Priority Assignment for Global Fixed Priority Scheduling on Multiprocessors

Xuanliang Deng*, Shriram Raja*, Yecheng Zhao, Haibo Zeng
Virginia Polytechnic Institute and State University, Blacksburg, Virginia, United States

***Abstract*—Global fixed-priority (G-FP) scheduling is a widely applied scheduling policy for real-time systems running on multiprocessor platforms. The state-of-the-art in priority assignment for G-FP follows one of two approaches. The first is to use a simple heuristic for priority assignment that works with any (thus the most accurate) schedulability analysis. The second is to leverage Audsley's polynomial-time optimal priority assignment (OPA) algorithm, which can only accommodate a less accurate analysis that satisfies the compatibility conditions required by OPA. In this paper, we study this critical issue and present a novel algorithm. We first use the concept of response time estimation range to build a new priority assignment framework, which is optimal with a more accurate schedulability analysis than OPA since its compatibility conditions are much weaker than those of OPA. This new frontier on the second approach is then judiciously combined with the first approach to take advantage of both. We evaluate the effectiveness of the proposed algorithm with various task sets. Compared with existing approaches, our algorithm always achieves the highest acceptance ratio and can outperform them by 25% on average.**

***Index Terms*—Global Fixed Priority Scheduling, Response Time Analysis, Response Time Estimation Range, Optimization**

I. Introduction

THE widespread adoption of multiprocessors in real-time systems application has inspired many research studies on the scheduling policies for multiprocessor platforms. Among them, global fixed-priority (G-FP) scheduling is a popular choice due to its advantages of application-transparent task migration and load balancing across all processors [1]. In G-FP, each task is assigned with a static priority that applies to all its instances. At runtime, tasks are selected for execution based on their priorities, and they are allowed to execute on any processor and migrate from one processor to another [2].

The problem of priority assignment for G-FP faces two intertwined challenges [3]: (1) an analysis to check system schedulability, where the problem of exact schedulability for G-FP is proven to be NP-hard [4], and (2) the identification of the optimal priority order among a total of $n!$ possible ones, where n is the number of tasks. For the latter, Audsley's optimal priority assignment (OPA) algorithm [5] is a well-known efficient algorithm that only explores $O(n^2)$ priority orders. However, it is only "optimal" with respect to the analysis that satisfies its compatibility conditions: in particular the schedulability of a task only depends on the set of its higher-priority tasks (or the set of lower-priority tasks), but not on the relative orders among those tasks.

Based on their compatibility with OPA, the schedulability analyses for G-FP can be classified into two categories: (1) OPA-compatible tests or (2) OPA-incompatible ones. Deadline Analysis (DA test) [6] and the improved Deadline Analysis with Limited Carry-in (DA-LC) [2] are two examples under the first category, both of which use the deadline of higher priority tasks to bound their interferences. In contrast, the Response Time Analysis (RTA test) [7] and the improved RTA test with Limited Carry-in (RTA-LC) [8] utilize the response time upper bounds of higher-priority tasks, which depend on the relative order among them. Recently, a new analysis called EPE-ZLL is proposed to improve upon RTA-LC by excluding the parallel execution of higher priority tasks in the calculation of the interference [9]. RTA test, RTA-LC, and EPE-ZLL all violate the compatibility conditions of OPA.

Since many of the schedulability tests are not OPA-compliant, the current proposals for priority assignment in G-FP follow two different directions. The first is to use OPA together with an OPA-compliant analysis. This approach has to sacrifice accuracy in the schedulability analysis in order to leverage OPA. As far as we know, the most accurate analysis that is OPA-compliant is DA-LC [2]. Instead, the second approach settles with a suboptimal priority assignment algorithm such as Deadline Monotonic Priority Ordering (DMPO), Deadline Minus Computation Monotonic (D-CMPO), and DkC [2], [10], [11], in favor of accommodating an OPA-incompatible analysis that is much more accurate than DA-LC.

