \in \Gamma$ upon the sole processor.

- A necessary and sufficient condition for any implicit-deadline periodic or sporadic task system Γ to be successfully scheduled using EDF is that
$$\sum_{\tau_i \in \Gamma} \frac{C_i}{T_i} \leq 1. \tag{1}$$
- For constrained-deadline sporadic and synchronous periodic task systems Γ that are scheduled under fixed priorities, the *response time* R_i of task $\tau_i \in \Gamma$ —the maximum duration that may elapse between the instant that a job of task τ_i arrives and the instant it completes execution— is equal to the smallest positive solution of the following recurrence:[4]
$$R_i \geq \sum_{j \leq i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{2}$$

[3] It has been shown that if NP = coNP, then the polynomial hierarchy collapses to NP.

[4] Since the *smallest* solution is necessarily satisfied with equality, this is often written with an = rather than a $\geq$.

(Recall that tasks in Γ are assumed indexed according to priorities: the index j in the summation therefore ranges over all tasks with greater priorities that τ_i, plus the task τ_i.)

IV. Problems with ILP Representations

In this section we describe polynomial-time algorithms for obtaining ILP representations of (i) partitioned EDF scheduling of implicit-deadline sporadic or periodic task systems (Sec. IV-A), and (ii) partitioned FP scheduling of constrained-deadline sporadic or synchronous periodic task systems (Sec. IV-B).

A. EDF scheduling of implicit-deadline systems

The ILP representation of this problem is straight-forward and has been part of the folklore in scheduling theory for decades; we present it here for the sake of completeness. The ILP is constructed using $(n \times m)$ zero-one integer *variables* x_{ik} for each $(i,k) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\}$. Variable x_{ik} has the following intended interpretation:

$$x_{ik} = 1 \text{ if } \tau_i \text{ is assigned to the } k\text{'th processor} \quad (3)$$

The *constraints* in the ILP, and their intended interpretation, are as follows:

- Each task must be assigned to some processor:

$$\sum_{k=1}^{m} x_{ik} \geq 1 \text{ for each } i, 1 \leq i \leq n \quad (4)$$

- The tasks assigned to each processor are successfully scheduled during run-time; this is assured by appealing to the uniprocessor EDF-schedulability condition of Expression 1:

$$\sum_{i=1}^{n} x_{ik} \cdot \frac{C_{ik}}{T_i} \leq 1 \text{ for each } k, 1 \leq k \leq m \quad (5)$$

It is fairly obvious that the above ILP, comprising $n \times m$ zero-one integer variables and $n + m$ linear constraints, has a solution if and only if implicit-deadline sporadic or periodic task system Γ can be partitioned amongst the m processors.

B. FP scheduling of constrained-deadline systems

Here we present an ILP for solving the partitioned FP-schedulability problem for constrained-deadline systems upon unrelated heterogeneous processors. While this ILP is more involved than the ILP in Sec. IV-A, it also has only a polynomial number of variables and constraints and can easily be constructed in polynomial time from a given task set. (The key ideas here can readily be extracted from an ILP by Zheng et al. [26] that was created for calculating end-to-end response times in distributed systems.)

We first list the *variables* used in the ILP.

1) As in Section IV-A, we have $(n \times m)$ zero-one x_{ik} variables with the same intended interpretation (articulated in Expression 3).
2) We have n positive real-valued variables $R_1, R_2, \ldots, R_n$, where R_i represents the worst-case response time of τ_i.
3) We have $(n^2 \times m)$ zero-one integer valued variables s_{ijk} for all $(i,j,k) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \times \{1,\ldots,m\}$, with the intended interpretation that

$$s_{ijk} = 1 \text{ if } \tau_i \text{ and } \tau_j \text{ are both assigned to processor } k. \quad (6)$$

4) We have $(n^2 \times m)$ non-negative integer variables Z_{ijk} for all $(i,j,k) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \times \{1,\ldots,m\}$, with the intended interpretation that

$$Z_{ijk} \geq \left\lceil \frac{R_i}{T_j} \right\rceil \text{ if } \tau_i \text{ and } \tau_j \text{ are both assigned to processor } k. \quad (7)$$

We now list the *constraints* in our ILP, that enforce the intended interpretations of the variables defined above.

1) Each task must be assigned some processor: the Constraints 4 of Section IV-A are used here as well.
2) We enforce the intended interpretation of the s_{ijk} variables by having a constraint

$$s_{ijk} \geq x_{ik} + x_{jk} - 1 \quad (8)$$

for each $(i,j,k) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \times \{1,\ldots,m\}$. The above constraint indeed ensures the intended interpretation for s_{ijk}: if τ_i and τ_j are both assigned to the k'th processor then $x_{ik} = 1$ and $x_{jk} = 1$, and the RHS of the constraint becomes $1 + 1 - 1 = \mathbf{1}$.
3) We enforce the intended interpretation in the Z_{ijk} variables by having a constraint

$$Z_{ijk} \geq \frac{R_i}{T_j} - M(1 - s_{ijk}) \quad (9)$$

for each $(i,j,k) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \times \{1,\ldots,m\}$. Here M is a sufficiently large positive constant; hence the RHS of the expression is bounded from below by a negative number if s_{ijk} is 0, and by R_i/T_j if s_{ijk} is 1. The non-negative integer variable Z_{ijk} thus has the intended interpretation: it is $\geq \lceil R_i/T_j \rceil$ if τ_i and τ_j are both assigned to the k'th processor.
4) For each $(i,k) \in \{1,\ldots,n\} \times \{1,\ldots,m\}$, we have a constraint

$$\sum_{j=1}^{i} Z_{ijk} \times C_{jk} \leq R_i \quad (10)$$

These constraints apply the response-time recurrence (Expression 2) to bound the worst-case response time of task τ_i, in the following manner. Suppose τ_i is assigned to the k'th processor (i.e., $x_{ik} = 1$). The Constraints 9 ensure that for each task τ_j also assigned to the k'th processor we will have $Z_{ijk} \geq \lceil R_i/T_j \rceil$. Constraint 10 is then simply a restatement of the response-time recurrence (Expression 2) for task τ_i, on the k'th processor.
5) Finally, we introduce n constraints, one for each task τ_i, requiring that τ_i's worst-case response time, represented by the variable R_i, not exceed its relative deadline:

$$R_i \leq D_i \quad (11)$$

It is evident that this ILP is correct, and that it is generated in

polynomial time. It has $(nm+n^2m+n^2m) = \boldsymbol{(nm+2n^2m)}$ integer variables (of which all but n^2m are zero-one variables) plus $\boldsymbol{n}$ real-valued variables (the R_i's), and a total of $(n + n^2m + n^2m + nm + n) = \boldsymbol{(2n + 2n^2m + nm)}$ constraints. Hence it is indeed a polynomial-sized ILP as claimed.

