On the Feasibility of Sporadic Tasks with Restricted Parallelism on Heterogeneous Multiprocessors

Denver Massey, Shareef Ahmed, and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill
USA
{denmas22,shareef,anderson}@cs.unc.edu

ABSTRACT

Given a system of real-time tasks, a fundamental question is whether it can be feasibly scheduled on a specific hardware platform. Prior work has provided feasibility tests for the common sporadic task model on different types of multiprocessors. These results, until now, have not been extended to the more generalized *restricted parallelism sporadic*, or *rp-sporadic*, task model. Under this model, jobs of the same task may execute at the same time. Due to its increased flexibility, this model may prove beneficial for applications able to leverage such parallelism. This work presents feasibility tests for rp-sporadic tasks with three common multiprocessor specifications: Identical with Affinity Masks, Uniform, and Unrelated, extending prior analysis to this more generalized model and laying a foundation for its future use. These theoretical contributions are accompanied by a large-scale experimental evaluation of 189 different system configurations.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**.

KEYWORDS

heterogeneous multiprocessors, concurrency, soft real-time

ACM Reference Format:
Denver Massey, Shareef Ahmed, and James H. Anderson. 2024. On the Feasibility of Sporadic Tasks with Restricted Parallelism on Heterogeneous Multiprocessors. In *32nd ACM International Conference on Real-Time Networks and Systems (RTNS '24), November 6–8, 2024, Porto, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3696355.3699709

ACKNOWLEDGMENTS

Work supported by NSF grants CPS 2038960, CPS 2038855, CNS 2151829, and CPS 2333120.

1 INTRODUCTION

Advances in the computational capabilities of hardware have enabled the deployment of ever more sophisticated software in safety-critical real-time systems. Autonomous vehicles, for example, are deploying algorithms capable of spotting human figures in cluttered images or planning routes through quickly changing traffic [2, 8]. In ensuring system safety, it is paramount that the timeliness of program execution can be mathematically guaranteed. Such guarantees must be based on proofs that consider both the characteristics of the software workload and the capabilities of the underlying hardware platform.

Most real-time analysis pertains to workloads modeled as a collection of recurring tasks, where successive invocations of the same task are prevented from running in parallel with each other. In this way, one could model a periodically invoked computer-vision classifier for a self-driving car. In the standard workload model, every instance of a task must wait for prior instances of that task to complete before it is allowed to run. While this assumption is appropriate for tasks that must necessarily be invoked sequentially, it may be unnecessarily restrictive in other contexts. For example, in the aforementioned computer-vision application, if each frame of video can be processed independently, then there is no reason why successive frames cannot be processed at the same time. Indeed, such parallelism could greatly lessen processing latencies. Even if some frame-to-frame dependencies exist, it still might be desirable to allow some parallelism. For example, if the processing of one video frame takes longer than normal, it might be beneficial to process the next frame using older data from prior frames. Whether this is reasonable would be an application design decision, and specifications such as the OpenVX standard allow for a wide variety of dependencies to exist [2, 11]. These observations motivate introducing *intra-task parallelism*.

With the development of powerful hardware accelerators, such as graphics processing units (GPUs), platforms are also becoming increasingly heterogeneous. It is of principle importance, therefore, to employ mathematical tools that support this kind of heterogeneity. Current methods do not, however, account for both intra-task parallelism and multiprocessor heterogeneity. As such, there is a deficiency in the theoretical foundation necessary for implementing parallelizable software stacks in real-time systems. This work remedies this and begins developing a framework for future analysis of these types of systems.

Related work. Prior work partially accounted for intra-task parallelism by considering the *npc-sporadic* (no precedence constraints) task model [9, 18]. Under this model, each instance of a task is independent of the others, meaning it does not have to wait for any preceding instances to complete. Later work then generalized this notion to allow for varying degrees of parallelism using the *rp-sporadic* (restricted parallelism) task model [2]. This generalization will be discussed in detail in the remainder of this paper.

Additionally, feasibility conditions were originally presented for the standard sporadic task model on Identical multiprocessors [14].

RTNS '24, November 6–8, 2024, Porto, Portugal

ACM ISBN 979-8-4007-1724-6/24/11
https://doi.org/10.1145/3696355.3699709

Similar conditions were found for heterogeneous multiprocessor types, such as Uniform multiprocessors [10], Identical multiprocessors with Affinity Masks [5, 17], and Unrelated multiprocessors [4, 6]. It follows from results in [15] that the essence of feasibility comes from determining whether or not a system is over-utilized; however, it is not so straightforward to define "overutilization" on these heterogeneous systems.

Contributions. This work extends known feasibility results to the rp-sporadic model, enabling multiple jobs of the same task to run in parallel. To accomplish this, we introduce a novel method to analytically transform a set of tasks that allow intra-task parallelism to a set of tasks that do not. Using the associated task sets derived from this method, necessary and sufficient conditions for feasibility are proven for several common multiprocessor models.

Furthermore, we conducted a large-scale experimental evaluation of randomly generated systems to study how much hardware capacity is forfeited when multiprocessors are modeled in an overly simplistic manner. We paired these results with data showing how intra-task parallelism was able to recover some of this lost capacity. These experiments yielded over 500 graphs which emphasize both the importance of accurately modeling systems and the benefits of the increased flexibility offered by the rp-sporadic model.

Organization. The remainder of this paper provides notation and background information (Sec. 2), discusses modeling intra-task parallelism (Sec. 3), introduces the transformation technique (Sec. 4), presents generalized feasibility conditions (Sec. 5), presents the experimental evaluation (Sec. 6), and concludes (Sec. 7). The appendices provide server abstractions for sporadic tasks (**??**) and experimental results (**??**).[1]

2 BACKGROUND

In this section, we formally define the rp-sporadic task model (restricted parallelism) [2] and provide necessary background on feasibility. Table 1 summarizes this notation.

We consider a system of *rp-sporadic* tasks. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n tasks to be scheduled on a multiprocessor platform $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ with m processors. The speed of τ_i on π_j is given by $s_{i,j} \geq 0$. Each task τ_i releases a potentially infinite sequence of *jobs*. An rp-sporadic task is represented as $\tau_i = (C_i, T_i, p_i)$, where $C_i > 0$ is τ_i's *worst-case execution requirement* (WCER)[2] relative to an execution speed of 1.0, $T_i > 0$ is τ_i's *period*, representing the minimum separation between any two of its job releases, and $1 \leq p_i \leq m$ is τ_i's *parallelism level*, representing how many jobs of τ_i can execute in parallel, as will be discussed in Sec. 3. Tasks are assumed to have implicit deadlines, meaning the deadline of each task is T_i time units after its release. Deadline misses are allowed and, as will be shown, are unavoidable in certain circumstances. When $s_{i,j} > 0$, any job of τ_i that executes for its WCER and runs uninterrupted on π_j completes in $C_i/s_{i,j}$ time units. Otherwise, $s_{i,j} = 0$, and τ_i cannot run on π_j. We assume that C_i and T_i are both positive, rational numbers and that p_i is an integer. The *utilization* of τ_i is denoted $u_i = C_i/T_i$. For a subset of tasks $\tau' \subseteq \tau$, we let $\mathcal{U}_{\tau'} = \sum_{\tau_i \in \tau'} u_i$ denote the *total utilization* of τ'. We similarly define $\mathcal{P}_{\tau'} = \sum_{\tau_i \in \tau'} p_i$ as the sum of all parallelism levels of tasks in τ'.

We denote the j^{th} job of task τ_i as $\tau_{i,j}$. The *release time* and *completion* (*finish*) *time* of job $\tau_{i,j}$ are denoted by $r_{i,j}$ and $f_{i,j}$, respectively. The execution requirement of $\tau_{i,j}$ is $C_{i,j} \leq C_i$. The *response time* of a job is the time between when it is released and when it completes. The *deadline* of a job is given by $r_{i,j} + T_i$.

At time t, a job $\tau_{i,j}$ can be either *unreleased*, if $t < r_{i,j}$, *pending*, if $r_{i,j} \leq t < f_{i,j}$, or *complete*, if $t \geq f_{i,j}$. A job is *ready* at time t if it is eligible for being scheduled at time t. Note that a pending job may or may not be ready, depending on the status of preceding jobs and the task's parallelism level. We give a more precise definition of readiness in Sec. 3. Lastly, we assume that jobs of the same task are scheduled in *First-In-First-Out* (FIFO) order based on their release times, i.e., if multiple jobs of the same task are ready to be scheduled at the same time, the one with the earliest release time is chosen. Time is assumed to be continuous.

Other task models. An *ordinary sporadic* task system allows for no intra-task parallelism, i.e., $p_i = 1$ for all τ_i, and a job can begin execution only after the prior job from its task completes. In contrast, an *npc-sporadic* task system allows unrestricted parallelism, i.e.,

Table 1: Summary of Notation.

