Response Time Analysis for Fixed-Priority Preemptive Uniform Multiprocessor Systems

Binqi Sun
TUM School of Engineering and Design, Technical University of Munich, Germany

Tomasz Kloda
LAAS-CNRS, Insa de Toulouse, France

Marco Caccamo
TUM School of Engineering and Design, Technical University of Munich, Germany

Abstract

We present a response time analysis for global fixed-priority preemptive scheduling of constrained-deadline tasks upon a uniform multiprocessor where each processor can be characterized by a different speed. A fixed-priority scheduler assigns the jobs with the highest priorities to the fastest processors. Since determining whether all tasks can meet their deadlines is generally intractable even with identical processors, we propose two sufficient schedulability tests that calculate upper bounds on the task's worst-case response time within polynomial and pseudo-polynomial time. The proposed tests leverage the linear programming model to upper bound the interference of the higher-priority tasks. Furthermore, we identify specific conditions and platforms upon which the problem can be solved more efficiently within linear time. These formulations are used to iteratively evaluate and refine possible solutions until a safe upper bound on the task's worst-case response time is found. Additionally, we demonstrate that, with specific minor modifications, the proposed tests are compatible with Audsley's optimal priority assignment. Experimental evaluations performed on synthetic task sets show that the proposed approach outperforms the state-of-the-art methods.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Real-time scheduling, Uniform multiprocessor, Response time analysis

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.17

Funding *Marco Caccamo*: Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

1 Introduction

Heterogeneous multiprocessor systems integrate on the same die different core types optimized for specific workloads and different performance goals. Typically, such design can involve several high-performance cores that boost performance co-allocated with low-power but slower cores that reduce energy consumption. *ARM's big.LITTLE* succeeded by *DynamIQ*, and *Intel's Alder Lake* are examples of such architectures.

The scheduling problem where processors have different speeds, known in the literature as *uniform* multiprocessors, has been extensively studied. Most works adopt *dynamic-priority* scheduling policies [10, 11, 18, 39, 40, 43, 50, 64, 65], resulting in good processor utilization and efficient schedulability tests. On the other hand, *fixed-priority* scheduling policies are more commonly used in the industry [1], having the advantage of more precise control of high-priority task timelines [6, 25, 61]. Existing fixed-priority schedulability tests for uniform platforms [15–17] relate platform and task set characteristics (*e.g.,* processor speed, system load) with schedulability and return a boolean answer, either schedulable or not schedulable. Consequently, the task response time information, often required in offline design [34],

© Binqi Sun, Tomasz Kloda, and Marco Caccamo;
licensed under Creative Commons License CC-BY 4.0

remains unknown. Moreover, the existing fixed-priority schedulability tests are derived for specific priority assignments (*e.g., Deadline Monotonic* in [15] or *Rate Monotonic* in [16]), which have been shown ineffective for global fixed-priority scheduling [31].

In this paper, we introduce *response time analysis* (*RTA*) for *global fixed-priority preemptive* scheduling of *constrained-deadline sporadic* task systems upon *uniform* multiprocessors compatible with any priority assignment. First, we upper bound the total interference generated by high-priority tasks using a *linear programming* problem formulation. Such problems can be solved in polynomial time, but we also establish specific conditions for processor speeds when the total interference calculation can be done in linear time. Finally, we extend the previous results from *identical multiprocessors* to calculate an upper bound on a task workload, limiting its carry-in and carry-out interference. Based on these results, we devise two schedulability tests (*sufficient* conditions), running respectively in polynomial and pseudo-polynomial time, for *sporadic* task systems. The tests return an upper bound on the worst-case response time of each task. Additionally, we show that the proposed tests can be modified to be compatible with Audsley's optimal priority assignment (OPA) policy [4, 5]. The evaluation results demonstrate a significant improvement in the schedulability ratio over the state-of-the-art schedulability approaches.

The remainder of this paper is organized as follows. Section 2 gives background on multiprocessor platform types, task models, and scheduler classifications. Section 3 covers related work regarding aperiodic and recurrent tasks scheduling upon uniform platforms. Section 4 gives the system model and notation used in the rest of this paper. Section 5 contains the main results. The section starts with the definition of the linear programming problem for cumulative interference calculation. The following subsections derive platform-specific conditions for which the interference computation can be done in linear time. Finally, we generalize the previously obtained schedulability conditions for the *sporadic* task model, which requires an upper bound for a single task workload over a generic time interval. Section 6 contains schedulability test evaluations and Section 7 concludes the paper with a summary and future research directions.

2 Background

We briefly review the most relevant concepts related to multiprocessor real-time scheduling.

2.1 Processors

Three kinds of multiprocessor platforms can be distinguished [14, 27, 32]:

Identical. All the processors are identical in terms of computing capacity (*e.g.,* given two identical processors and two jobs, each job always executes with the same speed regardless of which processor the job is running upon).

Uniform. Each processor has its own computing capacity and a job that executes uninterruptedly on a processor with computing capacity s for $t > 0$ time units completes $s \cdot t$ of its execution requirement (*e.g.,* given two uniform processors and two jobs, if the first processor is faster than the second one, both jobs execute faster on the first processor and slower on the second processor).

Unrelated. Each job can have a different execution rate r for each processor and completes $r \cdot t$ units of execution when executing uninterruptedly on that processor for $t > 0$ time units (*e.g.,* given two unrelated processors and two jobs, the first job can execute faster on the first processor than on the second one while the second job can execute faster on the second processor than on the first one).

2.2 Jobs and tasks

A *job* is a single process that performs a specific computation using processor resources. It can be characterized by its *worst-case execution time* and *release* time. We will call a job that executes only once as *aperiodic*. A *recurrent* task can generate a potentially infinite sequence of identical *jobs* (also called task *instances*). There are two main task activation models: *periodic* and *sporadic*. In the periodic task, the arrivals of the consecutive jobs are separated by a fixed time interval. In the sporadic task, the arrivals of any two consecutive jobs of the same task are separated by the minimum inter-arrival time elapsed since the first release.

Each real-time task is characterized by a *deadline* by which each task job must complete its execution. There are three types of constraints on task deadlines: *implicit*, when each task has its deadline equal to its period, *constrained*, when each task has a deadline that is less than or equal to its period, and *arbitrary*, when a deadline of each task might be less than, equal to or even greater than its period.

2.3 Schedulers

A *scheduling algorithm* decides at each point in time which job should run on which processor. This decision, in *static* algorithms, is based on fixed parameters assigned to tasks before their activation or, in *dynamic* algorithms, depends on tasks and system properties that might change over time. In particular, in *fixed-priority* scheduling algorithms, each task has a constant (static) priority determined offline, and the active jobs with the highest priorities at a given time are selected for execution.

A *preemptive* scheduler can stop any job running upon a processor and resume it later (during that time interval, the processor can execute other jobs). In contrast, a *non-preemptive* scheduler executes each job uninterruptedly: once the scheduler starts a job, the job runs until full completion.

In *partitioned* scheduling approaches, the jobs are statically mapped to the processors such that each job is assigned to a processor on which it can execute. In *global* scheduling, each job can execute upon any processor and migrate from one processor to another.

3 Related work

We review four main areas of related work: scheduling aperiodic jobs on a uniform multiprocessor, scheduling periodic tasks on a uniform multiprocessor, response time analysis for uniprocessor and multiprocessor, and complexity of multiprocessor scheduling problems.

3.1 Multiprocessor aperiodic job scheduling

Several scheduling algorithms have been proposed to solve various schedule optimization problems of aperiodic jobs (*i.e.,* a job that executes only once) executed on uniform platforms. Gonzalez and Sahni [42] propose a linear-time algorithm to obtain a preemptive schedule with minimal completion time for independent aperiodic jobs that execute upon uniform processors. For the same problem, Horvath et al. [50] use the concept of *task level* to construct a schedule where the remaining job executions and assigned processor speeds are proportional. Federgruen and Groenevelt [38] apply network flow techniques to minimize the maximum lateness of the schedule on uniform processors. Gonzalez [41] considers the problem of mean flow time minimization in a preemptive schedule. Non-preemptive parallel scheduling problems in general form with arbitrary processing times are already

NP-hard for identical processors [49], so heuristics and constrained models are studied in this area. Graham et al. [43] consider the problem of minimizing the makespan for non-preemptive scheduling on uniform processors where all jobs have single-unit processing requirements. An optimal schedule can be found in polynomial time by modeling the problem as a transportation network.

3.2 Multiprocessor recurrent task scheduling

A *recurrent* task can generate a potentially infinite sequence of identical *jobs* according to two activation models: *periodic* and *sporadic*.

Baruah [10] obtains in polynomial time a schedule for periodic fully preemptive tasks executed upon uniform multiprocessor by applying the makespan minimization scheduling rule for aperiodic jobs proposed by Graham et al. [43] and shows that the same problem under integer boundary constraint (*i.e.,* a job can be preempted only if it has completed an integer number of execution units) is NP-hard in the strong sense.

The *Earliest Deadline First* (*EDF*) [57] scheduling algorithm was successfully applied in the context of uniform multiprocessor systems. Funk et al. [40] apply the resource augmentation technique [51] and establish the relationship between a generic task set feasibility on a less powerful platform and its *EDF*-schedulability on a more powerful one to derive global *EDF* schedulability test for implicit-deadline periodic tasks. In particular, the exact feasibility condition is obtained by extending the previously mentioned task level algorithm [50]. Baruah [11] improves the above test by showing that the test is robust with respect to processor speeds (*i.e.,* a task set deemed to be schedulable upon a particular platform must also be schedulable on a more powerful one). For partitioned uniform processors, Funk and Baruah [39] propose a method for periodic tasks allocation. Based on the concept of *demand bound function* [19], Baruah and Goossens [18] propose a new schedulability test for constrained-deadline sporadic tasks when scheduled using global *EDF*. Another test for global *EDF* is proposed in [13].