V. ILP-Intractable Partitioning Problems

We now look at partitioned scheduling problems for which it can be shown that it is *unlikely* that they can be represented as ILPs in polynomial time (since doing so would disprove widely held conjectures in complexity theory). First in Section V-A we explain why the ILPs of Section IV probably cannot be extended to more general task models. In Section V-B we introduce the PARTITIONED SCP problem and establish its computational complexity; we use PARTITIONED SCP in Section V-C to show that some additional partitioning problems are unlikely to be representable as ILPs in polynomial time.

A. Some hardness results

Let us first explore the possibility of extending the ILP of Section IV-A, which partitions implicit-deadline tasks for EDF scheduling, to constrained-deadline sporadic (or synchronous periodic) systems. This problem has been looked at previously [27]–[31]. Baruah and Bini [27] obtained an ILP representation of this problem with polynomially many variables but *exponentially* many constraints. They showed how this ILP could be modified to obtain approximations that have polynomially many constraints at a speedup cost that depends upon the degree of approximation. Marchetti-Spaccamela et al. subsequently [28]–[31] improved this result to show that polynomial-sized ILPs are possible if one is willing to pay a speedup cost larger than 2. However the following theorem, in conjunction with Proposition 2, offers strong evidence that we are unlikely to obtain speedup-optimal ILP representations in polynomial time:

Theorem 1. *The partitioned EDF-schedulability problem for sporadic (or synchronous periodic) tasks with constrained or arbitrary deadlines is both strongly* NP*-hard and* coNP*-hard, even when restricted to $m = 2$ identical processors.*

Proof. Already the uniprocessor version of this problem is strongly coNP-complete [13], and therefore the partitioned version is clearly strongly coNP-hard for $m = 2$ processors.

The partitioned EDF-schedulability problem is strongly NP-hard even for implicit-deadline tasks ($C_i = D_i$ for all $\tau_i \in \Gamma$) and $m = 2$ processors: this is easily shown by reducing from PARTITION WITH RATIONAL WEIGHTS, which is known [32] to be strongly NP-complete (the reduction maps the rational weights to task utilizations). □

Proposition 2 states that problems that are both NP and coNP hard are unlikely to have efficient ILP representations; hence the ILP of Section IV-A is unlikely to extend to constrained-deadline systems.

Extending the ILP of Section IV-B to the partitioned FP-scheduling of arbitrary-deadline sporadic or synchronous periodic systems also seems unlikely, since even the uniprocessor version of this problem is not known to be in NP.[5]

B. Partitioned simultaneous congruences

The *Simultaneous Congruences Problem* (SCP) – see Definition 1 below – has played an important role in establishing the computational complexity of several basic (uniprocessor) scheduling problems concerning periodic real-time task systems (see, e.g., [21]–[23], [33]). SCP was shown to NP-complete in [22]; this was later strengthened in [34] to show that SCP is in fact NP-complete in the strong sense.

In this section we define and study a generalization to SCP that suggests itself as a natural extension for the purposes of studying partitioned scheduling of periodic task systems. We prove (Theorem 2 below) that this generalization, which we have called PARTITIONED SCP (Definition 2), is complete for the complexity class Σ_2^{P}. We can look upon Theorem 2 as the analogue in the second level of the polynomial hierarchy depicted in Figure 1, of the NP-hardness of SCP that was proved in [22], [34]. In fact in the following sections of this paper we will use PARTITIONED SCP to pigeonhole the computational complexity of several partitioned scheduling problems, in much the same manner as SCP was used [21]–[23], [33] to establish the complexity of many uniprocessor scheduling problems.

Definition 1 (Simultaneous Congruences Problem (SCP))**.**
INSTANCE: $\langle A, k\rangle$, where $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ is a set of ordered pairs of non-negative integers satisfying $a_i < b_i$ for each i, and k is a positive integer that is $\leq n$.

QUESTION: Is there an integer x and an $A' \subseteq A$ such that $|A'| = k$ and $(x \equiv a_i \pmod{b_i})$ for each $(a_i, b_i) \in A'$? □

Any instance $\langle A, k\rangle$ for which the answer to the above question is "yes" is said to *belong to* SCP; this may be expressed formally as "$\langle A, k\rangle \in$ SCP".

We now define PARTITIONED SCP as a natural generalization to the SCP:

Definition 2 (PARTITIONED SCP)**.**
INSTANCE: $\langle A, k, m\rangle$, where $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ is a set of ordered pairs of non-negative integers satisfying $a_i < b_i$ for each i, and k and m are positive integers both $\leq n$.

QUESTION: Is there an m-partitioning $\langle A_1, \ldots, A_m\rangle$ of A such that $\langle A_i, k\rangle \notin$ SCP for each i, $1 \leq i \leq m$? In other words, is it the case that there does not exist an integer x and an $A' \subseteq A_i$ for any partition A_i such that $|A'| = k$ and

$$x \equiv a_i \pmod{b_i}$$

for each $(a_i, b_i) \in A'$? □

Similarly to above, we will say an instance $\langle A, k, m\rangle \in$ PARTITIONED SCP if the answer to the question above upon instance $\langle A, k, m\rangle$ is "yes".

[5] In Lemma 6.1, we will present the first upper bound on this uniprocessor scheduling problem's complexity: we show that it is in Π_2^{P}.

The remainder of this section is devoted to showing that PARTITIONED SCP is a highly intractable problem; this is formally stated in Theorem 2 below. But before getting to that we introduce an additional problem called the GENERALIZED GRAPH COLORING problem[6] that we will use to prove Theorem 2, that has previously been considered in the computational complexity literature. The following special case of GENERALIZED GRAPH COLORING was shown [35, Theorem 5] to be Σ_2^{P}-complete (although we will refer to this special case as the GENERALIZED GRAPH COLORING problem in the remainder of this manuscript, we emphasize here that it is in fact only a special case of the more general problem of this name).