Symbol	Meaning
τ	Task system
τ_i	i^{th} task of τ
$\tau_{i,j}$	j^{th} job of τ_i
τ'	Subset of τ
n	Number of tasks
π	Multiprocessor platform
π_j	j^{th} processor of π
m	Number of processors
$s_{i,j}$	Speed of τ_i on π_j
C_i	WCER of τ_i
$C_{i,j}$	Execution cost of $\tau_{i,j}$
T_i	Period of τ_i
p_i	Parallelism level of τ_i
$r_{i,j}$	Release time of $\tau_{i,j}$
$f_{i,j}$	Completion (finish) time of $\tau_{i,j}$
u_i	Utilization of τ_i; C_i/T_i
$\mathcal{U}_{\tau'}$	Total utilization of τ'; $\sum_{\tau_i \in \tau'} u_i$
$\mathcal{P}_{\tau'}$	Total of parallelism levels of τ'; $\sum_{\tau_i \in \tau'} p_i$
$\hat{\tau}$	Split task set
$\hat{\tau}_i^k$	k^{th} split task of τ_i
$\hat{\tau}_{i,j}^k$	j^{th} job of $\hat{\tau}_i^k$
$\hat{\tau}'$	Subset of $\hat{\tau}$
H	Hyperperiod for a task set; $\text{lcm}_{\tau_i \in \tau}\{T_i\}$
LP-Feas(τ, π)	Linear program feasibility test for τ
LP-Feas$(\hat{\tau}, \pi)$	Linear program feasibility test for $\hat{\tau}$
$x_{i,j}$, $x_{i,j}^k$, and ℓ	Variables for linear programs
α_i	Affinity mask for τ_i; $\alpha_i \subseteq \pi$
$\alpha_{\tau'}$	Aggregate affinity mask for τ'; $\bigcup_{\tau_i \in \tau'} \alpha_i$
split(τ')	Subset of $\hat{\tau}$ containing all split tasks of τ'

[1] Appendix available online: https://jamesanderson.web.unc.edu/papers/rtns24-long.pdf.

[2] This quantity is usually called the worst-case execution *time* (WCET) in the literature; however, this name is less appropriate when processors have different speeds [16].

$p_i = m$ for all τ_i, and a job can be scheduled concurrently with any other job from its task, limited only by the number of processors. These are both special cases of the rp-sporadic task model.

Feasibility. The feasibility of a task system is contingent upon the timing constraints specified for the system. Hard real-time (HRT) constraints require each job to complete before its deadline. Soft real-time (SRT) constraints, in our context, require that some finite bound exists for the response times of the system. We say that (τ, π) is *HRT-feasible* (resp., *SRT-feasible*) if, for any valid release pattern of jobs of τ, there exists a schedule for the jobs of τ on π where each job meets its HRT (resp., SRT) timing constraints. Otherwise, (τ, π) is *infeasible*. We assume that there are zero overheads for preemptions or migrations in the system; however, such factors can be accounted for by inflating job costs [7].

Multiprocessor models. We consider the following multiprocessor platform models: Identical, Identical with Affinity Masks, Uniform, and Unrelated. The Unrelated multiprocessor model generalizes the other three models and allows each $s_{i,j}$ to be different. If $s_{i,j} = 1$ for all $\tau_i \in \tau$ and $\pi_j \in \pi$, we say π is an *Identical* multiprocessor, i.e., all processors run at the same speed, normalized to one. If speeds can be either zero or one, we say π is *Identical with Affinity Masks*, where τ_i has *affinity* for π_j if $s_{i,j} \neq 0$. If, for each $\pi_j \in \pi$ there exists s_j so that $\forall \tau_i, s_{i,j} = s_j$, we say π is a *Uniform* multiprocessor, i.e., speeds may vary between different processors, but any given processor will run all jobs at the same speed. The feasibility condition for Identical has already been established and is included below.

Theorem 2.1 (from [2]). *The system (τ, π), where π is an Identical multiprocessor, is feasible if and only if for all τ_i, $u_i \leq p_i$ and $\mathcal{U}_\tau \leq m$.*

3 INTRA-TASK PARALLELISM

In this section, we discuss how intra-task parallelism affects the scheduling of real-time tasks.

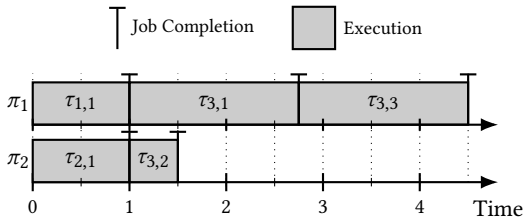
Intra-task parallelism vs. job-level dependencies. For τ_i, p_i specifies the maximum number of jobs that can be scheduled concurrently at any time and is therefore used when determining if jobs are ready to be scheduled. For example, consider some τ_i with $p_i = 3$. Job $\tau_{i,10}$ will be prevented from executing despite being pending until at least seven prior jobs of τ_i complete execution. There are different ways of enforcing this in a system, and prior work did not consider the conflicting ways of interpreting this value. We will discuss two such interpretations, which we refer to as **(A)** and **(B)**.

Definition 3.1. **(A)**: $\tau_{i,j}$ is ready at time t if $\tau_{i,j}$ is pending at time t and if $j \leq p_i$ or job $\tau_{i,j-p_i}$ is complete at time t. ◁

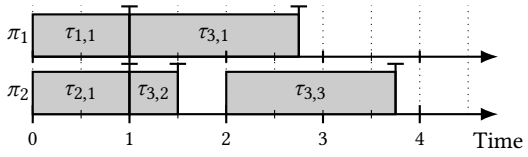
Definition 3.2. **(B)**: $\tau_{i,j}$ is ready at time t if $\tau_{i,j}$ is pending at time t and if $\left|\{\tau_{i,k} : 1 \leq k \leq j :: \tau_{i,k} \text{ is pending at time } t\}\right| \leq p_i$. ◁

When $p_i = 1$, these definitions collapse into the ready condition for the ordinary sporadic task model ($\tau_{i,j}$ is ready at time t if $\tau_{i,j}$ is pending at time t and $j = 1$ or $\tau_{i,j-1}$ is complete at time t).

Interpretation **(A)** establishes explicit dependencies between jobs of a task. This is motivated by frameworks such as OpenVX that specify data dependencies between instances of a computer-vision task [2, 11]. Such strict dependencies might be too restrictive for certain applications. Interpretation **(B)** allows for more flexibility.



(a) A schedule possible with either interpretation (A) or (B).



(b) A schedule possible with interpretation (B) but not (A).

Figure 1: Example 3.3 demonstrates conflicting decisions made when scheduling under interpretations (A) and (B).

If $\tau_{i,j-p_i}$ has not yet completed, there might be another job that has provided data which $\tau_{i,j}$ can use. Consider again the job $\tau_{i,10}$ with $p_i = 3$. According to interpretation **(A)**, this job must wait for the completion of $\tau_{i,7}$. Alternatively, under interpretation **(B)**, $\tau_{i,10}$ is ready to execute once any seven prior jobs have completed.

Moreover, if a task has a parallelism level of $p_i = m$, then we would like it to function as an npc-sporadic task. Interpretation **(B)** allows this. However, under interpretation **(A)**, it is possible to have m jobs pending with fewer than m jobs ready at a time t.

Example 3.3. Consider the task set $\tau = \{\tau_1 = (1, 20, 1), \tau_2 = (1, 20, 1), \tau_3 = (1.75, 1, 2)\}$. Let τ_1 and τ_2 each release a job at time 0 with execution requirements equal to one unit. Suppose τ_3 releases three jobs at times $r_{3,1} = 0$, $r_{3,2} = 1$, $r_{3,3} = 2$ with execution requirements $C_{3,1} = 1.75$, $C_{3,2} = 0.5$, $C_{3,3} = 1.75$. When scheduling these jobs on an Identical multiprocessor with $m = 2$, interpretations **(A)** and **(B)** cause different schedules, as illustrated in Figure 1.

Specifically, at time $t = 2$, $\tau_{3,3}$ is ready under **(B)** but not under **(A)**. This allows $\tau_{3,3}$ to complete 0.75 time units sooner in the second schedule, as seen in Figure 1(b). Note that similar schedules exist that allow some jobs to finish sooner under **(A)** than under **(B)**. ◁

Example 3.3 also illustrates two important aspects of rp-sporadic tasks. First, the rp-sporadic model supports tasks with utilizations greater than the speed of any processor. Second, in order to make use of parallelism, deadline misses must be allowed for implicit-deadline tasks. Otherwise, no task will ever have more than one ready job at a time. The remainder of this paper uses the terms feasibility and SRT-feasibility interchangeably.

Even though interpretations **(A)** and **(B)** can prompt different scheduling decisions, the following lemma shows that they are equivalent (in a feasibility sense) when every job runs for its task's WCER, assuming Identical multiprocessors.

Lemma 3.4. *If every job of τ executes for its WCER, then (τ, π) is feasible under **(A)** if and only if (τ, π) is feasible under **(B)**, when π is an Identical multiprocessor.*

Proof. $A \Rightarrow B$. Let S be a feasible schedule of (τ, π) where **(A)** is used to determine each job's readiness and all jobs execute for their WCERs. We will show that S is also a feasible schedule of (τ, π) under **(B)**. We will show that at all time instants t, any scheduled job $\tau_{i,j}$ in S must also be ready under interpretation **(B)**. Let $\tau_{i,j}$ be scheduled (hence, ready) in S at time t. We have two cases.

Case AB.1. $j \leq p_i$. $\tau_{i,j}$ is one of the first p_i jobs released by τ_i. Therefore, the set $\{\tau_{i,k} : 1 \leq k \leq j :: \tau_{i,k} \text{ is pending at time } t\}$ consists of at most p_i jobs. Thus, $\tau_{i,j}$ is ready under **(B)**.

Case AB.2. $j > p_i$. $\tau_{i,j}$ is ready under **(A)**, so $\tau_{i,j-p_i}$ is complete at time t in S. In this case, every job of τ_i released prior to $\tau_{i,j-p_i}$ must also be complete at time t in S. Otherwise, $\tau_{i,j-p_i}$ must have begun execution earlier than one of those jobs, as all jobs must execute for their WCERs, and all jobs take the same amount of time to execute on an Identical multiprocessor. This would violate the FIFO scheduling requirement of jobs of the same task. Thus, there are at most $j - (j - p_i) = p_i$ jobs in the set $\{\tau_{i,k} : 1 \leq k \leq j :: \tau_{i,k} \text{ is pending at time } t\}$, and $\tau_{i,j}$ is ready under **(B)**.