Following the same research line as in [40], Baruah and Goossens derive an efficient sufficient schedulability test for implicit-deadline periodic tasks executed under *Rate Monotonic* policy [16,17] and for arbitrary-deadline sporadic tasks executed under *Deadline Monotonic* policy [15]. The latter work also applies to the model considered in this paper, however, it cannot be used to obtain the task worst-case response time information and cannot be applied to other priority assignment rules. Cucu and Goossens [29] characterize the feasibility interval for a fixed-priority preemptive scheduler upon a uniform multiprocessor. The latter results apply to periodic tasks only and cannot be generalized to sporadic tasks.

3.3 Response time analysis

Worst-case response time is the longest time interval between task release and its completion. To compute it, a method called *Response time analysis* (*RTA*) evaluates for a task under analysis the interference generated by other tasks in a given time interval. In particular, under fixed-priority uniprocessor scheduling, *RTA* [7,8,54] is a sufficient and necessary test that can be evaluated in pseudo-polynomial time. For identical multiprocessor systems, only sufficient *RTA* tests [2,9,21,22,58] can be derived in the same complexity class. The total interference upper bound used in the tests is constructed by considering that the interference is distributed evenly among different processors, keeping them busy as long as possible to prevent the task under analysis from executing. Compared to the uniprocessor case, it is more challenging to estimate the interference generated by other tasks as the *critical instant* (*i.e.,* task arrival leading to the longest task response time) might be unknown [53] (for uniprocessor, it is a synchronous arrival of all tasks [57]) and postponing the arrival of consecutive jobs of a task can have a detrimental effect on the task set schedulability [2] (for uniprocessor, consecutive jobs should arrive as soon as possible [57]). In [64], Yang and Anderson derive the response time upper bounds for preemptive and non-preemptive global *EDF* on uniform multiprocessors for the parallel tasks. Linear programming has also been adopted to derive the response time bounds. Gujarati et al. [47] apply it to identical multiprocessors with arbitrary processor affinities. Several other works [30, 36, 56, 66, 67] model a response time as an integer linear program for single processor systems.

3.4 Time complexity of multiprocessor scheduling

The problem of deciding whether a periodic task set is schedulable on identical processors under a fixed-priority policy has been shown to be NP-hard in the strong sense [55] (the problem of verifying schedulability of a task set that runs on multiprocessor is reduced from the *3-partition* problem that has been shown to be strongly NP-complete). The theorem considers that all tasks have the same period and thus can be easily applied to the aperiodic jobs as well. Clearly, the identical multiprocessor is a special case of uniform multiprocessor scheduling considered in this paper, and thus the problem remains strongly NP-hard. Response time computation (*i.e.,* finding its exact value) has been proven to be NP-hard for fixed-priority scheduling even on a single processor [35]

4 System model

We consider a global fixed-priority preemptive scheduling of sporadic constrained-deadline tasks upon a uniform multiprocessor platform comprised of $m \geq 1$ processors.

In this work, we consider uniform multiprocessor platform. Each processor is characterized by a constant computing capacity (also denoted as speed) $s_j \in \mathbb{R}_+$. We assume that the processors are indexed in non-increasing order of computing capacities: $\forall\ 1 \leq j < m : s_{j+1} \leq s_j$. The first processor is the fastest, having computing capacity s_1, the last is the slowest, having computing capacity s_m. Following Funk et al. [40], we define the cumulative computing capacity of k fastest processors as:

▶ **Definition 1** (S_k cumulative computing capacity of k fastest processors)**.**

$$S_k = \sum_{j=1}^{k} s_j. \tag{1}$$

Each task τ_i gives rise to a potentially infinite sequence of identical jobs. Task τ_i releases jobs sporadically after the minimum inter-arrival time T_i (period), and each of its jobs J_i must execute for at most C_i execution units to complete before its deadline that occurs D_i time units after the job's release where C_i is the worst-case execution requirement of each job of task τ_i and D_i is the task τ_i relative deadline assumed to be less than or equal to the task period (*i.e., constrained deadlines*): $D_i \leq T_i$. For instance, it takes 2 time units to complete a job with an execution requirement of 6 using a processor with a capacity of 3, and 4 time units to complete the same job using a processor with a capacity of 1.5. All the above job and task parameters are positive integers. The tasks are independent (*e.g.,* do not access shared data or exchange messages) and do not share resources other than the processors. Each task might execute on at most one processor at a time (*i.e.,* no single job parallelism). The task worst-case execution requirement includes preemption, context switch, migration, and scheduler overheads. Once a job starts, it will not voluntarily self-suspend its execution.

We consider a set of n tasks which are scheduled by a fixed-priority preemptive scheduler. Each task has a unique priority, and each job of the task inherits its priority. At each point in time, the scheduler assigns the ready jobs with the highest priorities to the fastest processors according to the priority order (*e.g.*, among all ready jobs at a given time instant, a job with the highest priority runs on the fastest processor, and a job with the m-highest priority on the slowest one). When a new job with a priority higher than the priority of any currently running job becomes ready, all running jobs with lower priorities are preempted, and the processors are reassigned. We consider a continuous scheduler where preemptions may occur at any time instant. We introduce the notation $hp(i)$ for the set of tasks with priorities higher than the priority of task τ_i. We also define task τ_i utilization as $U_i = C_i/T_i$ and the task set utilization as $U = \sum_{i=1}^{n} U_i$.

The *worst-case* task τ_i *response time* R_i is the maximum time duration between the release of any job of task τ_i and the time the job completes execution. We say that task τ_i is *schedulable* if each job of τ_i always meets its deadline: $R_i \leq D_i$.

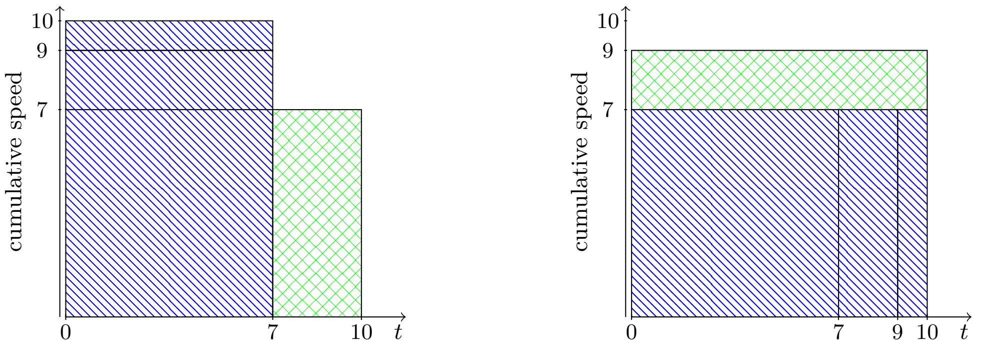
5 Response time analysis

We derive the response time analysis for constrained sporadic task systems executed on a uniform multiprocessor platform under a fixed-priority preemptive scheduling policy. Our approach follows the same general framework as the other response time analysis methods. The schedulability test seeks a minimal time interval for which the processing resources are sufficient to complete the maximum possible cumulative execution requirement generated, within such an interval, by the task under analysis and the other higher-priority tasks. In Section 5.2, we obtain the worst-case response time upper bound for aperiodic jobs by formulating a linear programming problem for finding the interfering jobs' distribution across different processors having the most severe impact on the job under analysis. In Sections 5.3, 5.4 and 5.5, we identify platforms upon which the problem can be solved more efficiently. Finally, in Section 5.6, we generalize the aperiodic task test for sporadic tasks by upper bounding the sporadic task execution requirement over any generic time interval.

5.1 Problem statement

On identical processors, the worst-case interference for a job is when all processors are running higher-priority jobs and the job is completely blocked (otherwise, since all processors have the same speed, the job can run on a processor without any interference). The following example shows that on a uniform multiprocessor, the worst case might be when only a few processors are running the higher-priority jobs, and the job under analysis is running simultaneously.

▶ **Example 2.** Consider the execution of four jobs: J_1 (highest priority), J_2 (high priority), J_3 (low priority), J_4 (lowest priority) within time interval $\Delta = 10$. The maximal execution requirements of jobs J_1, J_2, J_3, and J_4 are respectively $C_1 = 49$, $C_2 = 14$, and $C_3 = 7$. The uniform multiprocessor upon which the jobs execute comprises three processors with computing capacities of $s_1 = 7$, $s_2 = 2$, and $s_3 = 1$. Let us assume that job J_4 is released at time instant 0 and jobs J_1, J_2, and J_3 can be released at any arbitrary instant. We evaluate the processing time available in a time interval of length 10 for the lowest priority job J_4. If jobs J_1, J_2, and J_3 occupy the processors simultaneously in the interval $[0, 7]$, as depicted in Figure 1 (a), job J_4 can execute at most $3 \cdot 7 = 21$ units within the interval $[7, 10]$, running on the fastest processor (J_4 execution is marked as the green ▨ area in both figures while the other jobs execution as the blue ▨ area). If the jobs arrive asynchronously and run on the fastest processor one after another, then the second-fastest processor is available for job J_4 during the entire interval $[0, 10]$ resulting in $10 \cdot 2 = 20 < 21$ of available computation units as shown in Figure 1 (b).



(a) Higher-priority jobs arrive synchronously. **(b)** Higher-priority jobs arrive one after another.

Figure 1 Fixed-priority schedules of four jobs on three processors ($m = 3$) from Example 2.