In this paper, we propose a novel algorithm for priority assignment in G-FP. We first push the frontier for the OPA-based approach, by developing an optimization framework that is optimal for a broader range of schedulability analysis than OPA. In other words, its compatibility conditions are much weaker than those of OPA, which makes it possible to work with RTA-LC, a significantly more accurate analysis than DA-LC. We observe that this new frontier for the first approach, called MITER (Maximum Unschedulable response Time Estimation Range) with RTA-LC, is now performing better than the best of the second approach (DkC with EPE-ZLL) under low system utilization. We then develop a hybrid algorithm that works in a similar way as OPA but can use any schedulability test. Specifically, while assigning priority at a certain level, we use MITER with RTA-LC to estimate the priority order of higher priority tasks

The authors with * contributed equally to this paper. Haibo Zeng is the corresponding author. Email: hbzeng@vt.edu

when the system utilization is low and use DkC with EPE-ZLL when the utilization is high. This judiciously combines the strengths of both approaches, achieving better schedulability than the two across all system settings.

We organize the rest of the paper as follows. Section II reviews the related work on G-FP. Section III introduces the system model. Section IV describes the concepts and properties related to the MITER optimization framework. Section V presents the proposed hybrid priority assignment algorithm for G-FP scheduling. Section VI conducts experiments to compare our approach with the state-of-the-art methods. Finally, Section VII concludes the paper.

II. Related Work

Here we only provide a review of the most recent and relevant research results for G-FP. For some of the more classical results, the readers are referred to [1], [12].

Compared to single-processor schedulability, the problem of exact schedulability for G-FP on a multiprocessor platform is far more challenging since the critical instant is unknown. As such, the research focus is to develop sufficient-only schedulability analyses and improve their accuracy. Andersson et al. propose a simple response time upper bound for tasks with constrained deadlines [13]. Bertogna et al. develop the DA test [6] where the interference from a carry-in job is upper bounded using its deadline. RTA test instead uses the carry-in job's worst-case response time to bound its interference [7]. Guan et al. propose RTA-LC [8], an improvement over RTA test by bounding the number of carry-in jobs using the idea from Baruah [14]. Davis et al. [2] use the same idea to improve DA test and derive DA-LC. Zhou et al. present an improvement over RTA-LC called ZLL [15], by observing that part of the carry-in workload needs to be completed earlier, while RTA-LC assumes they are executed as late as possible. Later Zhou et al. propose a new method, henceforth denoted as EPE-ZLL, by excluding the parallel execution in the calculation of all interferences (not just those from carry-in jobs) [9]. There are also other *sufficient-only* schedulability tests in the literature such as [16], [17], but EPE-ZLL is demonstrated to achieve the best performance among them [9].

There are a few *exact* schedulability tests available, but they are either limited to the case of strictly periodic tasks [18], or time- and memory-consuming thus only practical for analyzing small tasksets (no more than 13 tasks) [19]. Cucu and Goossens derive an exact analysis for periodic tasks by simulating the task executions [18]. Baker et al. propose an exact schedulability test for sporadic tasks with brute-force search for feasible system states [20]. Bonifaci et al. improve this work by traversing a state transition graph and searching for an unschedulable state [4]. In [21], Burmyakov et al. improve the work in [4] by cutting down the state space for analysis. In [19], they further exploit the state-pruning idea to obtain a speedup of 2-3 magnitudes compared to Bonifaci's test. As a different approach than [4], [19], [20], [21], Sun et al. model the schedulability of G-FP as a linear hybrid automaton [22].

For priority assignment in G-FP, there is no known algorithm that is both efficient and compatible with the most accurate schedulability test. Cucu uses exhaustive search to find the optimal priority assignment policy for periodic tasks [23], requiring to check all $n!$ possible priority orderings where n is the number of tasks. Audsley's OPA algorithm only checks $O(n^2)$ priority orders [5], but it comes with a set of compatibility conditions [2]. As far as we know, the most accurate analysis that is compatible with OPA is DA-LC test [2]. Also, various heuristic priority assignment policies are applied to G-FP, including DMPO [10], D-CMPO [11], and DkC [24]. In DMPO [10], the smaller the task deadline, the higher the priority. In D-CMPO [11], the smaller the difference between deadline and worst-case execution time, the higher the priority. DkC is a variant of D-CMPO that includes an extra parameter k, which depends on the number of processors [24]. These heuristics [10], [11], [24] can be combined with any schedulability test. Recently, Lee et al. proposed a machine learning (ML) framework [25], which requires the incremental construction of samples. It may become hard to infer a feasible priority assignment when n is large.