$B \Rightarrow A$. Let S be a feasible schedule for the system (τ, π) when **(B)** is used to determine each job's readiness and all jobs execute for their WCERs. We will show that S is also a feasible schedule under **(A)**. Let $\tau_{i,j}$ be scheduled (hence, ready) in S at time t. We now show that $\tau_{i,j}$ is ready under **(A)**. We have two cases.

Case BA.1. $j \leq p_i$. $\tau_{i,j}$ is ready under **(A)**.

Case BA.2. $j > p_i$. We will show that $\tau_{i,j-p_i}$ completes execution by time t. Assume otherwise. Then, the jobs $\tau_{i,j-p_i}, \tau_{i,j-p_i+1}, \ldots, \tau_{i,j}$ must all be pending in S at time t, as all jobs of τ_i execute for their WCER in FIFO order on an Identical multiprocessor. This is a contradiction, as $|\{\tau_{i,k} : 1 \leq k \leq j :: \tau_{i,k} \text{ is pending at time } t\}| \leq p_i$. Consequently, $\tau_{i,j}$ is ready under **(A)**. □

The following lemma strengthens Lemma 3.4 by removing the assumption that every job executes for its WCER.

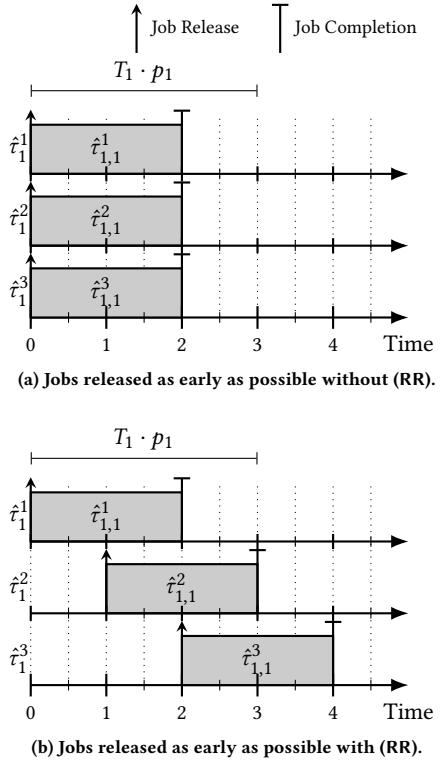
Lemma 3.5. *(τ, π) is feasible under **(A)** if and only if (τ, π) is feasible under **(B)**, when π is an Identical multiprocessor.*

Proof. Suppose (τ, π) is feasible under **(A)**. For any release pattern of jobs, there exists a feasible schedule under **(A)** where each job executes for its WCER. By Lemma 3.4, this is also a feasible schedule under **(B)** when each job executes for its WCER. For any job that executes for less than its WCER, idleness can be inserted into this schedule in place of the missing execution without affecting feasibility. Therefore, (τ, π) is also feasible under **(B)**. By a similar argument, the converse follows. □

Thus, interpretations **(A)** and **(B)** are equivalent in a feasibility sense, as long as tasks are running on an Identical multiprocessor. We defer further investigation on the impact of these interpretations on more general multiprocessor models as a future work due to space constraints. In the remainder of the paper, we will consider job readiness only according to interpretation **(A)**.

4 SPLITTING TASKS

In this section, we introduce an analytical technique for transforming a system of rp-sporadic tasks into a system of ordinary sporadic tasks. This is motivated by the precedence constraints imposed by **(A)** and will allow us to leverage existing results for sporadic tasks. We begin by defining the notion of a *split task*.



(a) Jobs released as early as possible without (RR).

(b) Jobs released as early as possible with (RR).

Figure 2: Possible scenario from Example 4.2 executing at its WCER on three Identical processors with and without (RR). The jobs in (b) do obey (RR), so the first jobs of $\hat{\tau}_1^2$ and $\hat{\tau}_1^3$ have their releases postponed by T_i and $2T_i$, respectively.

Definition 4.1. Given an rp-sporadic task set τ, we define the set of *split tasks* $\hat{\tau}$ for τ: For each $\tau_i \in \tau$ and each $1 \leq k \leq p_i$, include the *split task* $\hat{\tau}_i^k = (C_i, T_i \cdot p_i, 1)$. Additionally, the tasks of $\hat{\tau}$ must obey the release offsets specified by **(RR)**, given below. ◂

We denote the j^{th} job of task $\hat{\tau}_i^k$ as $\hat{\tau}_{i,j}^k$. Let $r_{i,j}^k$ denote the release time of $\hat{\tau}_{i,j}^k$. The *round-robin property*, denoted **(RR)**, is as follows: For all $1 \leq i \leq n$ and $j \geq 1$,

$$\left.\begin{aligned} & r_{i,1}^1 \geq 0 \\ & r_{i,j}^1 \geq r_{i,j-1}^{p_i} + T_i, \quad \text{for } 1 < j \\ & r_{i,j}^k \geq r_{i,j}^{k-1} + T_i, \quad \text{for } 1 < k \leq p_i \text{ and } 1 \leq j. \end{aligned}\right\} \quad \textbf{(RR)}$$

This property requires each split task to wait at least T_i time units after its preceding split task releases a job to release its own and allows us to associate any schedule for $\hat{\tau}$ with a corresponding schedule for τ. Without **(RR)**, $\hat{\tau}$ can release up to $\mathcal{P}_\tau$ jobs at once, while τ can release at most n jobs at once (see Figure 2). Therefore, these release offsets force job releases in a *round-robin* fashion.

Example 4.2. Consider a task $\tau_1 = (2, 1, 3)$. By Def. 4.1, there are three split tasks corresponding to τ_1. Each split task has parameters $(2, 3, 1)$. Figure 2 shows potential job releases of these split tasks as ordinary sporadic tasks. In Figure 2(a), job releases do not obey **(RR)**, but in Figure 2(b), they do. Note that τ_1 could never have the job releases shown in Figure 2(a). ◁

As will be shown in Lemma 4.4, the release offsets from **(RR)** allow for a straightforward association between schedules of τ and its related set of split tasks $\hat{\tau}$. Example 4.3 provides a concrete system that will aid in the presentation of the proof of Lemma 4.4.

Example 4.3. Consider $\tau = \{\tau_1 = (3, 2, 3), \tau_2 = (2, 6, 2), \tau_3 = (1, 7, 1)\}$. The set $\hat{\tau}$ is given by $\hat{\tau} = \{\hat{\tau}_1^1, \hat{\tau}_1^2, \hat{\tau}_1^3, \hat{\tau}_2^1, \hat{\tau}_2^2, \hat{\tau}_3^1\}$, where $\hat{\tau}_1^1, \hat{\tau}_1^2$, and $\hat{\tau}_1^3 = (3, 6, 1)$, $\hat{\tau}_2^1$ and $\hat{\tau}_2^2 = (2, 12, 1)$, and $\hat{\tau}_3^1 = (1, 7, 1)$.

Suppose these tasks release jobs as soon as possible and each job executes for its WCER. Figure 3(a) illustrates the global earliest-deadline-first (GEDF) schedule of τ on two Identical processors. Since $m = 2$, any scheduled job must have the first or second earliest deadline of all currently ready jobs. At time 8, job $\tau_{2,2}$, with a deadline of 12, is preempted when $\tau_{1,5}$ releases with a deadline of 10. Intra-task parallelism is exhibited by jobs of τ_1. Figure 3(b) depicts a schedule for the jobs of $\hat{\tau}$ with deadlines set to $r_{i,j}^k + T_i$. ◁

Lemma 4.4. *(τ, π) is feasible if and only if $(\hat{\tau}, \pi)$ is feasible.*

Proof. We show this by associating jobs of τ with jobs of $\hat{\tau}$.

($\Rightarrow$) Suppose (τ, π) is feasible under interpretation **(A)**. Given a release pattern of jobs of $\hat{\tau}$, we relabel the j^{th} job of $\hat{\tau}_i^k$ as job $(j-1)p_i + k$ of τ_i. For example, in Figure 3, the 2^{nd} job of $\hat{\tau}_1^1$ corresponds to job 4 of τ_1. The minimum separation time between consecutive job releases of τ_i is at least T_i, as no two newly relabeled jobs will have release times within T_i time units of each other due to **(RR)**. Therefore, these job releases for $\hat{\tau}$ can be converted to valid job releases for τ. Moreover, if this job of τ_i is ready under interpretation **(A)**, then the previous job of $\hat{\tau}_i^k$ is necessarily complete, and no job of $\hat{\tau}$ gets scheduled when it is not ready. Therefore, a schedule for τ can be used as a schedule for $\hat{\tau}$.

($\Leftarrow$) Suppose $(\hat{\tau}, \pi)$ is schedulable. Given a release pattern of jobs of τ, we can relabel the jth job of τ_i as job $\lceil j/p_i \rceil$ of $\hat{\tau}_i^{(j-1 \mod p_i)+1}$. For example, in Figure 3, the 5^{th} job of τ_1 corresponds to job 2 of $\hat{\tau}_1^2$. These jobs satisfy **(RR)** because no two jobs of τ_i are ever released within T_i time units of each other. Therefore, these job releases for τ can be converted to valid job releases for $\hat{\tau}$. Moreover, a split job will only be ready when the previous job from its split task has been completed. This corresponds to job $\tau_{i,j-p_i}$, as $\lceil (j-p_i)/p_i \rceil = \lceil j/p_i \rceil - 1$ and $j - 1 - p_i \equiv j - 1 \mod p_i$, so no job of τ is scheduled when it is not ready. Therefore, a schedule for $\hat{\tau}$ can be used as a schedule for τ.