Clearly, the above example shows that the synchronous arrival of high-priority jobs might not always be a critical instant (*i.e.,* a task arrival instant at which the task has the longest response time [57]) as it has already been proved for identical multiprocessor [53]. The example shows also, and more importantly, that concentrating all high-priority jobs interference in the same time interval might not always lead to the worst case. It cannot be decided *a priori* whether the worst case is when all processors are busy, or just a few faster processors are kept busy but for longer. We formulate this problem in the following subsection.

5.2 Response time analysis for aperiodic jobs

We search for a safe upper bound $\hat{R}_i$ on the aperiodic job J_i worst-case response time R_i.

In fixed-priority preemptive scheduling, job execution is not affected by lower-priority jobs. The interference can arise only from higher-priority jobs. Consider job J_i and suppose that each higher-priority job $J_k \in hp(i)$ executes $0 \leq e_k \leq C_k$ computation units during the job J_i worst-case response time R_i. Let $\Delta_j \geq 0$ be a cumulative time length within

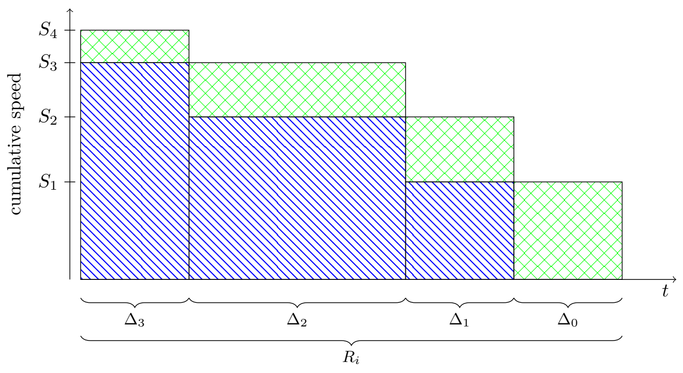


Figure 2 Execution of higher-priority jobs $hp(i)$ ▨ and job J_i under analysis ▩.

the time interval $[0, R_i]$ during which $0 \leq j \leq m$ processors are busy executing $hp(i)$ jobs. During Δ_j, job J_i executes on the j+1-th processor with a speed of s_{j+1}, for $j < m$, and on none processor for $j = m$. Figure 2 illustrates our approach. If $i \leq m$, then no more than $i-1$ fastest processors can be busy executing $hp(i)$ jobs while job J_i runs on a processor with computing capacity no slower than s_i. By abuse of notation, we define $m(i) = \min\{m, i-1\}$ as the maximal number of processors that can be executing $hp(i)$ jobs while job J_i is ready for execution or already executing. For the sake of notation, during time interval $\Delta_0 = R_i - \sum_{j=1}^{m(i)} \Delta_j$ none of the $hp(i)$ jobs are executing. If job J_i has the worst-case response time R_i, then the following conditions must be satisfied:

$$\sum_{j=1}^{m(i)} S_j \cdot \Delta_j \;=\; \sum_{k \in hp(i)} e_k \tag{2}$$

$$\sum_{j=0}^{m(i)} \Delta_j \;=\; R_i \tag{3}$$

$$\sum_{j=0}^{m(i)} s_{j+1} \cdot \Delta_j = C_i \quad \text{with} \quad s_{m+1} = 0. \tag{4}$$

Equation (2) ensures that the interference generated by the $hp(i)$ jobs (please recall that e_k is the J_k's actual execution time) fits into all intervals Δ_j for all $j : 1 \leq j \leq m(i)$ and Equation (3) that the total time of the interference is within R_i. Equation (4) gives the number of execution units available to job J_i within the time interval $[0, R_i]$ that must be equal to C_i as J_i completes its execution at R_i.

Based on the above properties, we derive an upper bound $\hat{R}_i$ on the job J_i worst-case response time R_i. We formulate a linear programming problem (LP) where the variables are intervals $\Delta_0, \Delta_1, \ldots, \Delta_{m(i)}$ and the objective function is an upper bound $\hat{R}_i$ on the job J_i worst-case response time. Since the actual execution time e_k of each higher-priority job $J_k \in hp(i)$ is unknown, we upper bound its value by its worst-case execution time C_k. Equations (2)–(4) are rewritten as follows to formulate an instance of linear programming problem:

$$\text{maximize:} \quad \hat{R}_i \;=\; \sum_{j=0}^{m(i)} \Delta_j \tag{5}$$

$$\text{subject to:} \quad \sum_{j=1}^{m(i)} S_j \cdot \Delta_j \;\leq\; \sum_{k \in hp(i)} C_k \tag{6}$$

$$\sum_{j=0}^{m(i)} s_{j+1} \cdot \Delta_j \;=\; C_i \quad \text{with} \quad s_{m+1} = 0 \tag{7}$$

$$\Delta_j \;\geq\; 0 \qquad \forall\, 0 \leq j \leq m(i). \tag{8}$$

The upper bound $\hat{R}_i$ on the job J_i worst-case response time is found by solving the above linear programming problem where Formula (5) is maximized with the constraints given by Formulas (6)–(8). The linear programming problems can be solved in polynomial time in the size of the input [52] (in our case $m(i) \leq m$).

The upper bound overestimation comes from the fact that the above-defined problem allows a single job to be processed simultaneously on different processors while, in reality, every job can run only upon a single processor at a time. A simple bound can be found for Δ_{i-1} when $i \leq m$. The longest time all $i-1$ processors can be occupied by $i-1$ jobs is given as $\Delta_{i-1} = \min_{j<i}\{C_j/s_j\}$. However, for the general case (Δ_j for $j \neq i-1$), finding how long j processors are occupied by $i-1$ jobs can be equivalent to deciding their schedulability that has been shown to be a NP-hard problem (Theorem 2.6 in [55]) for identical processors.

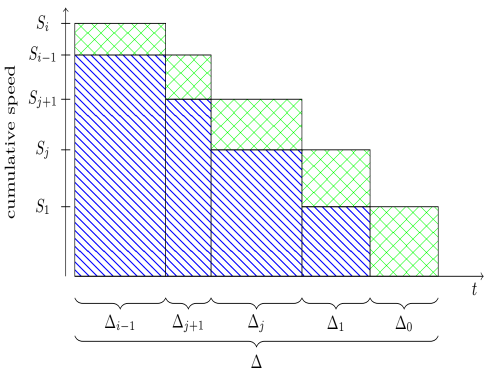
5.3 Special case: Dense schedule

Under certain circumstances, the worst-case scenario for computing the worst-case response time upper bound can be identified without solving the linear programming problem. We establish the relation between the computing capacities of particular processors for which the worst-case response time upper bound given by the solution of the linear problem defined with Formulas (5)–(8) has a specific form that we will refer to as *dense* schedule.

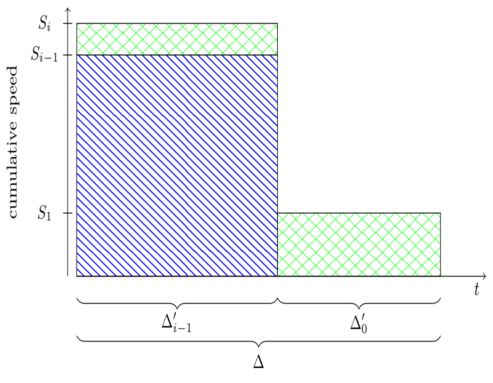
▶ **Definition 3** (Dense schedule)**.** *We say a schedule is dense when all higher-priority jobs $hp(i)$ execute simultaneously as much as possible, $\Delta_{m(i)} \geq 0$, $\Delta_0 \geq 0$, $\forall\, 0 < j < m(i) : \Delta_j = 0$ where $(\Delta_{m(i)}, \ldots, \Delta_0)$ is the solution of the linear problem defined with Formulas (5)–(8).*

▶ **Example 4.** Consider again the jobs running on the uniform processor from Example 2. Figure 1 (a) shows a dense schedule where $\Delta_3 = 7, \Delta_2 = \Delta_1 = 0$, and $\Delta_0 = 3$.

To ease the presentation, we assume $i > 1$. If $i = 1$, J_1 has the highest priority and always executes on the fastest processor. Figure 3 illustrates the proof strategy in Theorem 6 (for ease of the presentation, the figure assumes $1 < i \leq m$, but the theorem also considers the case when $i > m$). We first create two schedules. In the first schedule, the interference of higher-priority jobs $hp(i)$ (blue ▨ area) is distributed arbitrarily among all interference intervals $\Delta_1, \ldots, \Delta_{i-1}$. In the second schedule, the higher-priority jobs $hp(i)$ perfectly overlap, $\forall\, 0 < j < i-1 : \Delta_j = 0$, $\Delta_{i-1} = I_i / S_{i-1}$ where $I_i = \sum_{k \in hp(i)} C_k$ (blue ▨ area) forming a *dense* schedule. Then, we compare both schedules with respect to their worst-case interference and available execution for job J_i (green ▩ area).



(a) Arbitrary schedule $(\Delta_0, \ldots, \Delta_{i-1})$.



(b) Dense schedule $(\Delta'_0, \ldots, \Delta'_{i-1})$.

Figure 3 Two schedules from Theorem 6.

For the purpose of the next theorem, we introduce the following auxiliary definition:

▶ **Definition 5.** *For any $j > 1$ we define Ω_j as:*

$$\Omega_j = \frac{s_1 - s_j}{S_{j-1}} \tag{9}$$

with $s_j = 0$ and $S_j = S_m$ for $j > m$.

▶ **Theorem 6.** *Suppose that job J_i with static priority $i > 1$ is scheduled by a fixed-priority policy upon an m-uniform multiprocessor. If the following condition is satisfied:*

$$\Omega_i \;\geq\; \max_{1<j<i} \Omega_j \tag{10}$$

then the dense *schedule leads to the job J_i worst-case response time upper bound $\hat{R}_i > 0$.*

Proof. We first assume that $i \leq m+1$. We proceed by contradiction. Suppose that a dense schedule does not lead to the job J_i worst-case response time upper bound.