Definition 3 (GENERALIZED GRAPH COLORING)**.**
INSTANCE: $\langle G, k\rangle$, where G is a graph and k a positive integer.
QUESTION: Is there a 2-coloring of G that does not contain a clique of size k in which all vertices have the same color? (I.e., no set of k vertices, all of which have edges to each other, should all be of the same color.) □

Similarly to above, we will say an instance $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING if the answer to the question above upon instance $\langle G, k\rangle$ is "yes".

Theorem 2. PARTITIONED SCP *is* Σ_2^{P}*-complete.*

Proof. It is evident that PARTITIONED SCP $\in \Sigma_2^{\mathsf{P}}$: given an instance $\langle A, k, m\rangle$ we can simply guess an m-partitioning $\langle A_1, \ldots, A_m\rangle$ of A in polynomial time, and use an oracle for SCP, which is known [22], [33] to be NP-complete, to verify that each $\langle A_i, k\rangle \notin$ SCP for each $i, 1 \leq i \leq m$.

We now show that PARTITIONED SCP is Σ_2^{P}-hard, by defining a polynomial-time reduction from GENERALIZED GRAPH COLORING to PARTITIONED SCP. That is, given a graph G we describe below how to construct a set $f(G)$ of ordered pairs of integers in time that is polynomial in the number of vertices in G, such that for any integer k, $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING if and only if $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP. This, in conjunction with the Σ_2^{P}-hardness of GENERALIZED GRAPH COLORING [35, Theorem 5], establishes that PARTITIONED SCP is Σ_2^{P}-hard (since one can determine whether $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING by first computing $f(G)$ in polynomial time, and then checking whether $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP).

We now describe how, given a graph $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E the set of edges, we obtain the set $f(G)$ of n ordered pairs of integers – we point out that this construction is essentially identical to the one introduced by Leung and Whitehead [22, Theorem 3.6] for the purpose of showing that SCP is NP-hard. Our construction of $f(G)$ uses $(n^2 - n)/2$ distinct prime numbers; hence we point out that (i) it follows from the *Prime Number Theorem* (see, e.g., [36]) that the first n^2 prime numbers are bounded in value by a polynomial in n; and (ii) they can therefore be determined in time polynomial in n even by a relatively naïve algorithm such as the Sieve of Eratosthenes.

Let us refer to $(n^2 - n)/2$ distinct prime numbers as p_{ij} for each pair (i, j), $1 \leq i < j \leq n$. We construct the b_i's in the following manner:

$$b_i = \prod_{j<i} p_{ji} \times \prod_{i<j} p_{ij} \quad \text{for each } i, 1 \leq i \leq n \tag{12}$$

and the a_i's as follows:

$$a_i = \prod_{j<i} p_{ji} \times \prod_{\substack{i<j \wedge \\ (v_i, v_j)\in E}} p_{ij} \quad \text{for each } i, 1 \leq i \leq n \tag{13}$$

It is evident that this reduction takes time polynomial in n. It now remains to show that it is correct: i.e., that for any integer k, $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING if and only if $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP.

Lemma 2.1. $a_i \equiv a_j \pmod{gcd(b_i, b_j)}$ *iff* $(v_i, v_j) \in E$.

Proof. Without loss of generality assume that $i > j$ (the case for $i < j$ is symmetric).

- From Expression 12, $\gcd(b_i, b_j) = p_{ji}$, since that is the only prime number that b_i and b_j have in common (by construction).
- From Expression 13, p_{ji} is a factor of a_i; however it is a factor of a_j if and only if $(v_i, v_j) \in E$. Therefore $a_i \equiv 0 \pmod{p_{ji}}$, but $a_j \equiv 0 \pmod{p_{ji}}$ if and only if $(v_i, v_j) \in E$.

The lemma follows. □

Lemma 2.2. $V' \subseteq V$ *is a clique in* G *if and only if* $a_i \equiv a_j \pmod{gcd(b_i, b_j)}$ *for each pair of vertices* $v_i, v_j \in V'$.

Proof. Consider any $V' \subseteq V$. By definition of clique, V' is a clique in G iff there is an edge between each pair of vertices in V'. It follows by Lemma 2.1, that V' is a clique in G iff $a_i \equiv a_j \pmod{\gcd(b_i, b_j)}$ for each pair of vertices $v_i, v_j \in V'$. □

To complete the proof of Theorem 2, we must show that $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING if and only if $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP. We consider the "*if*" and "*only if*" separately below.

Suppose $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING: Consider the 2-coloring of G such that there is not a clique of size k of either color. Partition the ordered pairs in $f(G)$ into two sets S_1 and S_2 according to the color of the vertices to which they correspond; it follows from Lemma 2.2 and the Generalized Chinese Remainder Theorem [36] that $\langle S_1, k\rangle \notin$ SCP and $\langle S_2, k\rangle \notin$ SCP. Hence $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP.

Suppose $\langle f(G), k, 2\rangle \in$ PARTITIONED SCP: Consider some 2-partitioning of $f(G)$ into S_1 and S_2 such that $\langle S_1, k\rangle \notin$ SCP and $\langle S_2, k\rangle \notin$ SCP. By Lemma 2.2 and the Generalized Chinese Remainder Theorem, the vertices corresponding to all the ordered pairs in neither S_1 nor S_2 contain cliques of k vertices; hence color all the vertices corresponding to

[6]Listed as Problem [GT9] in the compendium [20] of problems that are complete for various levels of the polynomial-time hierarchy.

ordered pairs in S_1 to be of one color and those corresponding to ordered pairs in S_2 of another color to obtain a 2-coloring of G that bears witness to the fact that $\langle G, k\rangle \in$ GENERALIZED GRAPH COLORING. □

Observe that the reduction in the proof of Theorem 2, an arbitrary instance $\langle G, k\rangle$ of GENERALIZED GRAPH COLORING was reduced to an instance $\langle f(G), k, \mathbf{2}\rangle$ of PARTITIONED SCP; this immediately yields the following corollary:

Corollary 3. PARTITIONED SCP *is* Σ_2^{P}*-complete even when restricted to two partitions.*

C. HARDNESS RESULTS FOR ASYNCHRONOUS SYSTEMS

In this section we apply the hardness results for PARTITIONED SCP obtained in Section V-B above to show that partitioning asynchronous periodic task systems is computationally highly intractable. Both reductions in this section were closely inspired by Leung and Whitehead [22], who reduced from (ordinary) SCP to schedulability of uniprocessor asynchronous periodic tasks.