It follows that any pattern of job releases of τ corresponds to a pattern of job releases of $\hat{\tau}$, and vice versa, meaning that a feasible schedule for one is a feasible schedule for the other. □

Properties of $\hat{\tau}$. Note that $\hat{\tau}$ contains p_i split tasks for each τ_i in τ, and each $\hat{\tau}_i^k$ has utilization u_i/p_i. Therefore,

$$|\hat{\tau}| = \mathcal{P}_\tau \text{ and } \mathcal{U}_{\hat{\tau}} = \mathcal{U}_\tau. \tag{1}$$

Similarly, if $\hat{\tau}' \subseteq \hat{\tau}$ contains all split tasks of all tasks in $\tau' \subseteq \tau$, then

$$|\hat{\tau}'| = \mathcal{P}_{\tau'} \text{ and } \mathcal{U}_{\hat{\tau}'} = \mathcal{U}_{\tau'}. \tag{2}$$

5 FEASIBILITY RESULTS

This section utilizes the results proven thus far to establish feasibility conditions for the rp-sporadic task model under different heterogeneous multiprocessor platform specifications.

5.1 Unrelated

By Lemma 4.4, the feasibility condition for an rp-sporadic task set τ can be derived from that for a corresponding sporadic task set $\hat{\tau}$. This condition is presented in [4] and stated below.

Definition 5.1 (from [4]). Given a system $(\hat{\tau}, \pi)$, we define the following linear program:

$$\text{minimize } \ell$$

$$\text{subject to: } \forall \hat{\tau}_i^k, \sum_{\pi_j \in \pi} x_{i,j}^k \cdot s_{i,j} = u_i/p_i \tag{3}$$

$$\forall \hat{\tau}_i^k, \sum_{\pi_j \in \pi} x_{i,j}^k \leq \ell \tag{4}$$

$$\forall \pi_j, \sum_{\hat{\tau}_i^k \in \hat{\tau}} x_{i,j}^k \leq \ell, \tag{5}$$

with $x_{i,j}^k \geq 0$ and $\ell \geq 0$. We call this linear program *LP-Feas*$(\hat{\tau}, \pi)$. ◁

Recall that the task $\hat{\tau}_i^k$ has a utilization of u_i/p_i, so this value is substituted in for the utilizations in [4]. Each value $x_{i,j}^k$ corresponds to the fraction of time $\hat{\tau}_i^k$ spends on π_j in an ideal schedule, and the value ℓ represents the makespan, which the objective function aims to minimize [4, 6]. Next, we restate two important lemmas from [4] and argue that they apply to split task sets that obey **(RR)**.

Lemma 5.2 (Lemma 2 from [4]). *If LP-Feas$(\hat{\tau}, \pi)$ has a solution with $0 \leq \ell \leq 1$, then the system $(\hat{\tau}, \pi)$ is feasible.*

Proof. Lemma 2 from [4] and the schedule construction of [6] use a solution to *LP-Feas*$(\hat{\tau}, \pi)$ to create a schedule for the periodic job release pattern of $\hat{\tau}$ by assigning $\hat{\tau}_i^k$ to π_j for $x_{i,j}^k$ time units during $[0, \ell)$. The specifics of this construction are outside the scope of this paper, and the reader is referred to [4, 6] for complete details.

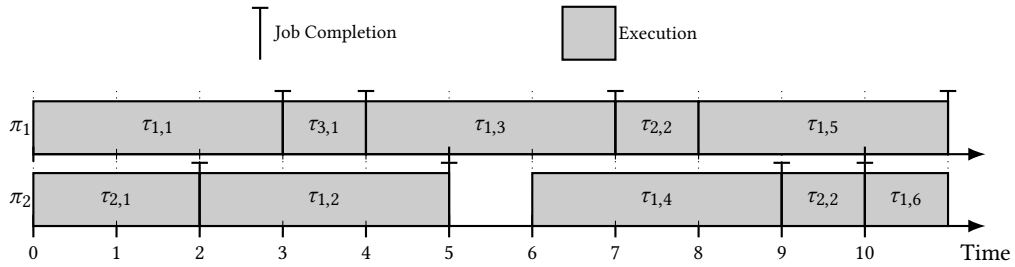
This schedule can be adapted to achieve bounded response times for the sporadic case using a server abstraction [1]. The details of this can be found in **??** due to space constraints.[3] □

Lemma 5.3 (Lemma 1 from [4]). *If $(\hat{\tau}, \pi)$ is feasible, then LP-Feas$(\hat{\tau}, \pi)$ has a solution with $0 \leq \ell \leq 1$.*

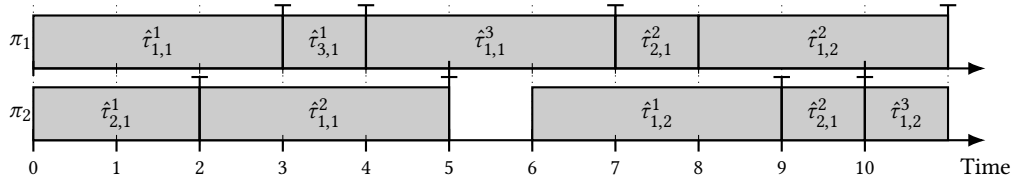
Proof. We present an addendum to the proof from [4], which uses the synchronous periodic job release pattern to solve *LP-Feas*$(\hat{\tau}, \pi)$. By **(RR)**, such a release pattern may be impossible for $\hat{\tau}$. Nevertheless, this lemma still holds.

Since we assume that $(\hat{\tau}, \pi)$ is feasible, we can choose a specific job release pattern that suffices for our proof. Suppose the tasks of $\hat{\tau}$ release jobs as soon as possible and each job executes for its WCER. By the assumption of this lemma, $(\hat{\tau}, \pi)$ is feasible, so there

[3] Appendix available online at https://jamesanderson.web.unc.edu/papers/rtns24-long.pdf



(a) GEDF schedule for rp-sporadic task set τ.



(b) GEDF schedule for normal sporadic task set $\hat{\tau}$ with deadlines set to $r^k_{i,j} + T_i$.

Figure 3: GEDF schedules for τ (top) and $\hat{\tau}$ (bottom) from Example 4.3. These jobs are scheduled on Identical processors π_1 and π_2. Note that there is a one-to-one mapping between the jobs of these two schedules, as discussed in Lemma 4.4.

exists a schedule S for these jobs such that the response time of every job is bounded by some value R, i.e., each $\hat{\tau}^k_{i,j}$ finishes by time $r^k_{i,j} + R$. Using this fact, we will derive values for the variables of *LP-Feas*$(\hat{\tau}, \pi)$. Each $x^k_{i,j}$ will end up being the fraction of time $\hat{\tau}^k_i$ spends on π_j over the course of the schedule, and we will refer to it as the *long-run execution rate* of $\hat{\tau}^k_i$ on π_j.

Let H be the *hyperperiod* of the tasks of $\hat{\tau}$, i.e., $H = lcm_{1 \le i \le n}\{T_i \cdot p_i\}$, where *lcm* denotes the least common multiple. We compute the long-run execution rate of each task on each processor by analyzing schedule S in every hyperperiod. When releasing jobs as early as possible, $\hat{\tau}^k_i$ releases exactly $H/(T_i \cdot p_i)$ jobs within $[0, H)$. Note that **(RR)** forces the release times of the first jobs of $\hat{\tau}^1_i$ and $\hat{\tau}^{p_i}_i$ to differ by $T_i(p_i - 1)$. Therefore, with $q \in \mathbb{N}$, each split task $\hat{\tau}^k_i$ releases exactly one job in any interval $[q \cdot T_i \cdot p_i, (q+1) \cdot T_i \cdot p_i)$. There are $H/(T_i \cdot p_i)$ such intervals in $[0, H)$.