Let $(\Delta_0, \ldots, \Delta_{i-1})$ be a schedule of $hp(i)$ jobs within the time interval $[0, \hat{R}_i]$ and assume that $(\Delta_0, \ldots, \Delta_{i-1})$ is not a dense schedule and no other schedule than $(\Delta_0, \ldots, \Delta_{i-1})$ gives a higher worst-case response time upper bound for job J_i. We define I_i as the total execution requirement of $hp(i)$ jobs within $\hat{R}_i$: $I_i = \min\{\sum_{k \in hp(i)} C_k, \hat{R}_i \cdot S_{i-1}\}$. Suppose that the execution units of $hp(i)$ jobs within each interference time Δ_j of j processors for $j \leq i-1$ are given by $\alpha_j \cdot I_i$ where $\alpha_j \in [0,1]$ and $\sum_{j=1}^{i-1} \alpha_j = 1$. The lengths of interference times are:

$$\begin{cases} \Delta_0 = \hat{R}_i - \sum_{j=1}^{i-1} \alpha_j \cdot I_i / S_j \\ \Delta_1 = \alpha_1 \cdot I_i / S_1 \\ \quad \ldots \\ \Delta_{i-2} = \alpha_{i-2} \cdot I_i / S_{i-2} \\ \Delta_{i-1} = \alpha_{i-1} \cdot I_i / S_{i-1}. \end{cases}$$

Let $(\Delta'_0, \ldots, \Delta'_{i-1})$ be a dense schedule of $hp(i)$ jobs within time interval $[0, \hat{R}_i]$:

$$\begin{cases} \Delta'_0 = \hat{R}_i - I_i / S_{i-1} \\ \Delta'_1 = 0 \\ \quad \ldots \\ \Delta'_{i-2} = 0 \\ \Delta'_{i-1} = I_i / S_{i-1}. \end{cases}$$

The schedule $(\Delta_0, \ldots, \Delta_{i-1})$ results in the worst-case interference for J_i, hence the dense schedule $(\Delta'_0, \ldots, \Delta'_{i-1})$ cannot lead to higher interference. According to Formula (7), it must hold that:

$$\sum_{j=0}^{i-1} s_{j+1} \cdot \Delta'_j \; > \; \sum_{j=0}^{i-1} s_{j+1} \cdot \Delta_j$$

We rewrite the above inequality as follows:

$$\underbrace{I_i / S_{i-1}}_{\Delta'_{i-1}} \cdot s_i + \underbrace{(\hat{R}_i - I_i / S_{i-1})}_{\Delta'_0} \cdot s_1 \; > \; \sum_{j=1}^{i-1} \underbrace{(\alpha_j \cdot I_i / S_j)}_{\Delta_j} \cdot s_{j+1} + \underbrace{(\hat{R}_i - \sum_{j=1}^{i-1} \alpha_j \cdot I_i / S_j) \cdot s_1}_{\Delta_0}.$$

We rearrange the terms:

$$\sum_{j=1}^{i-1} \alpha_j \cdot \left(\frac{s_1 - s_{j+1}}{S_j} \right) \; > \; \frac{s_1 - s_i}{S_{i-1}}.$$

Finally, we replace the terms on the left- and on the right-hand side using Equation (9) and shift the index j by one (*i.e.,* we iterate from 2 to i instead of from 1 to $i-1$):

$$\sum_{j=2}^{i} \alpha_{j-1} \cdot \Omega_j \; > \; \Omega_i.$$

Using Inequality (10), we can upper bound the left-hand side of the above relation as follows:

$$\sum_{j=2}^{i} \alpha_{j-1} \cdot \Omega_j \;\leq\; \Omega_i \cdot \sum_{j=2}^{i} \alpha_{j-1} \;=\; \Omega_i \cdot \sum_{j=1}^{i-1} \alpha_j \;=\; \Omega_i \cdot 1 \;=\; \Omega_i.$$

Comparing it to the right-hand of the previous relation gives a contradiction ($\Omega_i > \Omega_i$).

We now assume that $i > m+1$. The linear problem defined in Formulas (5)–(8) has the same number of constraints and variables Δ_j for $i = m+1$, considered in the first part of the proof, and for every $i > m+1$. The upper bound computation takes as the input the cumulative interference from all higher-priority jobs (*i.e.,* it is ignored how this interference is distributed among the particular jobs). We note that $\Omega_{m+1} = s_1/S_m$ and for any $j > m+1$ $\Omega_j = \Omega_{m+1}$ by Definition 5. Hence, the proof is equivalent to the proof for $i = m+1$ from the first part of the proof. ◀

If for some job J_i the conditions of Theorem 6 do not hold, but there exists $i' > i$ for which the conditions hold, it can be pessimistically assumed that job J_i has priority i' and can run upon $i' - i$ slower processors. The set of higher-priority jobs $hp(i)$ should remain unchanged.

5.4 Dense schedule in two-speed platforms

The worst-case scenario for computing the worst-case response time upper bound can also be simplified using the *dense schedule* for *two-speed* platforms with only two different processor speeds. In *ARM big.LITTLE* (now succeeded by *DynamIQ*) such architecture combines "*fast*" high-power cores with "*slow*" low-power cores.

In two-speed platforms, the higher-priority interfering jobs $hp(i)$, at any time instant, can run either on all fastest processors or all available processors. That is, if the higher-priority jobs $hp(i)$ run on all but one fastest processor, the job J_i can run on the fastest processor without any interference. If the higher-priority jobs $hp(i)$ run on all but one processor (among all the available processors), the job J_i can run on the slowest processor without any interference from the higher-priority jobs $hp(i)$ running on the other slowest processors.

Let m_1 be the number of fastest processors. Job J_i with m_1-th priority or higher, cannot be interfered by another $hp(i)$ job and will always execute with no interference. Job J_i with priority i such that $m_1 < i \leq m$ can only be interfered by the $hp(i)$ jobs running on the fastest processors. Hence, its worst-case response time occurs in a *dense* schedule.

In what follows, we focus on $i > m$ and search for such two-speed multiprocessors where a *dense* schedule leads to the worst-case response time upper bound. As in Theorem 6, we will create two schedules, an arbitrary schedule and a dense schedule, to compare them with respect to their worst-case interference and available execution for job J_i. Figure 4 shows the arbitrary schedule for the case of two-speed platforms.

▶ **Theorem 7.** *Suppose that job J_i with static priority $i > m$ is scheduled by a fixed-priority policy upon an m-uniform two-speed multiprocessor with m_1 s_1-speed and m_2 s_m-speed processors where $s_1 > s_m$ and $m_1 + m_2 = m$. If the following condition is satisfied:*

$$\frac{m_1}{m_2} \geq 1 - \frac{s_m}{s_1} \tag{11}$$

then the dense *schedule leads to the job J_i worst-case response time upper bound $\hat{R}_i > 0$.*

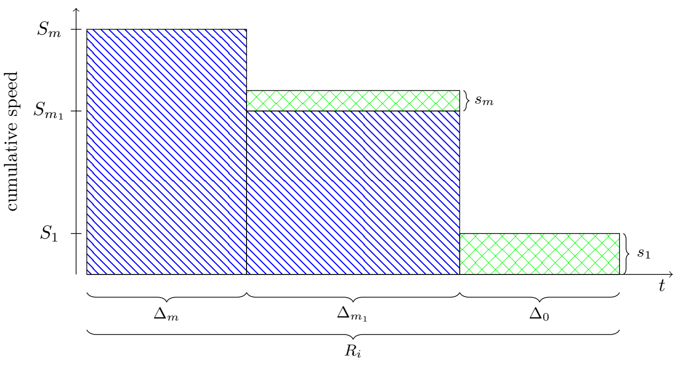


Figure 4 Execution of higher-priority jobs $hp(i)$ ▨ and job J_i $(i > m)$ under analysis ▩ upon a two-speed multiprocessor (with m_1 fastest processors).

Proof. We proceed by contradiction. Suppose that a dense schedule does not lead to the job J_i worst-case response time upper bound. All m_1 fastest processors have the same processing speed: $s_1 = s_2 = \ldots = s_{m_1}$, and the remaining m_2 processors (slow) have the same processing speed: $s_{m_1+1} = s_{m_1+2} = \ldots = s_m$. There are three interference times: Δ_0 where no interfering job is running, Δ_{m_1} where the interfering jobs are running only on m_1 fastest processors, and Δ_m where the interfering jobs are running on all available processors. Let $(\Delta_0, \Delta_{m_1}, \Delta_m)$ be a schedule of $hp(i)$ jobs within the time interval $[0, \hat{R}_i]$ and assume that $(\Delta_0, \Delta_{m_1}, \Delta_m)$ is not a dense schedule and no other schedule than $(\Delta_0, \Delta_{m_1}, \Delta_m)$ gives a higher worst-case response time upper bound for job J_i. We define I_i as the total execution requirement of $hp(i)$ jobs within $\hat{R}_i$: $I_i = \min\{\sum_{k \in hp(i)} C_k, \hat{R}_i \cdot S_m\}$. Suppose that $\alpha_{m_1} \cdot I_i$ of higher-priority $hp(i)$ workload occurs within interference time Δ_{m_1} and $\alpha_m \cdot I_i$ within interference time Δ_m where $\alpha_{m_1}, \alpha_m \in [0, 1]$ and $\alpha_{m_1} + \alpha_m = 1$. The lengths of interference times are:

$$\begin{cases} \Delta_0 = \hat{R}_i - (\alpha_{m_1} \cdot I_i/S_{m_1} + \alpha_m \cdot I_i/S_m) \\ \Delta_{m_1} = \alpha_{m_1} \cdot I_i/S_{m_1} \\ \Delta_m = \alpha_m \cdot I_i/S_m. \end{cases}$$