As mentioned in Section I, we establish all lower bounds for *identical* multiprocessors in order to get the strongest results. Throughout this section we simply write C_i for the execution time of τ_i, since $C_{ij} = C_{ik}$ for all processors j, k.

Theorem 4. *The partitioned schedulability problem for* any *work-conserving scheduler, and the partitioned feasibility problem, is* Σ_2^{P}*-hard for asynchronous periodic tasks. This holds even when restricted to* $m = 2$ *identical processors, constrained deadlines and total utilization bounded from above by any constant* $c > 0$.

Proof. We create a polynomial-time reduction from PARTITIONED SCP to the considered problem. Given a PARTITIONED SCP instance $\langle A, k, m\rangle$, we transform this instance into a set Γ of $n \stackrel{\text{def}}{=} |A|$ asynchronous periodic tasks. For each $(a_i, b_i) \in A$, we have a $\tau_i \in \Gamma$ with the following parameters:

$$\begin{aligned} O_i &\stackrel{\text{def}}{=} \mu a_i, \\ C_i &\stackrel{\text{def}}{=} 1, \\ D_i &\stackrel{\text{def}}{=} k - 1, \\ T_i &\stackrel{\text{def}}{=} \mu b_i, \end{aligned}$$

where

$$\mu \stackrel{\text{def}}{=} k\left\lceil \frac{n}{c} \right\rceil.$$

We note that by this construction, $x \equiv a_i \pmod{b_i}$ if and only if task τ_i releases a job at time-instant μx, for $x \geq 0$.

We will now show that Γ is feasible with partitioned scheduling on m processors (and also schedulable by *any* work-conserving partitioned scheduler on m processors) if and only if $\langle A, k, m\rangle \in$ PARTITIONED SCP by considering the two directions separately.

$\langle A, k, m\rangle \in$ PARTITIONED SCP $\Rightarrow \Gamma$ is schedulable: Assume that $\langle A, k, m\rangle \in$ PARTITIONED SCP. Then there exists a partitioning $A_1, \ldots, A_m$ of A such that no partition A_j has a simultaneous collision of k or more congruence classes. That is, there is no number x and subset $A' \subseteq A_j$ of size k where $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'$.

Let $\Gamma_1, \ldots, \Gamma_m$ be a partitioning of Γ such that $\tau_i \in \Gamma_j$ if $(a_i, b_i) \in A_j$. It follows that for all partitions Γ_j, there is no time point where more than $k-1$ tasks release a job at the same time. Further, since O_i and T_i are multiples of μ for all tasks τ_i, any jobs that are not released at the same time are separated by at least $\mu \geq k$ time units. As all jobs have execution time 1 and relative deadline $k-1$, all jobs will meet their deadlines under any work-conserving scheduling.

$\langle A, k, m\rangle \notin$ PARTITIONED SCP $\Rightarrow \Gamma$ is not schedulable: Assume that $\langle A, k, m\rangle \notin$ PARTITIONED SCP. Then all possible m-partitionings of A will contain at least one partition A_j with a subset $A' \subseteq A_j$ where $|A'| \geq k$, such that $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'$ and for some x. It follows that all possible m-partitionings of Γ will contain a partition where at least k tasks release a job at the same time point. Since all tasks have execution time 1 and relative deadline $k-1$, no scheduler can successfully schedule k or more jobs that are released at the same time point.

Finally, we note that the the reduction is easily carried out in polynomial time, that PARTITIONED SCP is Σ_2^{P}-complete already when $m = 2$ (by Corollary 3) and that the utilization of the produced task set is

$$\sum_{\tau_i \in \Gamma} \frac{C_i}{T_i} = \sum_{\tau_i \in \Gamma} \frac{1}{\mu b_i} \leq \frac{n}{\mu} \leq c,$$

which completes the proof. □

Since Theorem 4 is applicable to any work-conserving scheduler, we immediately get Σ_2^{P}-hardness for both partitioned EDF and partitioned FP (regardless of the priority ordering) for constrained-deadlines asynchronous periodic tasks.

In the next theorem we see that partitioned FP schedulability *with a specified priority ordering* is Σ_2^{P}-hard also with implicit deadlines.

Theorem 5. *The partitioned FP-schedulability problem for asynchronous periodic tasks is* Σ_2^{P}*-hard when the priority ordering is given, even when restricted to* $m = 2$ *identical processors, implicit deadlines and total utilization bounded from above by* $c \times m$*, for any constant* $c > 1/2$.

Proof. Here, given a PARTITIONED SCP instance $\langle A, k, m\rangle$, we transform it into a set Γ of $n + m$ asynchronous periodic tasks with implicit deadlines, where $n \stackrel{\text{def}}{=} |A|$. For each $(a_i, b_i) \in A$, let τ_i be a task with the following parameters.

$$\begin{aligned} O_i &\stackrel{\text{def}}{=} \mu^2 a_i, \\ C_i &\stackrel{\text{def}}{=} \mu, \\ D_i = T_i &\stackrel{\text{def}}{=} \mu^2 b_i, \end{aligned}$$

where

$$\mu \stackrel{\text{def}}{=} 2kn\left\lceil \frac{n+m}{c - 1/2} \right\rceil.$$

Then we create m more tasks τ_{n+i}, for $1 \leq i \leq m$ that all have the same parameters:

$$\begin{aligned} O_{n+i} &\stackrel{\text{def}}{=} 0, \\ C_{n+i} &\stackrel{\text{def}}{=} k\mu + 1, \\ D_{n+i} \;=\; T_{n+i} &\stackrel{\text{def}}{=} 2k\mu. \end{aligned}$$

The set Γ contains the above $n+m$ tasks. The tasks τ_i, for $1 \leq i \leq n$, are all given high priorities, while the tasks τ_{n+i}, for $1 \leq i \leq m$, are all given low priorities. The internal priority ordering for the first n tasks is RM. The internal priority ordering for the latter m tasks does not matter as long as they all have lower priorities than the first n tasks.

Note that, by construction, task τ_i will release a job at time point $\mu^2 x$ if and only if $x \equiv a_i \pmod{b_i}$, for $1 \leq i \leq n$ and $x \geq 0$.

We now show that Γ is schedulable if and only if $\langle A, k, m\rangle \in$ PARTITIONED SCP, considering the two directions separately.