When **(RR)** is enforced, the last job of $\hat{\tau}^k_i$ to be released in $[0, H)$ releases at time $H - T_i$ and can therefore complete as late as $H - T_i + R$. This fact also holds for any multiple of the hyperperiod, motivating the following definition: For times $t = qH - T_i + R$, $q \in \mathbb{N}$, let $p^k_{i,j}(t)$ denote the fraction (or proportion) of the total time over $[0, t)$ during which π_j executes jobs of $\hat{\tau}^k_i$ that are released *before* time qH. Thus, multiplying $p^k_{i,j}(qH - T_i + R)$ by $(qH - T_i + R) \cdot s_{i,j}$ yields the total completed execution of $\hat{\tau}^k_i$'s jobs that are released before time qH on processor π_j during $[0, qH - T_i + R)$. Since there are exactly $qH/(T_i \cdot p_i)$ jobs of $\hat{\tau}^k_i$ released during $[0, qH)$, each with an execution requirement of C_i, we have

$$\sum_{\pi_j \in \pi} p^k_{i,j}(qH - T_i + R) \cdot (qH - T_i + R) \cdot s_{i,j} = C_i \cdot \frac{qH}{T_i \cdot p_i}. \quad (6)$$

Dividing each side of (6) by $(qH - T_i + R)$ gives

$$\sum_{\pi_j \in \pi} p^k_{i,j}(qH - T_i + R) \cdot s_{i,j} = C_i \cdot \frac{qH}{(qH - T_i + R) \cdot T_i \cdot p_i}. \quad (7)$$

Taking the limit on both sides of (7), we determine the long-run execution rate of $\hat{\tau}^k_i$ over all processors in π:

$$\begin{aligned} \lim_{q \to \infty} \sum_{\pi_j \in \pi} & p^k_{i,j}(qH - T_i + R) \cdot s_{i,j} \\ &= \lim_{q \to \infty} C_i \cdot \frac{qH}{(qH - T_i + R) \cdot T_i \cdot p_i} \\ &= C_i \cdot \frac{H}{H \cdot T_i \cdot p_i} \\ &= u_i / p_i. \end{aligned} \quad (8)$$

We now show that all constraints in *LP-Feas*$(\hat{\tau}, \pi)$ are satisfied for a value of $0 \le \ell \le 1$ when $x^k_{i,j} = \lim_{q \to \infty} p^k_{i,j}(qH - T_i + R)$ for all i, k, and j. Constraint (3) follows from (8). Constraint (4) follows from the fact that each task can run on at most one processor at any given time instant, so the sum of these x values is at most one for a given task. Finally, constraint (5) holds as each processor cannot execute for more than t time units in the interval $[0, t)$ for any t, so the sum of these proportions adds up to at most one for each processor. Thus, if $\hat{\tau}$ is feasible, then there exists an assignment of variables of *LP-Feas*$(\hat{\tau}, \pi)$ so that $0 \le \ell \le 1$ holds. □

The next theorem follows from Lemmas 5.2 and 5.3.

Theorem 5.4. *A system $(\hat{\tau}, \pi)$ is feasible if and only if LP-Feas$(\hat{\tau}, \pi)$ has a solution with $0 \le \ell \le 1$.*

Theorem 5.4 gives a feasibility condition for sporadic split task set $\hat{\tau}$. Using Lemma 4.4 and Theorem 5.4, we now give a feasibility condition for the rp-sporadic task set τ.

Theorem 5.5. *The system (τ, π) is feasible if and only if the following linear program has a solution with $0 \le \ell \le 1$:*

$$\begin{aligned} &\textit{minimize } \ell \\ &\textit{subject to: } \forall \tau_i, \sum_{\pi_j \in \pi} x_{i,j} \cdot s_{i,j} = u_i && (9) \\ &\forall \tau_i, \sum_{\pi_j \in \pi} x_{i,j} \le \ell \cdot p_i && (10) \\ &\forall \pi_j, \sum_{\tau_i \in \tau} x_{i,j} \le \ell, && (11) \end{aligned}$$

with $x_{i,j} \ge 0$. We call this linear program LP-Feas(τ, π).

Proof. ($\Rightarrow$) Assume that (τ, π) is feasible. By Lemma 4.4, $(\hat{\tau}, \pi)$ is also feasible. By Theorem 5.4, there exists a feasible solution to *LP-Feas*$(\hat{\tau}, \pi)$ with $0 \le \ell \le 1$. Let the values $x^k_{i,j}$ be the values from this solution. We will demonstrate that there exists a feasible solution to *LP-Feas*(τ, π) with objective value at most one. Let

$$x_{i,j} = \sum_{k=1}^{p_i} x^k_{i,j}. \tag{12}$$

We now show that each constraint in (9)–(11) is satisfied with the $x_{i,j}$ values from (12). We first consider (9). By (12), we have

$$\begin{aligned} \forall \tau_i, \sum_{\pi_j \in \pi} x_{i,j} \cdot s_{i,j} &= \sum_{\pi_j \in \pi} \sum_{k=1}^{p_i} x^k_{i,j} \cdot s_{i,j} \\ &= \{\text{by swapping order of sums}\} \\ &\quad \sum_{k=1}^{p_i} \sum_{\pi_j \in \pi} x^k_{i,j} \cdot s_{i,j} \\ &= \{\text{by (3)}\} \\ &\quad \sum_{k=1}^{p_i} u_i / p_i = u_i. \end{aligned}$$

Thus, (9) is satisfied. We now consider (10). By (12), we have

$$\begin{aligned} \forall \tau_i, \sum_{\pi_j \in \pi} x_{i,j} &= \sum_{\pi_j \in \pi} \sum_{k=1}^{p_i} x^k_{i,j} \\ &= \{\text{by swapping order of sums}\} \\ &\quad \sum_{k=1}^{p_i} \sum_{\pi_j \in \pi} x^k_{i,j} \\ &\le \{\text{by (4)}\} \\ &\quad \sum_{k=1}^{p_i} \ell = \ell \cdot p_i. \end{aligned}$$

Thus, (10) is satisfied. Finally, we consider (11). By (12), we have

$$\begin{aligned} \forall \pi_j, \sum_{\tau_i \in \tau} x_{i,j} &= \sum_{\tau_i \in \tau} \sum_{k=1}^{p_i} x^k_{i,j} \\ &= \{\text{each } \tau_i \in \tau \text{ yields } p_i \text{ split tasks in } \hat{\tau}\} \\ &\quad \sum_{\hat{\tau}^k_i \in \hat{\tau}} x^k_{i,j} \\ &\le \{\text{by (5)}\} \\ &\quad \ell. \end{aligned}$$

Therefore, assignment of $x_{i,j}$ values according to (12) satisfies all constraints of *LP-Feas*(τ, π) with $0 \le \ell \le 1$.

($\Leftarrow$) Assume that *LP-Feas*(τ, π) has a solution so that $0 \le \ell \le 1$ holds. Let $x_{i,j}$ denote the values from this solution. We will show that there exists a solution to *LP-Feas*$(\hat{\tau}, \pi)$ with objective value at most one. For each $1 \le k \le p_i$, let

$$x^k_{i,j} = x_{i,j} / p_i. \tag{13}$$

We show that constraints (3)–(5) are satisfied when each $x^k_{i,j}$ is assigned according to (13). We first show that (3) is satisfied. Applying (13) in the left-hand side of (3), we have

$$\begin{aligned} \forall \hat{\tau}^k_i, \sum_{\pi_j \in \pi} x^k_{i,j} \cdot s_{i,j} = \sum_{\pi_j \in \pi} x_{i,j}/p_i \cdot s_{i,j} &= \frac{1}{p_i} \sum_{\pi_j \in \pi} x_{i,j} \cdot s_{i,j} \\ &= \{\text{by (9)}\} \\ &\quad u_i / p_i. \end{aligned}$$

Similarly, (4) is satisfied by applying (13), as

$$\begin{aligned} \forall \hat{\tau}^k_i, \sum_{\pi_j \in \pi} x^k_{i,j} = \sum_{\pi_j \in \pi} x_{i,j}/p_i &= \frac{1}{p_i} \sum_{\pi_j \in \pi} x_{i,j} \\ &\le \{\text{by (10)}\} \\ &\quad \frac{1}{p_i} \cdot \ell \cdot p_i = \ell. \end{aligned}$$

Finally, (5) is satisfied, as each τ_i results in p_i split tasks in $\hat{\tau}$:

$$\begin{aligned} \forall \pi_j, \sum_{\hat{\tau}^k_i \in \hat{\tau}} x^k_{i,j} &= \sum_{\tau_i \in \tau} \sum_{k=1}^{p_i} x^k_{i,j} \\ &= \{\text{by (13)}\} \\ &\quad \sum_{\tau_i \in \tau} \sum_{k=1}^{p_i} x_{i,j}/p_i \\ &= \sum_{\tau_i \in \tau} x_{i,j} \\ &\le \{\text{by (11)}\} \\ &\quad \ell. \end{aligned}$$

Thus, there exists an assignment of variables in *LP-Feas*$(\hat{\tau}, \pi)$ so that $0 \le \ell \le 1$ holds, which, by Theorem 5.4, implies that $\hat{\tau}$ is feasible. By Lemma 4.4, τ is feasible. □

Theorem 5.5 establishes a necessary and sufficient feasibility condition for (τ, π) when π is an Unrelated multiprocessor. This linear program contains $2n + m$ constraints and $n * m + 1$ variables. Linear programs can be solved in polynomial time with respect to

the number of variables [13]. Next, we derive simpler feasibility conditions for certain special cases of the Unrelated model.

5.2 Identical with Affinity Masks

We begin by defining affinity masks. An *affinity mask* $\alpha_i \subseteq \pi$ for τ_i is a nonempty set of processors on which τ_i is allowed to execute. For a subset of tasks $\tau' \subseteq \tau$, we define the *aggregate affinity mask* as $\alpha_{\tau'} = \bigcup_{\tau_i \in \tau'} \alpha_i$. The Identical with Affinity Masks model is a special case of the Unrelated model. Each speed $s_{i,j}$ is set as follows: If $\pi_j \in \alpha_i$, then $s_{i,j} = 1$, otherwise $s_{i,j} = 0$. Next, we amend the feasibility condition for this multiprocessor model presented in [17] to apply to rp-sporadic tasks.