Let $(\Delta'_0, \Delta'_{m_1}, \Delta'_m)$ be a dense schedule of $hp(i)$ jobs within time interval $[0, \hat{R}_i]$:

$$\begin{cases} \Delta'_0 = \hat{R}_i - I_i/S_m \\ \Delta'_{m_1} = 0 \\ \Delta'_m = I_i/S_m. \end{cases}$$

The schedule $(\Delta_0, \ldots, \Delta_{i-1})$ results in the worst-case interference for J_i, hence the dense schedule $(\Delta'_0, \ldots, \Delta'_{i-1})$ cannot lead to higher interference. According to Formula (7):

$$\sum_{j=0}^{m} s_{j+1} \cdot \Delta'_j > \sum_{j=0}^{m} s_{j+1} \cdot \Delta_j.$$

We rewrite the above inequality as follows:

$$\underbrace{(\hat{R}_i - I_i/S_m)}_{\Delta'_0} \cdot s_1 > \underbrace{(\alpha_{m_1} \cdot I_i/S_{m_1})}_{\Delta_{m_1}} \cdot s_m + \underbrace{(\hat{R}_i - (\alpha_{m_1} \cdot I_i/S_{m_1} + \alpha_m \cdot I_i/S_m))}_{\Delta_0} \cdot s_1$$

$$-s_1/S_m \;>\; (\alpha_{m_1} \cdot s_m/S_{m_1}) - (\alpha_{m_1} \cdot s_1/S_{m_1} + \alpha_m \cdot s_1/S_m)$$
$$\frac{s_1}{S_m} \cdot (\alpha_m - 1) \;>\; \frac{\alpha_{m_1}}{S_{m_1}} \cdot (s_m - s_1)\,.$$

By substituting $\alpha_{m_1} + \alpha_m = 1$:

$$\frac{s_1}{S_m} \cdot (-\alpha_{m_1}) \;>\; \frac{\alpha_{m_1}}{S_{m_1}} \cdot (s_m - s_1)$$

$$\frac{s_1}{S_m} + \frac{s_m - s_1}{S_{m_1}} \;<\; 0.$$

By substituting $S_m = m_1 \cdot s_1 + m_2 \cdot s_m$ and $S_{m_1} = m_1 \cdot s_1$:

$$\frac{s_1}{m_1 \cdot s_1 + m_2 \cdot s_m} + \frac{s_m - s_1}{m_1 \cdot s_1} \;<\; 0$$

$$m_1 \cdot s_1^2 + m_1 \cdot s_1 \cdot s_m + m_2 \cdot s_m^2 - m_1 \cdot s_1^2 - m_2 \cdot s_1 \cdot s_m < 0$$

$$m_1 \cdot s_1 + m_2 \cdot s_m - m_2 \cdot s_1 < 0$$

$$\frac{m_1}{m_2} \;<\; \frac{s_1 - s_m}{s_1}.$$ ◀

In particular, we can easily show that for any two-speed multiprocessor with the same number of fast and slow processors or with more fast processors, Theorem 7 is always true.

▶ **Corollary 8.** *On a two-speed uniform multiprocessor with no fewer fast than slow processors, the dense schedule always leads to the job J_i worst-case response time upper bound.*

Proof. For $m_1 \geq m_2$, Formula (11) gives $\frac{s_m}{s_1} \geq 0$ and Theorem 7 is always true. ◀

5.5 Response time computation for dense schedule

We now compute the worst-case response time upper bound $\hat{R}_i$ of job J_i that executes on a uniform multiprocessor system that fulfills the conditions of Theorems 6 and 7. Consider again Figure 3 (b). The total interference of higher-priority jobs $hp(i)$ is denoted by $I_i = \sum_{k \in hp(i)} C_k$. We consider two cases with respect to the job J_i priority: $i \leq m$ and $i > m$. If $i > m$, we define $\Delta_m = I_i/S_m$ and $\Delta_0 = C_i/s_1$. During interval Δ_m, job J_i is completely blocked, and during interval Δ_0, job J_i executes on the fastest processor with the computation speed s_1. If $i \leq m$, we define $\Delta_{i-1} = \min\{I_i/S_{i-1}, C_i/s_i\}$ and $\Delta_0' = (C_i - \Delta_{i-1} \cdot s_i)/s_1$. During interval Δ_{i-1}, job J_i executes with computation speed s_i, and during time interval Δ_0', job J_i, if not completed, executes the remainder of its workload on the fastest processor having the computation speed s_1.

$$\hat{R}_i \;=\; \begin{cases} \Delta_m \;+\; \Delta_0 & \text{if } i > m, \\ \Delta_{i-1} + \;\Delta_0' & \text{otherwise.} \end{cases} \tag{12}$$

The time complexity of the above upper bound computation is bounded by the sum computation in I_i. It can have at most $n - 1$ elements where n is the number of jobs. The worst-case time complexity is, therefore, $\mathcal{O}(n)$. Verifying the conditions of Theorem 6 (*i.e.*, computing Ω_j using Equation (9) and checking Formula (10)) can involve m steps using the fact that $S_j = S_{j-1} + s_j$.

5.6 Response time analysis for sporadic tasks

We search for an upper bound $\hat{R}_i$ on the sporadic task τ_i worst-case response time R_i.

We first upper bound the interference generated by the higher-priority tasks $\tau_k \in hp(i)$ within any generic time interval. Then, we insert this interference into the linear programming problem described in the previous section. We check whether a given time interval is large enough to accommodate the execution requirement. In the first schedulability test (*Single*), the length of the time interval is fixed to the relative deadline of the task under analysis, and in the second schedulability test (*RTA*), iteratively increased.

5.6.1 Upper bound on sporadic task workload

We compute an upper bound on the workload that a sporadic task $\tau_k \in hp(i)$ can execute in a generic time interval of length $\Delta > 0$. We extend the state-of-the-art workload computation methods derived for identical multiprocessors [9, 14, 21]. In particular, we modify the upper bound on the task start time by taking into account processor speeds.

Figure 5 illustrates our approach. We assume that task τ_k is schedulable and that we know its worst-case response time upper bound $\hat{R}_k$. The maximum possible τ_k workload within a time interval of length Δ is when the first τ_k job (also called *carry-in* job) starts as late as possible after its release, and all successive τ_k jobs are released as soon as possible. The interval starts when the *carry-in* job starts its execution. The latest start time of task τ_k job on uniform multiprocessor can be upper bounded by:

$$\delta_k = \hat{R}_k - C_k/s_1 \tag{13}$$

(*i.e.,* we assume that the job executes on the fastest processor uninterruptedly from its start to its end). The last τ_k job released within the interval (also called *carry-out* job) might not entirely fit the interval.

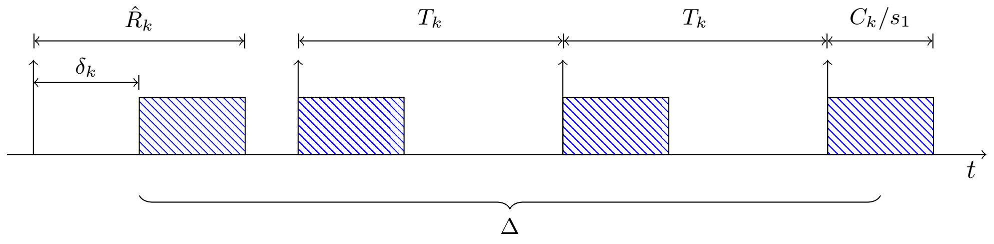


Figure 5 Sporadic task τ_k maximum workload in generic time interval of length $\Delta > 0$.

The contribution of τ_k *body* jobs (*i.e.,* jobs that are released after carry-in and before carry-out jobs) and the carry-in job within a time interval of Δ can be upper bounded by:

$$\left\lfloor \frac{\Delta + \delta_k}{T_k} \right\rfloor \cdot C_k.$$

The *carry-out* job contribution on uniform multiprocessor is at most:

$$\min\left\{C_k,\, s_1 \cdot ((\Delta + \delta_k) \bmod T_k)\right\}.$$

An upper bound on the task τ_k workload including a carry-in job within a generic time interval of length $\Delta > 0$ can be then expressed as:

$$I_k^{CI}(\Delta) \;=\; \left\lfloor \frac{\Delta + \delta_k}{T_k} \right\rfloor \cdot C_k \;+\; \min\left\{C_k,\, s_1 \cdot ((\Delta + \delta_k) \bmod T_k)\right\}. \tag{14}$$

An upper bound on the task τ_k workload without carry-in job within a generic time interval of length $\Delta > 0$ is given by:

$$I_k^{NC}(\Delta) \; = \; \left\lfloor \frac{\Delta}{T_k} \right\rfloor \cdot C_k \; + \; \min\left\{C_k, \, s_1 \cdot (\Delta \bmod T_k)\right\}. \tag{15}$$

We can limit the number of carry-in jobs using the approach proposed in [12,46]. A work-conserving algorithm does not leave any processor idle when there is a job ready to execute. Let t_0 denote the beginning of the interval of interest when the job under analysis is released and starts its execution that finishes at its worst-case response time. If all processors are busy executing higher-priority jobs at t_0, we can move left the time instant t_0 to the the first time instant when there is at least one processor idle or executing a lower- or equal-priority job. By doing so, the response time of the job under analysis will not decrease (*i.e.,* the job cannot execute as all the processors are busy). Consequently, at the time instant t_0 the number of the higher-priority active jobs cannot be more than $m-1$ for $i > m$ and $\max\{0, i-2\}$ otherwise. We define the maximum number of carry-in jobs as:

$$\mathcal{C}(i) \; = \; \max\left\{0, \min\left\{m-1, i-2\right\}\right\} \; = \; \max\left\{0, m(i)-1\right\}. \tag{16}$$

The cumulative interference of $hp(i)$ tasks within generic time interval of length Δ can be upper bounded by:

$$I^i(\Delta) \; = \; \sum_{k \in hp(i)} I_k^{NC}(\Delta) + \sum_{k \in hp(i)}^{\mathcal{C}(i) \text{ largest}} \left(I_k^{CI}(\Delta) - I_k^{NC}(\Delta)\right). \tag{17}$$

5.6.2 Worst-case response time upper bound computation

We extend the linear programming problem for computation of the worst-case response time upper bound for aperiodic jobs defined with Formulas (5)–(8) to recurrent tasks. We derive two schedulability tests: the first (*Single*) checks task τ_i schedulability in interval $\Delta = D_i$, and the second (*RTA*) increments iteratively interval Δ until it is large enough to execute task τ_i and all higher-priority interfering tasks $hp(i)$.