$\langle A, k, m\rangle \in$ PARTITIONED SCP $\Rightarrow \Gamma$ is schedulable: Assume that $\langle A, k, m\rangle \in$ PARTITIONED SCP, and that $A_1, \ldots, A_m$ is an m-partitioning of A such that no partition has a simultaneous collision of k or more congruence classes. Let $\Gamma_1, \ldots, \Gamma_m$ be an m-partitioning of Γ, where the n higher-priority tasks are assigned following the partitioning of A, such that $\tau_i \in \Gamma_j$ if $(a_i, b_i) \in A_j$. Further, we assign each lower-priority task τ_{n+j} to partition Γ_j.

We first note that the higher-priority tasks τ_i, for $1 \leq i \leq n$, are all schedulable. Their total utilization is

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = \sum_{i=1}^{n} \frac{\mu}{\mu^2 b_i} \leq \frac{n}{\mu} < \frac{1}{2},$$

and hence they are guaranteed to be schedulable by Liu and Layland's utilization bound [17], no matter how they are partitioned.

Then, to see that the low-priority task τ_{n+j} on each partition Γ_j is schedulable as well, we note that at most $k-1$ higher-priority tasks can release a job at the same time point on Γ_j, and that consecutive job releases from those tasks are at least μ^2 time units apart. Any job from the low-priority task then suffers at most $(k-1)\mu$ time units of interference, which can not cause it to miss its deadline since $C_{n+j} + (k-1)\mu < 2k\mu = D_{n+j}$.

$\langle A, k, m\rangle \notin$ PARTITIONED SCP $\Rightarrow \Gamma$ is not schedulable: Consider first the low-priority tasks τ_{n+i}, for $1 \leq i \leq m$. Since each of these has utilization $(k\mu+1)/(2k\mu) > 1/2$, each partition must have one of these tasks assigned to it, since otherwise some partition will be assigned at least two and therefore be unschedulable due to utilization overload.

Consider then the n high-priority tasks. Assume that $\langle A, k, m\rangle \notin$ PARTITIONED SCP, and hence that for all m-partitionings $A_1, \ldots, A_m$ of A, there exists some partition A_j and subset $A' \subseteq A_j$ with $|A'| \geq k$ and a number x, such that $x \equiv a_i \pmod{b_i}$, for all $(a_i, b_i) \in A'$. It follows that for any m-partitioning of the tasks τ_i, for $i \leq i \leq n$, there will be some partition where at least k of those tasks release a job at the same time point $\mu^2 x$, thereby creating high-priority workload of at least $k\mu$ time units at a this single time point.

If one of the low-priority tasks τ_{n+i} is also assigned to this partition, as it must if no partition should be overloaded with utilization > 1, then it will also release a job at time point $\mu^2 x$ since $\mu^2 x \equiv O_{n+i} \pmod{T_{n+i}}$. But that low-priority job then suffers interference of at least $k\mu$ time units from the higher-priority tasks, and it will miss its deadline since $C_{n+i} + k\mu = 2k\mu + 1 > 2k\mu = D_{n+i}$.

Finally, we (i) recall that PARTITIONED SCP is Σ_2^{P}-complete already for $m = 2$, (ii) note that Γ is easily produced in polynomial time and, (iii) see that the total utilization of Γ is

$$\begin{aligned} \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{m} \frac{C_{n+i}}{T_{n+i}} &\leq \frac{n}{\mu} + \frac{mk\mu + m}{2k\mu} \\ &\leq \frac{m}{2} + \frac{n+m}{\mu} \\ &\leq \frac{m}{2} + c - \frac{1}{2} \\ &\leq \frac{m}{2} + m\Big(c - \frac{1}{2}\Big) \\ &= cm, \end{aligned}$$

which completes the proof. □

VI. Upper Bounds on Complexity

In this section we show that certain partitioned scheduling problems belong to particular complexity classes and can therefore be no harder than the hardest problems in those classes.

It has been shown [21]–[23], [33], [37] that uniprocessor EDF-schedulability is in coNP. We begin by observing that moving to partitioned multiprocessors "lifts" the EDF-schedulability problem at most one level up in the polynomial hierarchy, from coNP to Σ_2^{P}, even in the most general setting:

Theorem 6. *Partitioned feasiblity (or, equivalently, partitioned EDF-schedulability) on m heterogeneous processors is in Σ_2^{P}, even for asynchronous periodic tasks with arbitrary deadlines.*

Proof. It is known [23] that the equivalent uniprocessor feasibility problem is in coNP, and therefore the uniprocessor *infeasibility* problem is in NP: Using the standard demand bound function analysis [23], a polynomial-time verifiable witness to infeasibility consists of two time points $\langle t_1, t_2\rangle$ such that $\mathrm{dbf}(t_1, t_2) > t_2 - t_1$.

Using the uniprocessor infeasibility problem as an NP-oracle it is easy to see that the partitioned feasibility problem is in Σ_2^{P}. A witness is simply an m-partitioning $\langle \Gamma_1, \ldots, \Gamma_m\rangle$ of Γ, and we can check whether each partition is feasible using the oracle. (It should be noted that it does not matter that the oracle is for infeasibility rather than feasibility since we can simply negate its answer.) □

Informally speaking, the partitioned EDF-schedulability problem is in Σ_2^{P} since it has the logical form "$\exists\forall$": does there *exist* a partitioning such that *for all* time intervals, no partition

is overloaded? Ultimately, this is because the uniprocessor EDF-schedulability problem is in coNP: the demand bound function analysis asks if *all* time intervals are non-overloaded.

In contrast, uniprocessor FP-schedulability for constrained-deadline sporadic/synchronous periodic systems is in NP rather than coNP, since it can be posed as "do there *exist* solutions to the response-time equation (Expression 2) that are $\leq D_i$ for each τ_i?". The partitioned FP-schedulability problem is then of the form "$\exists\,\exists$": does there *exist* a partitioning such that there *exist* solutions to the response-time equation on those partitions? Questions of the form "$\exists\,\exists$" can be flattened to just "$\exists$" by combining both witnesses into one, and therefore *partitioned FP-schedulability for constrained-deadline and implicit-deadline sporadic/synchronous periodic task systems is in* NP (indeed, that is why we were able to formulate the partitioned FP-schedulability problem as an ILP in polynomial time, in Section IV-B).