Theorem 5.6. *The system (τ, π), where π is Identical with Affinity Masks, is feasible if and only if $\forall \tau_i, u_i \leq p_i$ and there exist values $x_{i,j} \geq 0$ that satisfy*

$$\forall \tau_i, \sum_{\pi_j \in \alpha_i} x_{i,j} = u_i \tag{14}$$

$$\forall \pi_j, \sum_{\tau_i \in \tau} x_{i,j} \leq 1. \tag{15}$$

Proof. ($\Rightarrow$) Assuming that (τ, π) is feasible, by Theorem 5.5, *LP-Feas*(τ, π) has a solution with $0 \leq \ell \leq 1$. Since $\pi_j \in \alpha_i \Rightarrow s_{i,j} = 1$ and $\pi_j \notin \alpha_i \Rightarrow s_{i,j} = 0$,

$$\sum_{\pi_j \in \pi} x_{i,j} \cdot s_{i,j} = \sum_{\pi_j \in \alpha_i} x_{i,j}.$$

Hence, *LP-Feas*(τ, π) becomes:

$$\begin{aligned} &\text{minimize } \ell \\ &\text{subject to: } \forall \tau_i, \sum_{\pi_j \in \alpha_i} x_{i,j} = u_i && (16) \\ &\forall \tau_i, \sum_{\pi_j \in \alpha_i} x_{i,j} \leq \ell \cdot p_i && (17) \\ &\forall \pi_j, \sum_{\tau_i \in \tau} x_{i,j} \leq \ell, && (18) \end{aligned}$$

with $x_{i,j} \geq 0$, noting that $x_{i,j} = 0$ when $\pi_j \notin \alpha_i$. Constraints (16) and (17) imply that $u_i \leq \ell \cdot p_i \leq p_i$. Additionally, we see that (14) and (15) follow directly from (16) and (18), respectively, when $\ell \leq 1$.

($\Leftarrow$) Here, we assume $\forall \tau_i, u_i \leq p_i$ and both (14) and (15) hold. We then demonstrate that (τ, π) is feasible by finding a solution to *LP-Feas*(τ, π) with $\ell = 1$. (14) is equivalent to (16), (15) is equivalent to (18), and (17) follows from (14) and $u_i \leq p_i$. □

Theorem 5.6 simplifies *LP-Feas*(τ, π) by removing over n constraints. Additionally, suitable $x_{i,j}$ values can be found by treating this new formulation as a *max-flow* problem [17].

5.3 Uniform

Recall that, under the Uniform multiprocessor model, a processor π_j has a uniform speed s_j for all τ_i, i.e., $s_{i,j} = s_j$. Surprisingly, for ordinary sporadic tasks, the known *utilization-based* feasibility condition under Uniform multiprocessors [10] cannot be easily derived from the linear-program-based feasibility condition under Unrelated multiprocessors [4]. Instead, the derivation of the feasibility condition relies on the *Level Algorithm* [12]. Much like Lemma 5.2, the Level Algorithm generates a schedule that can be repeated every time unit, ensuring bounded response times for sporadic tasks. Thus, to derive a utilization-based feasibility condition under Uniform multiprocessors, we apply the results of [10] to split tasks.

We first state the known feasibility condition for ordinary sporadic tasks on Uniform multiprocessors [10]. For ease of notation, we assume that tasks of τ are indexed such that $u_i/p_i \geq u_j/p_j$ if $i \leq j$, noting that $p_i = 1$ in the case of ordinary sporadic tasks, and processors of π are indexed in order of non-increasing speed ($s_i \geq s_j$ if $i \leq j$). Slightly abusing the notation, we define $\mathcal{U}_k = \sum_{i=1}^{k} u_i$, $\mathcal{P}_k = \sum_{i=1}^{k} p_i$, and $\mathcal{S}_k = \sum_{j=1}^{k} s_j$, representing the sum of the first k utilizations, parallelism levels, and speeds, respectively. For an ordinary sporadic task set τ, the system (τ, π) is feasible if and only if [10]

$$\forall 1 \leq k \leq n, \ \mathcal{U}_k \leq \mathcal{S}_{\min(k,m)}.$$

An equivalent condition, given in [16], that is more convenient for our purpose is

$$\forall \tau' \subseteq \tau, \ \mathcal{U}_{\tau'} \leq \mathcal{S}_{\min(|\tau'|,m)}. \tag{19}$$

We now give a feasibility condition for rp-sporadic systems under Uniform multiprocessors.

Theorem 5.7. *The system (τ, π), where π is a Uniform multiprocessor, is feasible if and only if*

$$\forall \tau' \subseteq \tau, \ \mathcal{U}_{\tau'} \leq \mathcal{S}_{\min(\mathcal{P}_{\tau'},m)}. \tag{20}$$

Moreover, an equivalent condition is

$$\forall k : 1 \leq k \leq n, \ \mathcal{U}_k \leq \mathcal{S}_{\min(\mathcal{P}_k,m)}, \tag{21}$$

where tasks are ordered by non-increasing u_i/p_i and processors are ordered by non-increasing speed.

Proof. Consider the system of split tasks $(\hat{\tau}, \pi)$. By (19), $\hat{\tau}$ is feasible under π if and only if

$$\forall \hat{\tau}' \subseteq \hat{\tau}, \ \mathcal{U}_{\hat{\tau}'} \leq \mathcal{S}_{\min(|\hat{\tau}'|,m)}. \tag{22}$$

Therefore, by Lemma 4.4, (22) holds if and only if (τ, π) is feasible, again using the server abstractions explained in **??**. We now prove that (20), (21), and (22) are equivalent, showing that they are all necessary and sufficient feasibility conditions for (τ, π).

(22) $\Rightarrow$ (20). Given $\tau' \subseteq \tau$, let $\hat{\tau}'$ be the set of split tasks of all tasks in τ'. This means

$$\begin{aligned} \mathcal{U}_{\tau'} &= \{\text{as the utilization of } \tau' \text{ equals that of } \hat{\tau}', \text{ by (2)}\} \\ &\quad \mathcal{U}_{\hat{\tau}'} \\ &\leq \{\text{by (22)}\} \\ &\quad \mathcal{S}_{\min(|\hat{\tau}'|,m)} \\ &\leq \{\text{as } |\hat{\tau}'| = \mathcal{P}_{\tau'}, \text{ by (2)}\} \\ &\quad \mathcal{S}_{\min(\mathcal{P}_{\tau'},m)}, \end{aligned}$$

which is exactly (20).

(20) $\Rightarrow$ (21). The conditions in (21) are just a subset of the conditions in (20). To see this, let $1 \leq k \leq n$. The first k tasks of τ form the subset $\tau' = \{\tau_1, \ldots, \tau_k\}$ with $\mathcal{U}_{\tau'} = \mathcal{U}_k$ and $\mathcal{P}_{\tau'} = \mathcal{P}_k$.

(21) ⇒ *(22)*. We prove the contrapositive of this case, i.e.,

$$\exists \hat{\tau}' \subseteq \hat{\tau},\ \mathcal{U}_{\hat{\tau}'} > \mathcal{S}_{\min(|\hat{\tau}'|,m)} \Rightarrow \quad (\neg 22)$$
$$\exists k : 1 \le k \le n,\ \mathcal{U}_k > \mathcal{S}_{\min(\mathcal{P}_k,m)}. \quad (\neg 21)$$

Let $\hat{\tau}' \subseteq \hat{\tau}$ be a set with the fewest elements that satisfies (¬22). We must find a value of k that satisfies (¬21). There are two cases:

Case 1. $|\hat{\tau}'| \ge m$. This means $|\hat{\tau}| \ge m$, as $\hat{\tau}' \subseteq \hat{\tau}$. Also, note that $\mathcal{P}_\tau = \mathcal{P}_n$. Below, we show that (¬21) holds for $k = n$.

$$\begin{aligned}
\mathcal{U}_n &= \{\text{as the utilization of } \tau \text{ equals that of } \hat{\tau}, \text{ by (1)}\} \\
&\quad \mathcal{U}_{\hat{\tau}} \\
&\ge \{\text{as } \hat{\tau}' \subseteq \hat{\tau}\} \\
&\quad \mathcal{U}_{\hat{\tau}'} \\
&> \{\text{by } (\neg 22)\} \\
&\quad \mathcal{S}_{\min(|\hat{\tau}'|,m)} \\
&= \{\text{as } |\hat{\tau}'| \ge m \text{ by the assumption of this case}\} \\
&\quad \mathcal{S}_m \\
&= \{\text{as } m \le |\hat{\tau}| = \mathcal{P}_n, \text{ by (1)}\} \\
&\quad \mathcal{S}_{\min(\mathcal{P}_n,m)}.
\end{aligned}$$

Case 2. $|\hat{\tau}'| < m$. Let $\hat{\tau}_h^j \in \hat{\tau}'$ have the lowest utilization, chosen arbitrarily in case of a tie. The utilization of $\hat{\tau}_h^j$ is

$$\begin{aligned}
\frac{u_h}{p_h} &= \{\text{as } \mathcal{U}_{\hat{\tau}'} \text{ is the sum all utilizations from } \hat{\tau}'\} \\
&\quad \mathcal{U}_{\hat{\tau}'} - \mathcal{U}_{\hat{\tau}' \setminus \{\hat{\tau}_h^j\}} \\
&> \{\text{by } (\neg 22), \text{ and } \min(|\hat{\tau}'|, m) = |\hat{\tau}'| \text{ in this case}\} \\
&\quad \mathcal{S}_{|\hat{\tau}'|} - \mathcal{U}_{\hat{\tau}' \setminus \{\hat{\tau}_h^j\}} \\
&\ge \left\{\begin{array}{l}\text{as subsets of } \hat{\tau} \text{ smaller than } \hat{\tau}' \text{ satisfy (22),} \\ \mathcal{U}_{\hat{\tau}' \setminus \{\hat{\tau}_h^j\}} \le \mathcal{S}_{|\hat{\tau}' \setminus \{\hat{\tau}_h^j\}|}\end{array}\right\} \\
&\quad \mathcal{S}_{|\hat{\tau}'|} - \mathcal{S}_{|\hat{\tau}' \setminus \{\hat{\tau}_h^j\}|} \\
&= \left\{\text{as } \mathcal{S}_k = \textstyle\sum_{i=1}^{k} s_i \text{ and } |\hat{\tau}' \setminus \{\hat{\tau}_h^j\}| = |\hat{\tau}'| - 1\right\} \\
&\quad \sum_{i=1}^{|\hat{\tau}'|} s_i - \sum_{i=1}^{|\hat{\tau}'|-1} s_i = s_{|\hat{\tau}'|}. \qquad (23)
\end{aligned}$$