Below, the linear programming problem for computation of the task τ_i worst-case response time upper bound $\hat{R}_i$ for periodic and sporadic tasks over time interval $\Delta > 0$ is given (Formulas (18)–(21)). Compared to the linear programming problem for aperiodic jobs, we replace on the right-hand side of the first constraint, Formula (19), the workload of the higher-priority jobs with the workload of the higher-priority sporadic tasks that can be generated within a generic time interval Δ. Their cumulative interference is computed with Equation (17) derived in the previous subsection. If the resulting worst-case response time upper bound $\hat{R}_i$ is less than Δ, we can conclude that $\hat{R}_i$ upper bounds the worst-case response time R_i.

$$\text{maximize:} \quad \hat{R}_i \; = \; \sum_{j=0}^{m(i)} \Delta_j \tag{18}$$

$$\text{subject to:} \quad \sum_{j=1}^{m(i)} S_j \cdot \Delta_j \quad \leq \quad I^i(\Delta) \tag{19}$$

$$\sum_{j=0}^{m(i)} s_{j+1} \cdot \Delta_j \; = \; C_i \quad \text{with} \quad s_{m+1} = 0 \tag{20}$$

$$\Delta_j \; \geq \; 0 \qquad \forall \, 0 \leq j \leq m(i). \tag{21}$$

Single-interval test. Based on the above formulation, we propose our first schedulability test, *Single-interval*, for sporadic and periodic tasks running upon a uniform multiprocessor. For each task τ_i, in decreasing priority order, we solve the linear programming problem defined by Formulas (18)–(21) for $\Delta = D_i$. If the resulting worst-case response time upper bound $\hat{R}_i$ is less than or equal to D_i, we conclude that the task is schedulable and move to the next task τ_{i+1}. As we are proceeding in tasks decreasing priority order, we can use the previously obtained $\hat{R}_i$ to upper bound τ_i latest start time δ_i (see Equation (13)) needed for $hp(i+1)$ interference calculation. The overall running time of the test is bounded by the complexity of the linear programming problem, which is known to be polynomial [52].

Response time analysis. The second proposed approach, *response time analysis* (*RTA*), solves for each task τ_i, in decreasing priority order, the linear programming problem defined by Formulas (18)–(21) by increasing Δ until the obtained worst-case response time upper bound $\hat{R}_i$ is less than or equal to Δ. The schedulability test for task τ_i is given in Algorithm 1. In line 7, we increment Δ to the value of the closest integer greater than or equal to the previously obtained solution. We check only discrete values. The solution $\hat{R}_i$ is the task τ_i worst-case response time upper bound if the time interval length Δ is greater than or equal to $\hat{R}_i$. Algorithm 1 has pseudo-polynomial time complexity as it calls a polynomial-time procedure to solve the linear programming problem at most D_i times.

Algorithm 1 Worst-case response time upper bound using linear programming for sporadic tasks.

```
1: $\Delta \leftarrow C_i/s_1$
2: while $\Delta \leq D_i$ do
3:     $\hat{R}_i \leftarrow$ Solve LP given by Formulas (18)–(21)
4:     if $\hat{R}_i \leq \Delta$ then
5:         return $\hat{R}_i$
6:     end if
7:     $\Delta \leftarrow \lceil \hat{R}_i \rceil$
8: end while
```

In both tests, if Theorem 6 or 7 conditions can be verified (*dense* schedule), the linear programming problem can be replaced with a respective method.

5.6.3 Optimal priority assignment

A priority assignment policy $\mathcal{P}$ is optimal with respect to a schedulability test $\mathcal{S}$, if and only if there are no task sets that are deemed schedulable by test $\mathcal{S}$ using another priority assignment policy but not deemed schedulable by test $\mathcal{S}$ using policy $\mathcal{P}$ [33]. An *Optimal Priority Assignment* (*OPA*) algorithm was proposed for uniprocessor fixed-priority feasibility analysis by Audsley [5]. Later, it was proved in [31] that Audsley's OPA is compatible with a global multiprocessor fixed-priority schedulability test $\mathcal{S}$ if and only if the following three conditions hold:

▶ **Condition 1.** *The schedulability of a task τ_k may, according to test $\mathcal{S}$, depend on any independent properties of tasks with priorities higher than k, but not on any properties of those tasks that depend on their relative priority ordering.*

▶ **Condition 2.** *The schedulability of a task τ_k may, according to test $\mathcal{S}$, depend on any independent properties of tasks with priorities lower than k, but not on any properties of those tasks that depend on their relative priority ordering.*

▶ **Condition 3.** *When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test $\mathcal{S}$, if it was previously schedulable at the lower priority.*

According to the above three conditions, [31] classifies global fixed-priority schedulability tests into OPA-compatible and OPA-incompatible. Next, we show that the proposed schedulability test *Single* and *RTA* are OPA-compatible by replacing $\hat{R}_k$ by D_k in the calculation of the latest start time of interference task τ_k (Equation (13)), *i.e.*,

$$\delta_k = D_k - C_k/s_1. \tag{22}$$

We note that the latest start time given by (22) is more pessimistic than (13) since $\hat{R}_k \leq D_k$. However, it is necessary to use (22) instead of (13) for OPA-compatibility, since the response time $\hat{R}_k$ is dependent on the relative priority ordering (violating Condition 1), while the deadline D_k is an independent task property. Therefore, we derive two new variants of the proposed schedulability tests, namely *Single-OPA* and *RTA-OPA*, by replacing the latest start time calculation (13) with (22) in *Single* and *RTA*, respectively. We also note that one can first use OPA (*Single-OPA* or *RTA-OPA*) to derive task priorities and then apply the derived priorities to the original test (*RTA* or *Single*) with (13).

▶ **Theorem 9.** *The schedulability tests Single-OPA and RTA-OPA are OPA-compatible.*

Proof. It suffices to show that Conditions 1-3 hold. Formulas (14)-(22) depend on the set of higher-priority tasks, not on their relative priority ordering; hence Condition 1 holds. Moreover, these formulas have no dependency on lower-priority tasks; hence, Condition 2 holds. Consider two tasks τ_i and τ_j initially with priorities k and $k+1$, respectively (τ_i has higher priority than τ_j). The upper bound response time of task τ_j cannot increase when it is shifted up one priority level to priority k, as the only change in the response time computation is the removal of task τ_i from the set of tasks that have higher priority than task τ_j; hence Condition 3 holds. ◀

In what follows, we denote with *Single* and *RTA* the corresponding schedulability tests with *Rate Monotonic* priorities, while *Single-OPA* and *RTA-OPA* the tests with priorities derived by OPA.

6 Evaluation

In this section, we present experimental results on randomly generated task sets to evaluate the effectiveness of the schedulability analyses proposed in Section 5.

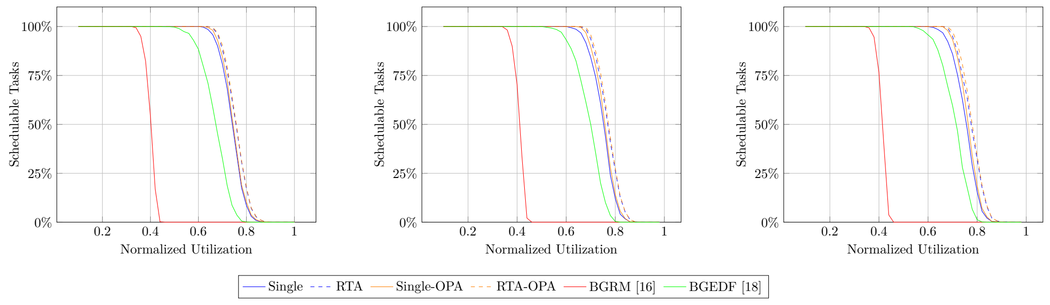
6.1 Evaluation settings

We generate synthetic task sets with three varied parameters: number of tasks $n \in \{8, 16\}$, processor speed configurations $[s_1, ..., s_m]$ given in Table 1, and task set utilization $U \in \{0.01 \cdot S_m, 0.02 \cdot S_m, \ldots, 0.99 \cdot S_m, S_m\}$. Given a parameter combination, a task set is generated according to the following procedures. First, we use the *Dirichlet-Rescale* (*DRS*) algorithm [44], a generalized version of the *UUnifast* [23] and *RandFixedSum* [37] algorithms, to generate random task-utilization values $[U_1, U_2, \ldots, U_n]$ that sum to the requested total utilization $U = \sum_{i=1}^{n} U_i$. Then, we randomly generate period T_i of each task τ_i in the range $[10\,000, 100\,000]$ microsecond and obtain its worst-case execution time as $C_i = U_i \cdot T_i$. The generated tasks are assumed to have implicit deadlines.