However, the smallest solution to Expression 2 equals the response time of a task under fixed-priority scheduling only for constrained-deadline and implicit-deadline sporadic/synchronous periodic task systems: this has been shown [38] to not be the case for arbitrary-deadline systems. As a consequence we are unable to extend the claim above (that partitioned FP-schedulability for constrained-deadline and implicit-deadline sporadic/synchronous periodic task systems is in NP) to arbitrary-deadline systems; in fact the best upper bound we have been able to establish for partitioned FP-schedulability with arbitrary deadlines and/or asynchronous periodic tasks is at the third level of the polynomial hierarchy, as we will see in Theorem 7. First we show that the corresponding uniprocessor FP-schedulability problem is in Π_2^{P}.

Lemma 6.1. *Uniprocessor FP-schedulability for asynchronous periodic tasks with arbitrary deadlines is in* Π_2^{P}.

Proof. Consider the decision problem CHECK JOB WITH BACKLOG that asks if *one* particular job J from one of the tasks τ_i in a task system Γ is schedulable on a uniprocessor, under the assumption that there is a backlog of exactly b units of unfinished work from jobs of equal or greater priority when J is released. (It is not a part of this decision problem to verify whether a backlog of b is actually possible.)

CHECK JOB WITH BACKLOG is in NP: It is readily verified that J is schedulable under the given assumptions if and only if there *exists* a positive solution $R \leq D_i$ to the equation

$$R \geq b + C_i + \sum_{j<i} \max\left(0, \left\lceil \frac{R - x_j}{T_j} \right\rceil C_j\right), \qquad (14)$$

where $x_j \geq 0$ is the minimum time between the release of J and the next release of a job from τ_j. Such a solution R is a witness for CHECK JOB WITH BACKLOG that can be verified in polynomial time, and therefore CHECK JOB WITH BACKLOG is in NP.

We will now use the above observation to show an upper bound of Π_2^{P} on the complexity of uniprocessor FP-schedulability. Consider a job J from some task $\tau_i \in \Gamma$ that is released at time t_2, and some time point $t_1 < t_2$. Let w denote the total workload of jobs from equal- or higher-priority tasks from Γ that are released in time interval $[t_1, t_2)$ and let $b = \max(0, w - (t_2 - t_1))$. We consider two cases based upon whether t_1 starts a *level-i busy period* [38] in the schedule.

Case 1 (t_1 is the start of a level-i busy period): Since t_1 is the start of a level-i busy-period, there can be no backlog of equal- or higher-priority jobs at t_1, and no idle time in $[t_1, t_2)$. The backlog of such jobs at t_2 is then exactly b, and $\langle J, b\rangle$ is an instance of CHECK JOB WITH BACKLOG *if and only if* job J is schedulable.

Case 2 (t_1 is *not* the start of a level-i busy period): Here there can be a backlog of equal- or higher-priority work already at t_1 and there could also be idle time in $[t_1, t_2)$. Both of these possibilities would cause b to be smaller than the actual backlog at t_2, and hence $\langle J, b\rangle$ is an instance of CHECK JOB WITH BACKLOG *if* job J is schedulable.

We can then verify *all* instances $\langle J, b\rangle$ using CHECK JOB WITH BACKLOG as an oracle, where J ranges over all jobs in the first two hyper-periods[7] and where b is calculated as above for all integer time points t_1 before the release of J. By the reasoning above, we will then have checked every job for schedulability, once exactly (when b is the exact backlog) and several more times with no false negatives (when b is less than the backlog), and hence this is a method of solving the uniprocessor version of the schedulability problem. While this is not at all a practical method, it does show that uniprocessor FP-schedulability is in Π_2^{P} since a single instance $\langle J, b\rangle$ serves as a potential counterexample that can be checked with the oracle. □

An upper bound for the partitioned version follows easily:

Theorem 7. *Partitioned FP-schedulability for asynchronous periodic tasks with arbitrary deadlines on m heterogeneous processors is in* Σ_3^{P}.

Proof. A partitioning $\Gamma_1, \ldots, \Gamma_m$ of Γ is a witness to partitioned FP-schedulability that can be verified using an oracle for the uniprocessor version. Since the oracle is in Π_2^{P} by Lemma 6.1, the considered problem is in Σ_3^{P}. □

We can look at the fact that partitioned FP-schedulability for asynchronous periodic tasks with arbitrary deadlines is in Σ_3^{P} in two ways. We can think of it as the partitioned version being solvable using an oracle for the uniprocessor version, which in turn uses an oracle for CHECK JOB WITH BACKLOG. But we could also think about it in a different way: that the partitioned version can be posed as a question of the form "$\exists\,\forall\,\exists$", namely, does there *exist* a partitioning, such that *for all* pairs $\langle J, b\rangle$ with those partitions, there *exists* a solution to Expression 14?

[7]Note that for task systems that are not over-utilized, the first two hyper-periods will contain all unique busy periods.

VII. A Map of the Complexity Landscape

In this section we integrate our results into a comprehensive view of the complexity landscape of partitioned schedulability problems — see Figure 2. Recall that all our upper bounds are valid for unrelated heterogeneous processors and all our lower bounds for identical processors; when the bounds match, we have pinpointed the precise complexity of the partitioning problem upon both identical and heterogeneous platforms. In Figure 2, lighter cells are problems that can be represented as ILPs in polynomial time (i.e., they are in NP) and darker cells are problems that cannot (they are not in NP unless the polynomial hierarchy collapses). For the dual-shaded cells, it is currently unknown if they can be so represented. In the following we use the numbers in the top-left corners of the cells (numbered 1–24) to refer to the problems.

Synchronous periodic and sporadic tasks. The scheduling problem is strongly NP-complete for partitioned EDF with implicit deadlines (7 and 10), and can readily be formulated as an ILP as described in Section IV-A. The well-known reduction from BIN PACKING shows that 7 is indeed NP-hard in the *strong* sense, but does not show the same for 10 since BIN PACKING is only strongly NP-complete for arbitrary number of bins. However, recent results [32] on PARTITION with rational weights are easily applied to show that also 10 is NP-hard in the strong sense, even for $m = 2$.

On the other hand, the FP-schedulability problem is NP-complete also for constrained deadlines, and 1, 2, 4 and 5 can all be solved by the ILP in Section IV-B. However, it is not immediate that the results of [32] apply here, so it is unknown if 4 and 5 are NP-hard in the strong sense.