Next, consider the set τ' of all tasks that have a split task in $\hat{\tau}'$, i.e., $\tau' = \{\tau_i \in \tau :: \exists \hat{\tau}_i^q \in \hat{\tau}'\}$. We denote the set containing all of the split tasks of τ' as $split(\tau')$. Thus, $\hat{\tau}' \subseteq split(\tau')$ and $|split(\tau')| = \mathcal{P}_{\tau'}$, by (2). We show (¬21) holds for $k = |\tau'|$.

$$\begin{aligned}
\mathcal{U}_k &= \mathcal{U}_{|\tau'|} \\
&\ge \left\{\begin{array}{l}\text{by task order, the total utilization of} \\ \text{the first } |\tau'| \text{ tasks is at least that of } \tau'\end{array}\right\} \\
&\quad \mathcal{U}_{\tau'} \\
&= \{\text{as the utilization of } \tau' \text{ equals that of } split(\tau'), \text{ by (2)}\} \\
&\quad \mathcal{U}_{split(\tau')} \\
&\ge \{\text{as } split(\tau') = \hat{\tau}' \cup (split(\tau') \setminus \hat{\tau}')\} \\
&\quad \mathcal{U}_{\hat{\tau}'} + \mathcal{U}_{split(\tau') \setminus \hat{\tau}'} \\
&> \{\text{by } (\neg 22), \text{ and } \min(|\hat{\tau}'|, m) = |\hat{\tau}'| \text{ in this case}\} \\
&\quad \mathcal{S}_{|\hat{\tau}'|} + \mathcal{U}_{split(\tau') \setminus \hat{\tau}'} \\
&\ge \left\{\begin{array}{l}\text{as the right term has } |split(\tau') \setminus \hat{\tau}'| \text{ elements,} \\ \text{and } u_h/p_h \text{ is defined to be the least of these}\end{array}\right\} \\
&\quad \mathcal{S}_{|\hat{\tau}'|} + \left|split(\tau') \setminus \hat{\tau}'\right| \frac{u_h}{p_h} \\
&> \{\text{by (23), } u_h/p_h > s_{|\hat{\tau}'|}\} \\
&\quad \mathcal{S}_{|\hat{\tau}'|} + \left|split(\tau') \setminus \hat{\tau}'\right| s_{|\hat{\tau}'|} \\
&\ge \{\text{as } |split(\tau') \setminus \hat{\tau}'| \ge \min(|split(\tau')|, m) - |\hat{\tau}'|\}. \\
&\quad \mathcal{S}_{|\hat{\tau}'|} + \sum_{j=|\hat{\tau}'|+1}^{\min(|split(\tau')|,m)} s_{|\hat{\tau}'|} \\
&\ge \{\text{by processor order, } s_j \le s_{|\hat{\tau}'|} \text{ if } j > |\hat{\tau}'|\}. \\
&\quad \mathcal{S}_{|\hat{\tau}'|} + \sum_{j=|\hat{\tau}'|+1}^{\min(|split(\tau')|,m)} s_j \\
&= \left\{\text{by definition, } \mathcal{S}_k = \textstyle\sum_{i=1}^{k} s_i\right\} \\
&\quad \mathcal{S}_{\min(|split(\tau')|,m)} \\
&= \left\{\left|split(\tau')\right| = \mathcal{P}_{\tau'}, \text{ by (2)}\right\} \\
&\quad \mathcal{S}_{\min(\mathcal{P}_{\tau'},m)}.
\end{aligned}$$

Thus, (¬21) holds. This completes the proof that (21) ⇒ (22).

We have shown that (22) ⇒ (20) ⇒ (21) ⇒ (22). □

Condition (20) is in a convenient form for certain types of analysis, much as is (19) [16]; however, it requires checking a number of conditions that is exponential in the size of the task system. Thus, we also present (21), as it is an equivalent polynomial time test.

6 EXPERIMENTAL EVALUATION

Prior work has explored some of the effects of parallelism levels in real-time systems [2, 3], but, to our knowledge, no prior work has experimentally investigated the feasibility impacts associated with enabling or discouraging node parallelism or choosing among the Unrelated, Uniform, and Identical models. In the context of this work, it would not be enlightening to randomly generate task sets and look solely at how parallelism affects their feasibility, as the results would simply reflect the distributions chosen for generating task utilizations. For example, the percentage of systems with tasks that required $p_i \ge 2$ would be equivalent to the proportion of tasks generated with utilizations at least 2. Instead, we compared how different multiprocessor models affect feasibility. What is lost when a model is oversimplified, as might be the case in real-world applications? What is gained by allowing parallelism in a system that can support it? In this section, we discuss the results of the experimental investigation we conducted to investigate such questions.

Task system generation. In our investigation, systems were randomly generated using three parameters: processor count, utilization distribution, and processor speed distribution. The processor count, m, was one of 4, 8, or 16.

Utilization distributions were either uniform or bimodal. Uniform distributions used sampling ranges that were either *light*, [0.001, 0.1], *moderate*, [0.1, 0.4], or *heavy*, [0.5, 0.9]. Bimodal distributions used two sampling ranges, [0.001, 0.5] and [0.5, 0.9]. The respective weights of these two ranges were 8/9 and 1/9 for *light* systems, 6/9 and 3/9 for *moderate* systems, and 4/9 and 5/9 for *heavy* systems.

Systems were randomly generated with processors grouped into either one, two, or three different speed classes. Identical systems had just one speed class, with a speed chosen uniformly from [0.5, 0.9]. Systems with two speed classes had both *fast* and *slow* speeds, sampled from [0.5, 0.9] and [0.1, 0.4], respectively. Such systems are either Uniform, where all processors in the same class had the same speed, or Unrelated, where the speed of each processor was chosen independently for each task. Similarly, systems with three speed classes were generated with *fast*, *medium*, and *slow* speeds, sampled from [0.6, 0.9], [0.3, 0.5], and [0.1, 0.2], respectively. These systems also had Uniform and Unrelated variations. An additional *random* system type allowed for every processor to be in its own class, where every speed was sampled uniformly from [0.0, 1.0]. These options resulted in 189 different possible system configurations.

Experimental design. Given a configuration as defined above, task systems were generated that had total utilizations ranging from 1.0 to m. The number of ways to assign parallelism levels to each of these tasks is too large to perform a reasonable study. Instead, we first focus on two particular assignments and then attempt to optimally choose the values for p_i.

While increasing parallelism should lead to more systems becoming feasible, it is not easy to see how large of an effect parallelism will have just by looking at the feasibility tests given. We therefore used our tests to check each system when $p_i = 1$ for all τ_i and when $p_i = m$ for all τ_i. Then, each of the above Unrelated systems was copied and *cast* to a new Uniform system. This means that for each π_j in the copied system, every speed $s_{i,j}$ was reduced to $\min_{1 \le i \le n}(s_{i,j})$, the slowest speed of any task on that processor. This mimics the way real-world systems might be pessimistically approximated using simpler models. Next, each Uniform system was cast to an Identical system, in which every speed was set to the slowest speed in the entire system. This was done to quantify the utilization lost when systems are subjected to analytical oversimplifications. This study is not intended to be a direct comparison of these multiprocessor models, as casting will necessarily lose capacity. We instead measure the benefits provided by parallelism, circumventing the issue of our results being directly proportional to the chosen random task utilization distributions.

For each system, we computed the *Average Necessary Parallelism (ANP)* to achieve feasibility. We achieved this by converting the condition from Theorem 5.5 into an Integer Linear Program (ILP), with each p_i allowed to range from 1.0 to m and $\ell = 1$. This ILP was then solved for the minimum average p_i value, and the objective value was recorded. We similarly cast every system to its Uniform or Identical approximation and repeated the tests. If fewer than 5% of systems were able to be made feasible in this way, then no data point was recorded.

Systems were generated for each data point until the margin of error for the 95% confidence interval was within 1% of the mean, or until 5,000 systems were generated, whichever came first. Example graphs are given in Figure 4, Figure 5, and Figure 6.

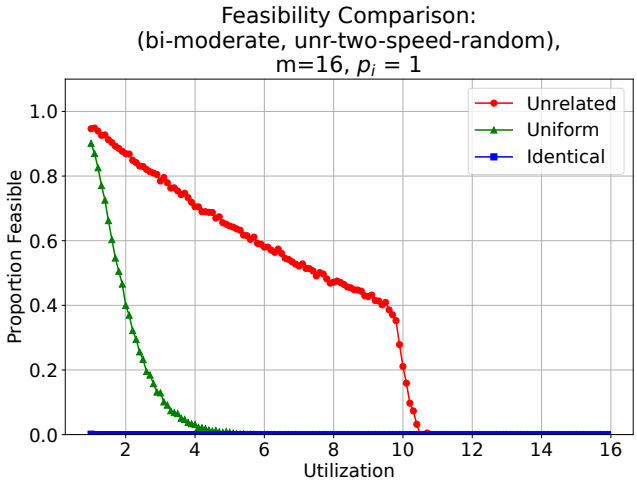


Figure 4: Feasibility comparison for task sets generated with moderate bimodal utilization distributions and two speed classes for Unrelated.