Table 1 Processor speeds $[s_1, \ldots, s_m]$.

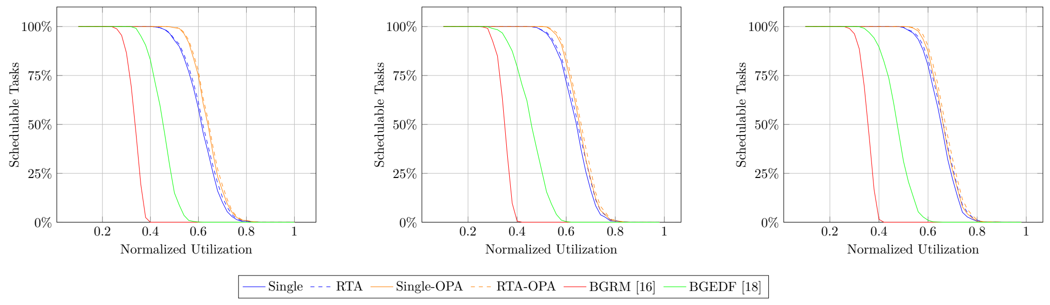
$m = 2$	$m = 4$	$m = 8$
$[2, 1]$	$[2, 2, 1, 1]$	$[2, 2, 2, 2, 1, 1, 1, 1]$
$[3, 1]$	$[3, 2, 2, 1]$	$[3, 3, 2, 2, 2, 2, 1, 1]$
$[4, 1]$	$[4, 3, 2, 1]$	$[4, 4, 3, 3, 2, 2, 1, 1]$

We evaluate the proposed schedulability tests *Single*, *RTA*, *Single-OPA*, and *RTA-OPA* with the schedulability tests for Rate Monotonic (*BGRM*) [16] and Earliest Deadline First (*BGEDF*) [18] proposed by Baruah and Goossens. For each parameter combination, we generate 2,000 task sets, as described above and evaluate their schedulability using the comparison schedulability tests. The tests were implemented using *Schedcat* Python/C++ library, and the linear programming problem was solved using Gurobi [48] solver version 11.0.0.



(a) 2-1 **(b)** 3-1 **(c)** 4-1

Figure 6 Schedulability ratios for m=2 processors and n=8 tasks.



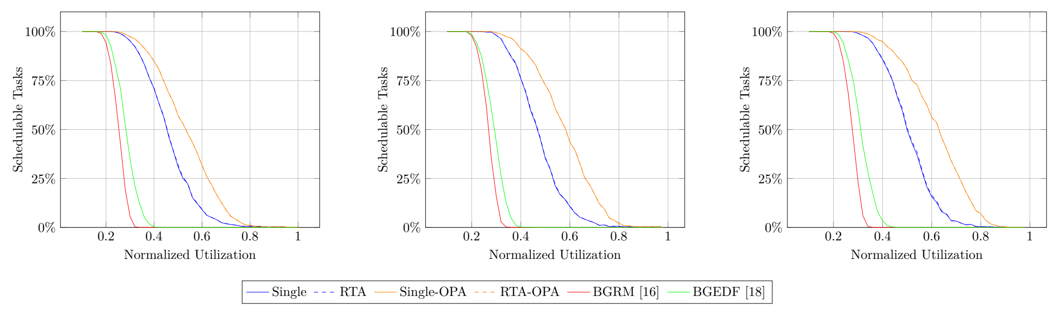
(a) 2-2-1-1 **(b)** 3-2-2-1 **(c)** 4-3-2-1

Figure 7 Schedulability ratios for m=4 processors and n=8 tasks.

6.2 Evaluation results

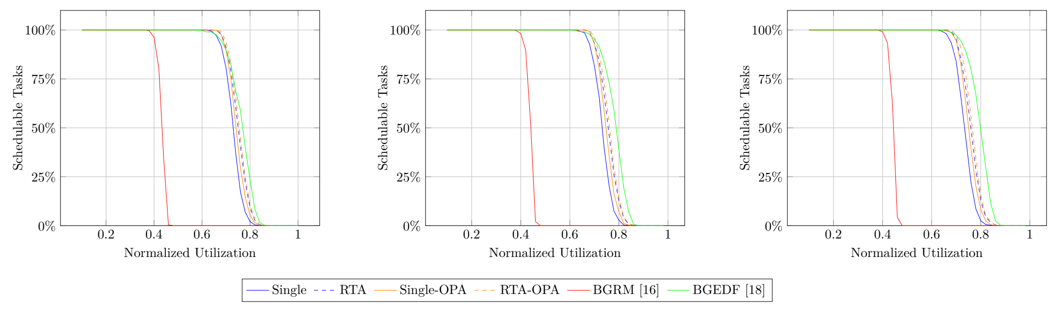
The percentage of schedulable task sets are shown in Figures 6, 7, 8, respectively, for $m = 2, 4, 8$ with $n = 8$ and Figures 9, 10, 11, respectively, for $m = 2, 4, 8$ with $n = 16$, as a function of normalized utilization U/S_m. The results support the following observations.

▶ **Observation 10.** *The state-of-the-art EDF test BGEDF [18] performs better than the RM test BGRM [16] on all problem settings. However, the performance gap decreases as the number of processors increases.*



(a) 2-2-2-2-1-1-1-1 **(b)** 3-3-2-2-2-2-1-1 **(c)** 4-4-3-3-2-2-1-1

Figure 8 Schedulability ratios for m=8 processors and n=8 tasks.



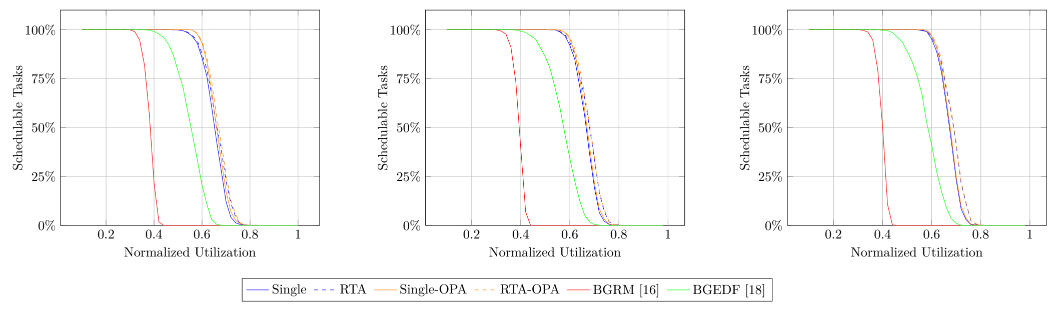
(a) 2-1 **(b)** 3-1 **(c)** 4-1

Figure 9 Schedulability ratios for m=2 processors and n=16 tasks.

▶ **Observation 11.** *The proposed RM tests Single and RTA outperform the state-of-the-art RM test BGRM [16] on all problem settings. This is expected as the state-of-the-art tests are based on the analytical bounds of the linear programming problem, which is solved by our tests without any further transformation and precision loss. Moreover, it performs better than the state-of-the-art EDF test BGEDF [18] on every problem setting except for $m = 2, n = 16$.*

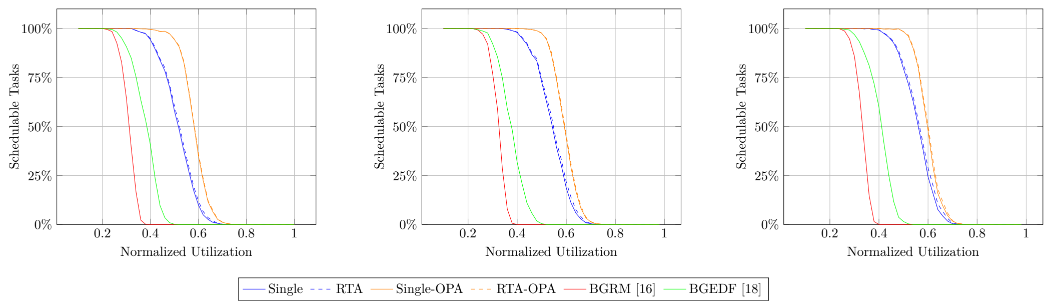
▶ **Observation 12.** *The improvement achieved by OPA (RTA-OPA/Single-OPA vs RTA/Single) is more significant for task sets with more processors.*

▶ **Observation 13.** *The improvement achieved by using the fixed-point iteration technique (RTA/RTA-OPA vs Single/Single-OPA) is more significant for task sets with fewer processors.*



(a) 2-2-1-1 **(b)** 3-2-2-1 **(c)** 4-3-2-1

Figure 10 Schedulability ratios for m=4 processors and n=16 tasks.



(a) 2-2-2-2-1-1-1-1 **(b)** 3-3-2-2-2-2-1-1 **(c)** 4-4-3-3-2-2-1-1

Figure 11 Schedulability ratios for m=8 processors and n=16 tasks.

7 Conclusions

In this paper, we proposed the response time analysis for sporadic constrained-deadline tasks executed by a global fixed-priority preemptive scheduler upon a uniform multiprocessor platform. The analysis leverages the linear programming model to upper bound the impact of the cumulative interference of the higher-priority tasks. We also observed the relation between processors' computational speeds, leading to the predefined worst-case schedules, especially in certain two-speed platforms commonly used in embedded systems. Based on the analysis, we propose two schedulability tests with different time complexities and their OPA-compatible version to exploit better priority assignment. Experimental evaluation based on synthetic task sets demonstrates the effectiveness of the proposed approaches.