For EDF with constrained or arbitrary deadlines, Theorem 1 tells us that schedulability is both NP- and coNP-hard, so 8, 9, 11 and 12 cannot be formulated as ILPs in polynomial time unless NP = coNP. They are all contained in Σ_2^{P} by Theorem 6, but it is unknown is they are hard for this class.

A much larger gap exists for FP with arbitrary deadlines, the lower bounds for 3 and 6 are just the same as the variants with implicit or constrained deadlines, but the best upper bound we have is Σ_3^{P} by Theorem 7. For 3 and 6 it is currently unknown if they can be formulated as ILPs in polynomial time.

Asynchronous periodic tasks. Here we have a complete classification for all the considered EDF-schedulability problems. The cases with implicit deadlines, 19 and 22, are strongly NP-complete and can be solved by the ILP in Section IV-A. For constrained and arbitrary deadlines, we have pinpointed the complexity at the second level of the polynomial hierarchy, so 20, 21, 23 and 24 are all Σ_2^{P}-complete by Theorems 4 and 6 (already when $m = 2$). These problems cannot be formulated as ILPs in polynomial time unless the polynomial hierarchy collapses to NP.

The FP-schedulability problems with constrained or arbitrary deadlines are at least as hard as with EDF: 14, 15, 17 and 18 are also Σ_2^{P}-hard by Theorem 4 (indeed, so are the schedulability problems for *all* work-conserving partitioned schedulers by that theorem), but for FP there is a gap to the upper bound of Σ_3^{P} from Theorem 7.

For implicit deadlines, the schedulability problem for FP is much harder than for EDF if the priority ordering is specified: 13 and 16 are Σ_2^{P}-hard as well by Theorem 5. However, if the priority ordering can be chosen freely we only have the same lower bounds as for synchronous tasks, and it is unknown if 13 and 16 can be formulated as ILPs in polynomial time in this case. The best known upper bound is Σ_3^{P} in either case.

VIII. Conclusions

Most task schedulability problems considered in the real-time systems literature are NP- or coNP-hard, but not all are *in* either of those classes. To understand the relative complexity of schedulability problems—to see why some are seemingly much harder than others—we would often have to look beyond NP and coNP.

In this work we have considered the complexity of partitioned schedulability problems for different types of recurrent tasks. In some settings, such as FP-scheduling of constrained-deadline sporadic tasks, the partitioned version of the schedulability problem remains in the same complexity class as the corresponding uniprocessor version (both are NP-complete). This partitioned schedulability problem can be solved by the ILP described in Sec. IV-B.

For other settings, such as the equivalent EDF-schedulability problem, the partitioned version is provably beyond the class of its corresponding uniprocessor version (under the widely held conjecture of complexity theory that the polynomial hierarchy does not collapse to NP).

We have shown that several of the basic partitioned schedulability problems are similarly of a qualitatively different sort than their uniprocessor variants. Some are exactly pinpointed in the polynomial hierarchy, for others gaps still remain between the best known upper and lower bounds. We are hopeful that the remaining gaps can be closed, perhaps with a starting point in the useful PARTITIONED SCP problem that is shown to be Σ_2^{P}-complete in this paper.

Other well-known schedulability problems could be beyond the polynomial hierarchy altogether. For example, the best known upper bound to most basic *global* schedulability problems put them in PSPACE or even double-exponential time [39], but the best known lower bounds for most of these are just the same as for the uniprocessor variants, leaving very large gaps.[8] Further exploring the complexity of all these common schedulability problems is an important step if we want to truly understand them.

[8]Some exact complexity results are in fact known for global scheduling: Global feasibility for implicit-deadline tasks is (quite remarkably!) in P [40] and global schedulability for arbitrary-deadline tasks where the *scheduler is specified as a part of the input* was shown to be PSPACE-complete by Geeraerts et al. [41], which puts it outside the entire polynomial hierarchy unless the hierarchy collapses.