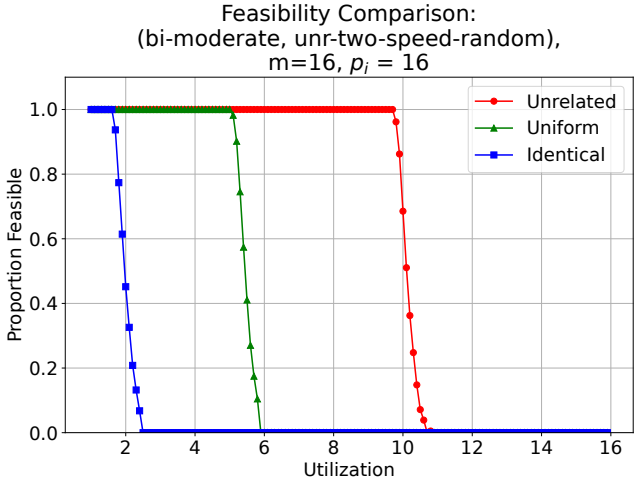


Figure 5: Feasibility comparison for task sets generated with moderate bimodal utilization distributions and two speed classes for Unrelated.

Performing all experiments yielded 567 total graphs. On average, each graph contained about 250 data points, and each point was computed with about 400 randomly generated systems. Some selected graphs can be found in Appendix B.[4]

6.1 Observations.

We lack the space for an in-depth discussion of all 567 graphs, so we use metrics to discuss them holistically. First, for each feasibility

[4]Additionally, all graphs and data can be found online at https://jamesanderson.web.unc.edu/papers/.

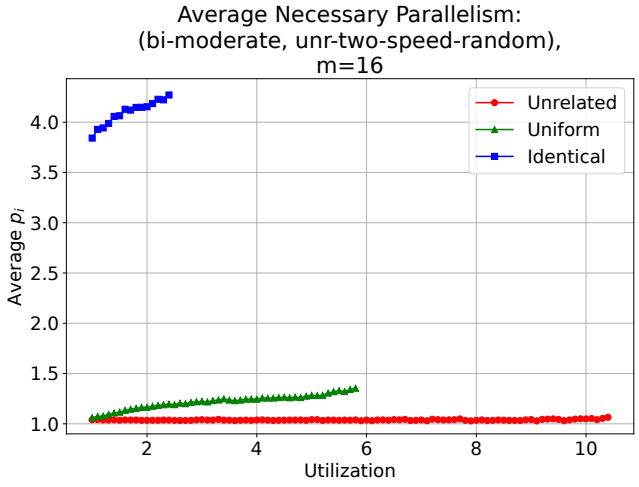


Figure 6: Average Necessary Parallelism for task sets generated with moderate bimodal utilization distributions and two speed classes for Unrelated. Graphs stop when fewer than 5% of systems are able to be made feasible.

graph, like those in Figure 4 and Figure 5, we measured the area under each curve. These values were then normalized by dividing by the processor count used in the corresponding experiment. We call this the *Normalized Feasible Region (NFR)* for each graph. For example, Figure 5 has an NFR of 0.61 for its Unrelated curve, 0.30 for its Uniform curve, and 0.07 for its Identical curve.

Next, we examined the utilization where each curve crosses the 80% feasible line. We will call this the *0.8-Threshold*, and it serves to measure the resulting feasibility of the system while allowing us to discuss the utilization lost when casting to simpler models.

Observation 1. **Oversimplifying the system model reduced the 0.8-Threshold by 33%, on average.** When casting a model to a simpler type, such as from Unrelated to Uniform or from Uniform to Identical, systems became infeasible at significantly lower utilizations. We also noticed a roughly 70% average decrease in NFR across all systems after being cast.

Observation 2. **When parallelism was allowed, NFR increased by 1.7 times on average.** For systems with a parallelism level of one, the average NFR was 0.26. This metric increased to 0.45 when p_i was set to m for all tasks. Additionally, ANP across all systems was 1.5, meaning that significant parallelism is generally not required for a system to become feasible. Overall, Identical systems required much more parallelism, due to the pessimistic nature of needing to use the slowest speed in the system.

We also saw that the 0.8-Threshold value increased by 2.1 times, on average, when parallelism levels were set to m, meaning that system utilization could be allowed to double under the right circumstances. Many systems were able to regain lost utilization after being cast to a simpler model when parallelism was allowed.

Figure 4 and Figure 5 provide two examples of these observations using systems with moderate bimodal utilization distributions running on 16 Unrelated processors with two speed classes. By comparing the three curves in Figure 4, the dramatic effects of casting systems can be seen. Moreover, the effects of allowing parallelism can be seen in Figure 5. Figure 6 shows how ANP for these systems was generally much less than the maximum possible value of 16, especially for the Uniform and Unrelated cases. This is especially interesting in the Unrelated case where NFR increased from 0.38 to 0.61 with parallelism levels of 16 despite having an ANP of 1.03.

The shapes of these curves were common throughout our entire dataset. 12 more examples are given in **??**.

7 CONCLUSION

Real-time computing today is being impacted by two trends: a growing need to support complex software workloads and increasing heterogeneity in the hardware platforms for supporting such workloads. In this work, we have considered the rp-sporadic task model, a generalization of the standard sporadic task model that allows for settable per-task parallelism levels. This model increases scheduling flexibility and can thereby support workloads that are infeasible under the standard sporadic model by leveraging concurrency.

Additionally, we have considered a range of multiprocessor platform models with varying degrees of heterogeneity. For each of these platform models, we have presented feasibility conditions for the scheduling of rp-sporadic task sets. In deriving these conditions, we showed that there exists a mapping from rp-sporadic task sets to standard sporadic task sets. This mapping serves as a useful analytical tool for understanding the rp-sporadic model.

Lastly, we covered a large-scale experimental evaluation of randomly generated systems. In these experiments, model oversimplification greatly impacted the feasibility of systems while the added flexibility offered by the rp-sporadic model offset such effects.

With these feasibility conditions and experimental results in place, we plan in future work to study the issue of scheduler optimality under the rp-sporadic model in the context of these various multiprocessor platform models. Both HRT and SRT optimality are of interest, as is the runtime efficiency of any considered schedulers.

REFERENCES

[1] Shareef Ahmed and James H. Anderson. 2021. Tight Tardiness Bounds for Pseudo-Harmonic Tasks Under Global-EDF-Like Schedulers. In *Euromicro Conference on Real-Time Systems, ECRTS*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:24.

[2] Tanya Amert, Sergey Voronov, and James H. Anderson. 2019. OpenVX and Real-Time Certification: The Troublesome History. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 312–325.

[3] Tanya Amert, Ming Yang, Sergey Voronov, Saujas Nandi, Thanh Vu, James H. Anderson, and F. Donelson Smith. 2021. The price of schedulability in cyclic workloads: The history-vs.-response-time-vs.-accuracy trade-off. *Journal of Systems Architecture* 120 (2021), 102292.

[4] Sanjoy Baruah. 2004. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 37–46.

[5] Sanjoy Baruah and Björn Brandenburg. 2013. Multiprocessor Feasibility Analysis of Recurrent Task Systems with Specified Processor Affinities. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 160–169.

[6] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. 2020. Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 216–221.

[7] B. B. Brandenburg. 2011. *Scheduling and locking in multiprocessor real-time operating systems.* Ph. D. Dissertation. University of North Carolina, Chapel Hill, NC.

[8] Glenn A. Elliott, Kecheng Yang, and James H. Anderson. 2015. Supporting Real-Time Computer Vision Workloads Using OpenVX on Multicore+GPU Platforms. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 273–284.
[9] Jeremy P. Erickson and James H. Anderson. 2011. Response Time Bounds for G-EDF Without Intra-Task Precedence Constraints. In *International Converence On Principles Of Distributed Systems, ICPDS*. IEEE, 128–142.
[10] Shelby Funk, Joel Goossens, and Sanjoy Baruah. 2001. On-line scheduling on uniform multiprocessors. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 183–192.
[11] The Khronos Group. [n. d.]. The OpenVX Specification. https://registry.khronos.org/OpenVX/specs/1.3.1/html/OpenVX_Specification_1_3_1.html#sub_graph_parameters
[12] Edward C. Horvath, Shui Lam, and Ravi Sethi. 1977. A Level Algorithm for Preemptive Scheduling. *J. ACM* 24, 1 (1977), 32–43.
[13] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2021. A faster algorithm for solving general LPs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*. ACM, 823–832.
[14] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* 20, 1 (1973), 46–61.
[15] Aloysius Mok. 1983. *Fundamental design problems of distributed systems for the hard-real-time environment*. Ph. D. Dissertation. Massachusetts Institute of Technology.
[16] Stephen Tang, Sergey Voronov, and James H. Anderson. 2019. GEDF Tardiness: Open Problems Involving Uniform Multiprocessors and Affinity Masks Resolved. In *Euromicro Conference on Real-Time Systems, ECRTS*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13–21.
[17] Sergey Voronov and James H. Anderson. 2018. An Optimal Semi-Partitioned Scheduler Assuming Arbitrary Affinity Masks. In *IEEE Real-Time Systems Symposium, RTSS*. IEEE, 408–420.
[18] Kecheng Yang and James H. Anderson. 2014. Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *IEEE Symposium on Embedded Systems for Real-time Multimedia, ESTIMedia*. IEEE, 30–39.