If we look beyond G-FP and review the approaches for priority assignment in other types of real-time systems, an authoritative survey can be found in [3]. In particular, when Audsley's algorithm is not optimal (either because the schedulability analysis violates its compatibility conditions, or the problem involves an objective function or other constraints), the state-of-the-art approaches may be classified into four categories. The *first* is to leverage meta-heuristics such as genetic algorithm (e.g., [26]). We compare with ML, possibly the most advanced method in this category and demonstrate that our approach may be better. The *second* is to develop problem specific heuristics (e.g., [27], [28], [29]). This category often relies on certain properties in the system and cannot easily carry over to G-FP. The *third* category is to directly employ standard optimization frameworks, including BnB (e.g., [30]) and Integer Linear Programming (ILP) (e.g., [31]). However, besides the potential scalability issue, frameworks such as ILP may not be applicable to priority assignment in G-FP. For example, EPE-ZLL lacks an analytical form to calculate the interferences from higher priority tasks [9]. The *fourth* category is to leverage Audsley's algorithm and develop domain-specific frameworks to optimize real-time systems [32], [33], [34], [35]. However, the status quo is that they are all limited to systems where the exact schedulability analysis is still compliant with Audsley's algorithm. Thus for priority assignment in G-FP, the frameworks in [32], [33], [34], [35] are no longer applicable.

III. System Model and Preliminaries

We consider a real-time application consisting of n periodic or sporadic tasks $\Gamma = \{\tau_1, \tau_2, ... \tau_n\}$ scheduled on m identical processors under global fixed priority scheduling algorithm. Each task τ_i is characterized by a tuple $\langle C_i, T_i, D_i \rangle$, where C_i is its Worst-Case Execution Time (WCET), T_i is the period or minimum inter-arrival time, and D_i is the deadline. We assume that the tasks have constrained deadlines ($D_i \le T_i$). Since all events in the system happen at integer clock ticks [9], we assume that these parameters are all *positive integers*. The utilization of task τ_i is defined as $U_i = \frac{C_i}{T_i}$ and the total utilization of the system is $U = \sum_{i=1}^{n} U_i$. Each task τ_i is associated with a unique priority level π_i. For two tasks, τ_i and τ_j, if $\pi_i > \pi_j$, then task τ_i has a higher priority than τ_j. We denote $hp(i) = \{\tau_j | \pi_j > \pi_i\}$ (resp. $lp(i) = \{\tau_j | \pi_j < \pi_i\}$) as the set of tasks with priority higher (resp. lower) than τ_i. Given a taskset Γ, we aim to find a schedulable task priority assignment $\mathbf{P}$ such that all tasks meet their deadline. The task worst-case response time (or in short, *response time*) is denoted as R_i for each task τ_i.

A. Schedulability Analysis

We now provide a summary of the response time analyses. We consider *sufficient-only schedulability analysis* methods since all the exact analysis methods for sporadic tasks suffer from the issue of high computational complexity and can only be practical for *analyzing* a small taskset (up to 13 tasks) [19]. The problem of priority assignment may involve analyzing the schedulability of a large number of design alternatives.

Definition 1. Workload [7], [8]. The workload $W_j(a,b)$ of a task τ_j in an interval $[a,b)$ is the accumulated execution time of τ_j within the interval $[a,b)$.

Definition 2. Interference [7], [8]. The interference $I_j(a,b)$ from a task τ_j on the target task τ_i over a time interval $[a,b)$ is the part of the workload of τ_j that can actually prevent τ_i from executing.

The workload consists of three different parts: *body*, *carry-in*, and *carry-out*. The *body* workload refers to the contribution of all jobs with both release time and deadline in the interval; each job of τ_j contributes to the workload in $[a,b)$ with a complete execution time C_j. The *carry-in* workload is the contribution of at most one job with its release time before a and deadline in $[a,b)$. The *carry-out* workload is the contribution of at most one job with its release time in $[a,b)$ and deadline after b.