The key limiting factor of any global scheduling policy is the cost of thread migration [20, 24,59,60,62]. To partially alleviate this cost, the next generation of *big.LITTLE* heterogeneous computing architecture, *ARM DynamIQ*, provides the shared L3 cache to enable simplified and more rapid thread migration between the processors [3]. In light of this, our future work will seek to apply the proposed analysis to limited preemption policies [26, 28, 63].

The proposed test can also be further improved by adding the constraints on the time during which a given number of processors is busy and by tightening the carry-in interference bounds [45]. The work can also be extended to other scheduling policies (*e.g.,* global *EDF*).

References

1 Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems*, 2021. `doi:10.1007/s11241-021-09376-1`.

2 Björn Andersson and Jan Åke Jönsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proceedings of the WIP session of IEEE Real-Time Systems Symposium (RTSS)*, 2000.

3 ARM. ARM DynamIQ Shared Unit. Technical Reference Manual. Revision: r0p2. `https://developer.arm.com/documentation/100453/latest/`. Accessed: 2024-05-09.

4 Neil C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS-164, Department of Computer Science, University of York, 1991.

5 Neil C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001. `doi:10.1016/S0020-0190(00)00165-4`.

6 Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tindell, and Andy J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2–3):173–198, 1995. doi:10.1007/BF01094342.

7 Neil C. Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292(8), 1993. doi:10.1049/sej.1993.0034.

8 Neil C. Audsley, Alan Burns, Mike Richardson, and Andy Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991. doi:10.1016/S1474-6670(17)51283-5.

9 Theodore P. Baker. Multiprocessor EDF and Deadline Monotonic schedulability analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 120–129, 2003. doi:10.1109/REAL.2003.1253260.

10 Sanjoy Baruah. Scheduling periodic tasks on uniform multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 7–13, 2000. doi:10.1109/EMRTS.2000.853986.

11 Sanjoy Baruah. Robustness results concerning EDF scheduling upon uniform multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 95–102, 2002. doi:10.1109/EMRTS.2002.1019189.

12 Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 119–128, 2007. doi:10.1109/RTSS.2007.35.

13 Sanjoy Baruah. An improved global EDF schedulability test for uniform multiprocessors. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 184–192, 2010. doi:10.1109/RTAS.2010.11.

14 Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated, 2015.

15 Sanjoy Baruah and Joël Goossens. Deadline monotonic scheduling on uniform multiprocessors. In Theodore P. Baker, Alain Bui, and Sébastien Tixeuil, editors, *Principles of Distributed Systems*, 2008. doi:10.1007/978-3-540-92221-6_8.

16 Sanjoy Baruah and Joël Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 52(7):966–970, 2003. doi:10.1109/TC.2003.1214344.

17 Sanjoy Baruah and Joël Goossens. Rate-monotonic scheduling on uniform multiprocessors. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, pages 360–366, 2003. doi:10.1109/ICDCS.2003.1203485.

18 Sanjoy Baruah and Joël Goossens. The EDF scheduling of sporadic task systems on uniform multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 367–374, 2008. doi:10.1109/RTSS.2008.32.

19 Sanjoy Baruah, Aloysius Mok, and Louis Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 182–190, 1990. doi:10.1109/REAL.1990.128746.

20 Andrea Bastoni, Bjorn B. Brandenburg, and James H. Anderson. Is semi-partitioned scheduling practical? In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 125–135, 2011. doi:10.1109/ECRTS.2011.20.

21 Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160, 2007. doi:10.1109/RTSS.2007.31.

22 Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(4):553–566, 2009. doi:10.1109/TPDS.2008.129.

23 Enrico Bini and Giorgio Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, 2005. doi:10.1007/s11241-005-0507-9.

24 Björn B. Brandenburg and Mahircan Gül. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 99–110, 2016. doi:10.1109/RTSS.2016.019.

25 Giorgio Buttazzo. Rate Monotonic vs. EDF: Judgment Day. *Real-Time Systems*, 29:5–26, 2004. doi:10.1023/B:TIME.0000048932.30002.d9.

26 Giorgio Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013. doi:10.1109/TII.2012.2188805.

27 Jacek Błażewicz, Klaus Ecker, Erwin Pesch, Günter Schmidt, and Jan Węglarz. *Handbook on Scheduling: Models and Methods for Advanced Planning.* Springer-Verlag, 2007.

28 Bipasa Chattopadhyay and Sanjoy Baruah. Limited-preemption scheduling on multiprocessors. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 225–234, 2014. doi:10.1145/2659787.2659798.

29 Liliana Cucu and Joël Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 397–404, 2006. doi:10.1109/ETFA.2006.355388.

30 Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *ACM/IEEE Design Automation Conference (DAC)*, pages 278–283, 2007.

31 Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47:1–40, 2011. doi:10.1007/s11241-010-9106-5.

32 Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4), 2011. doi:10.1145/1978802.1978814.

33 Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64–82, 2016. doi:10.1016/j.sysarc.2016.04.002.

34 Robert I. Davis, Attila Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008. doi:10.1109/TC.2008.66.

35 Friedrich Eisenbrand and Thomas Rothvoß. Static-priority real-time scheduling: Response time computation is np-hard. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 397–406, 2008. doi:10.1109/RTSS.2008.25.

36 Pontus Ekberg and Sanjoy Baruah. Partitioned scheduling of recurrent real-time tasks. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 356–367, 2021. doi:10.1109/RTSS52674.2021.00040.

37 Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010. URL: https://www.cs.york.ac.uk/rts/static/papers/R:Emberson:2010a.pdf.

38 Awi Federgruen and Henry Groenevelt. Preemptive scheduling of uniform machines by ordinary network flow techniques. *Management Science*, 32(3):341–349, 1986. doi:10.1287/mnsc.32.3.341.

39 Shelby Funk and Sanjoy Baruah. Task assignment on uniform heterogeneous multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 219–226, 2005. doi:10.1109/ECRTS.2005.31.

40 Shelby Funk, Joël Goossens, and Sanjoy Baruah. On-line scheduling on uniform multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 183–192, 2001. doi:10.1109/REAL.2001.990609.

41 Teofilo Gonzalez. Optimal mean finish time preemptive schedules. In *Technical Report 220*. Computer Science Department, Pennsylvania State University Chichester, 1977.

42 Teofilo Gonzalez and Sartaj Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM (JACM)*, 25(1):92–101, 1978. doi:10.1145/322047.322055.

43 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979. doi:10.1016/S0167-5060(08)70356-X.

44 David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020. doi:10.1109/RTSS49844.2020.00018.

45 Nan Guan, Meiling Han, Chuancai Gu, Qingxu Deng, and Wang Yi. Bounding carry-in interference to improve fixed-priority global multiprocessor scheduling analysis. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 11–20, 2015. doi:10.1109/RTCSA.2015.9.

46 Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 387–397, 2009. doi:10.1109/RTSS.2009.11.

47 Arpan Gujarati, Felipe Cerqueira, and Björn B. Brandenburg. Multiprocessor real-time scheduling with arbitrary processor affinities: From practice to theory. *Real-Time Systems*, 51(4):440–483, 2015. doi:10.1007/s11241-014-9205-9.

48 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

49 Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976. doi:10.1145/321941.321951.

50 Edward C. Horvath, Shui Lam, and Ravi Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM (JACM)*, 24(1):32–43, 1977. doi:10.1145/321992.321995.

51 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000. doi:10.1145/347476.347479.

52 Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

53 Sylvain Lauzac, Rami Melhem, and Daniel Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *EUROMICRO Workshop on Real-Time Systems*, pages 188–195, 1998. doi:10.1109/EMWRTS.1998.685084.

54 John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, 1989. doi:10.1109/REAL.1989.63567.

55 Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982. doi:10.1016/0166-5316(82)90024-4.

56 Björn Lisper and Peter Mellgren. Response-time calculation and priority assignment with integer programming methods. In Eduardo Tovar and Christer Norström, editors, *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 13–16, 2001. URL: https://www.es.mdh.se/pdf_publications/282.pdf.

57 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

58 Lars Lundberg. Multiprocessor scheduling of age constraint processes. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 42–47, 1998. doi:10.1109/RTCSA.1998.726350.

59 Geoffrey Nelissen, Vandy Berten, Joël Goossens, and Dragomir Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 15–24, 2011. doi:10.1109/RTCSA.2011.57.

60 N. Saranya and R.C. Hansdah. Dynamic partitioning based scheduling of real-time tasks in multicore processors. In *IEEE International Symposium on Real-Time Distributed Computing*, pages 190–197, 2015. doi:10.1109/ISORC.2015.23.

61 Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28:101–155, 2004. doi:10.1023/B:TIME.0000045315.61234.1e.

62 Mayank Shekhar, Abhik Sarkar, Harini Ramaprasad, and Frank Mueller. Semi-partitioned hard-real-time scheduling under locked cache migration in multicore systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 331–340, 2012. doi:10.1109/ECRTS.2012.27.

63 Abhilash Thekkilakattil, Robert I. Davis, Radu Dobrin, Sasikumar Punnekkat, and Marko Bertogna. Multiprocessor fixed priority scheduling with limited preemptions. In *International Conference on Real-Time and Networks Systems (RTNS)*, pages 13–22, 2015. doi:10.1145/2834848.2834855.

64 Kecheng Yang and James Anderson. Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pages 30–39, 2014. doi:10.1109/ESTIMedia.2014.6962343.

65 Kecheng Yang and James Anderson. On the soft real-time optimality of global EDF on uniform multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 319–330, 2017. doi:10.1109/RTSS.2017.00037.

66 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 *IEEE Real-Time Systems Symposium (RTSS)*, pages 161–170, 2007. doi:10.1109/RTSS.2007.40.

67 Qi Zhu, Haibo Zeng, Wei Zheng, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4), 2013. doi:10.1145/2362336.2362352.