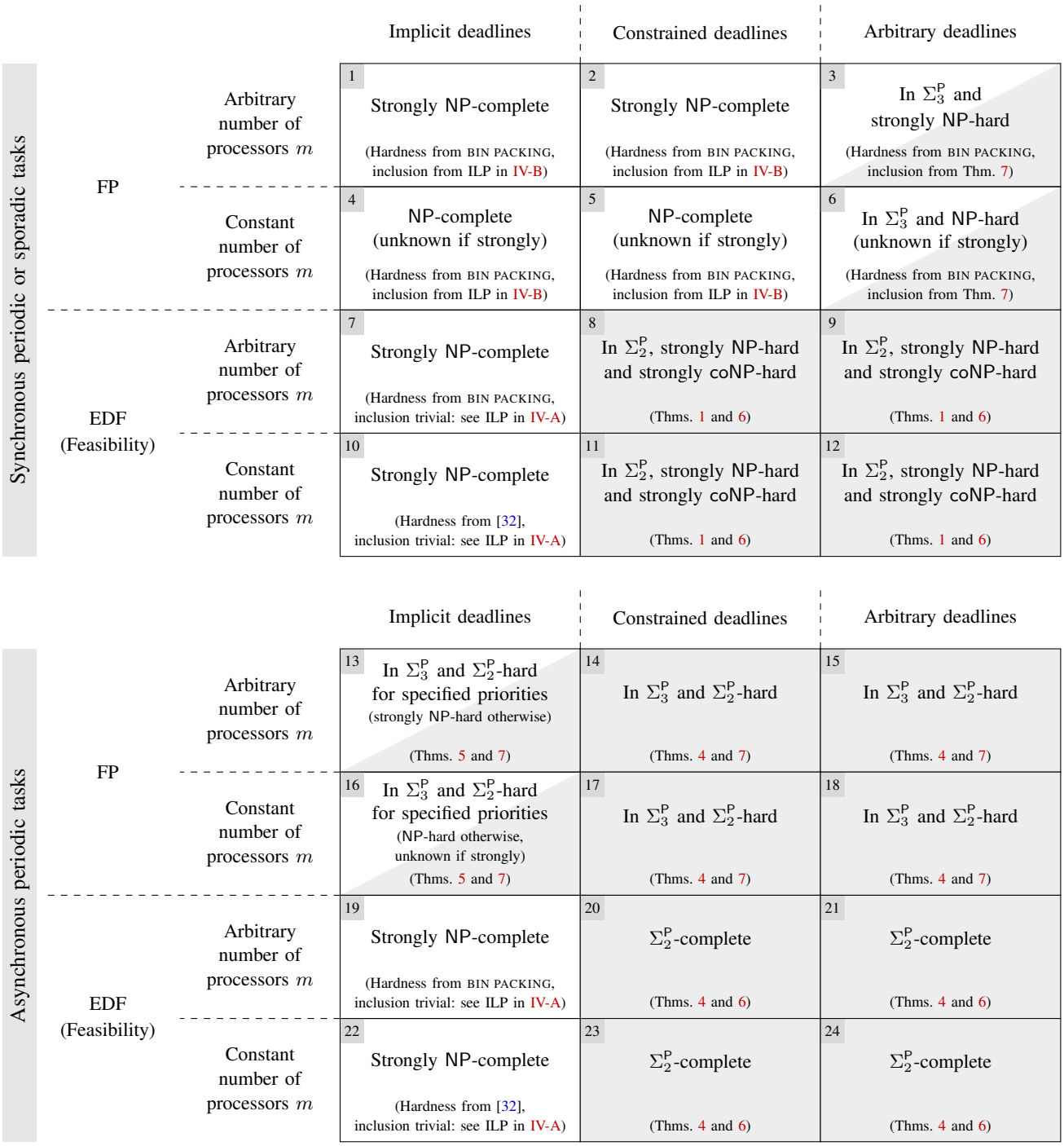

Fig. 2. Complexity of partitioned multiprocessor schedulability problems. Lighter cells have efficient ILP representations; darker cells are unlikely to. It is currently unknown whether the dual-shaded cells (3 , 6 , 13 and 16) have such ILP representations or not.

ACKNOWLEDGMENT

We are thankful to the anonymous reviewers for pointing us in a direction that led to the prior work by Zheng et al. [26], which already contained the key ideas for the ILP in Sec. IV-B.

This work was funded in part by the US National Science Foundation under Grants CNS-1814739 and CPS-1932530, and by the Swedish Research Council under Grant 2018-04446.

REFERENCES

[1] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and company, NY, 1979.

[2] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 25–34, Stockholm, Sweden, June 2000. IEEE Computer Society Press.

[3] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor rate-monotonic scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):642–653, 2004.

[4] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68, 2004.

[5] Sanjoy Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 321–329, Miami, Florida, December 2005. IEEE Computer Society Press.

[6] Nathan Fisher, Sanjoy Baruah, and Theodore Baker. The partitioned scheduling of sporadic tasks according to static priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 118–127, Dresden, Germany, July 2006. IEEE Computer Society Press.

[7] Nathan Fisher. *The Multiprocessor Real-Time Scheduling of General Task Systems*. PhD thesis, Department of Computer Science, The University of North Carolina at Chapel Hill, 2007.

[8] Sanjoy Baruah. Partitioned EDF scheduling: A closer look. *Real-Time Systems: The International Journal of Time-Critical Computing*, 49(6):715–729, 2013.

[9] Bipasa Chattopadhyay and Sanjoy Baruah. A lookup-table driven approach to partitioned scheduling. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, Chicago, 2011. IEEE Computer Society Press.

[10] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[11] Friedrich Eisenbrand and Thomas Rothvoß. Static-priority real-time scheduling: Response time computation is NP-hard. In *Proceedings of the Real-Time Systems Symposium*, Barcelona, December 2008. IEEE Computer Society Press.

[12] Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2010.

[13] P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 281–286, 2015.

[14] P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization. In *2015 IEEE Real-Time Systems Symposium*, pages 87–95, 2015.

[15] Pontus Ekberg and Wang Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 139–146. IEEE Computer Society, 2017.

[16] Pontus Ekberg. Rate-monotonic schedulability of implicit-deadline tasks is NP-hard beyond Liu and Layland's bound. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 308–318. IEEE, 2020.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[18] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[19] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1976.

[20] Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News*, 2002.

[21] Joseph Y.-T Leung and M. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[22] Joseph Y.-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[23] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

[24] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[25] I. Borosh and L. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55:299–304, 1976.

[26] Wei Zheng, Qi Zhu, Marco Di Natale, and Alberto Sangiovanni Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 161–170, 2007.

[27] Sanjoy Baruah and Enrico Bini. Partitioned scheduling of sporadic task systems: an ILP-based approach. In *Proceedings of the 2008 Conference on Design and Architectures for Signal and Image Processing*, 2008.

[28] Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne van der Ster, and Andreas Wiese. Assigning sporadic tasks to unrelated parallel machines. In *ICALP (1)*, pages 665–676, 2012.

[29] Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne van der Ster, and Andreas Wiese. Assigning sporadic tasks to unrelated parallel machines. *Mathematical Programming*, 152:247–274, 2015.

[30] Sanjoy Baruah, Vincenzo Bonifaci, Renato Bruni, and Alberto Marchetti-Spaccamela. ILP-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors. In *Proceedings of the 2016 28th EuroMicro Conference on Real-Time Systems*, ECRTS '16, Toulouse (France), 2016. IEEE Computer Society Press.

[31] Sanjoy K. Baruah, Vincenzo Bonifaci, Renato Bruni, and Alberto Marchetti-Spaccamela. ILP models for the allocation of recurrent workloads upon heterogeneous multiprocessors. *Journal of Scheduling*, Dec 2018.

[32] Dominik Wojtczak. On strong NP-completeness of rational problems. In *Proceedings of the 13th International Computer Science Symposium in Russia (CSR)*, June 2018.

[33] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, 1993.

[34] S. Baruah, R. Howell, and L. Rosier. On preemptive scheduling of periodic, real-time tasks on one processor. In *Fifteenth International Symposium on the Mathematical Foundations of Computer Science*, volume 2, pages 173–179, Bansk'a Bystrica, Czechoslovakia, 1990. Springer-Verlar.

[35] Vladislav Rutenburg. Complexity of generalized graph coloring. In Jozef Gruska, Branislav Rovan, and Juraj Wiedermann, editors, *Mathematical Foundations of Computer Science 1986*, pages 573–581, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[36] D. Knuth. *The Art of Computer Programming: Vol. 2, Seminumerical Algorithms*. Addison-Wesley, 1973.

[37] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[38] J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[39] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63(4):763–780, Aug 2012.

[40] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[41] Gilles Geeraerts, Joël Goossens, and Markus Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 49(2):171–218, Mar 2013.