DA Test [6]. DA test is based on the observation that the maximum interference could occur when the carry-in job is executed as late as possible and finishes at its deadline. An upper bound on the workload of task τ_j in a time interval of length l is derived as

$$W_j^D(l) = N_j^D(l) \cdot C_j + \min\Big(C_j, l + D_j - C_j - N_j^D(l) \cdot T_j\Big) \quad (1)$$

where $N_j^D(l) = \left\lfloor \frac{l+D_j-C_j}{T_j} \right\rfloor$ denotes the number of jobs whose release time and deadline are both inside the time interval with length l. For task τ_i, the upper bound on the interference introduced by a higher priority task τ_j within the time interval of length l is given by

$$I_j^D(l, C_i) = \min\Big(W_j^D(l), l - C_i + 1\Big) \quad (2)$$

Then, a sufficient schedulability condition for task τ_i is derived by considering a time interval of length D_i

$$D_i \ge R_i = C_i + \left\lfloor \frac{1}{m} \sum_{j \in hp(i)} I_j^D(D_i, C_i) \right\rfloor \quad (3)$$

It is obvious that the $W_j^D(D_i)$ and $I_j^D(D_i, C_i)$ functions only require the knowledge of D_j, T_j, and C_j, which are all independent from τ_j's priority level. Hence, DA-test satisfies the compatibility conditions of Audsley's OPA algorithm [2].

DA-LC Test [2]. DA-LC improves DA test by limiting the interference from carry-in jobs: at most $m-1$ higher priority tasks can contribute to the carry-in workload. This leads to the following schedulability condition for τ_i:

$$D_i \ge R_i = C_i + \left\lfloor \frac{1}{m} \left\{ \sum_{j \in hp(i)} I_j^N(D_i, C_i) + \sum_{j \in [\![i]\!]_{m-1}} \left\{ I_j^D(D_i, C_i) - I_j^N(D_i, C_i) \right\} \right\} \right\rfloor \quad (4)$$

where $[\![i]\!]_{m-1}$ denotes the set of $m-1$ tasks in $hp(i)$ that have the largest values of $I_j^D(D_i, C_i) - I_j^N(D_i, C_i)$, and $I_j^N(D_i, C_i)$ is the maximum interference from τ_j if it has no carry-in job. For an interval of length l, $I_j^N(l, C_i)$ is

$$I_j^N(l, C_i) = \min\left\{ \left\lfloor \frac{l}{T_j} \right\rfloor C_j + \min(C_j, l - \left\lfloor \frac{l}{T_j} \right\rfloor T_j), l - C_k + 1 \right\} \quad (5)$$

Overall, the right-hand side of (4) achieves the maximum value over any subset of $m-1$ tasks in $hp(i)$.

Like DA test, DA-LC still only relies on the parameters T_j, C_j, and D_j of a higher priority task τ_j, which are all independent from its priority. DA-LC is the most accurate analysis that is compatible with OPA [2].

RTA Test [7]. RTA test is similar to DA test but with a more accurate way to execute the carry-in job: the latest time that a job can execute is at its worst-case response time rather than at its deadline. Hence, the workload of τ_j given in (1) can be more accurately rewritten as

$$W_j^R(l) = N_j^R(l) \cdot C_j + \min\Big(C_j, l + R_j - C_j - N_j^R(l) \cdot T_j\Big) \quad (6)$$

where $N_j^R(l) = \left\lfloor \frac{l+R_j-C_j}{T_j} \right\rfloor$ denotes the number of jobs of τ_j whose entire execution is inside the time interval with length l. The response time of τ_i under G-FP is the least fixed point solution of the following equation

$$R_i = C_i + \left\lfloor \frac{1}{m} \sum_{\tau_j \in hp(i)} I_j^R(R_i, C_i) \right\rfloor \quad (7)$$

where the inference $I_j^R(\cdot,\cdot)$ of τ_j now uses the new workload function $W_j^R(\cdot)$

$$I_j^R(l, C_i) = \min\Big(W_j^R(l), l - C_i + 1\Big) \quad (8)$$

Since $I_j^R(R_i, C_i)$ now depends on τ_j's response time R_j, the response time of τ_i not only relies on $hp(i)$ but also on the relative order among the tasks in $hp(i)$. Thus, RTA test is OPA-incompatible [2].

RTA-LC Test [8]. RTA-LC, similar to DA-LC, is based on the idea of limiting the interference from carry-in jobs. RTA-LC uses R_j as the latest completion time of a higher priority task τ_j as opposed to D_j in DA-LC. This requires the following modification to (4):

$$R_i = C_i + \left\lfloor \frac{1}{m}\Omega_i(R_i) \right\rfloor \tag{9}$$

where the function $\Omega_i(\cdot)$ is defined as

$$\Omega_i(l) = \sum_{j\in hp(i)} I_j^N(l, C_i) + \sum_{j\in [\![i]\!]_{m-1}} \left\{ I_j^R(l, C_i) - I_j^N(l, C_i) \right\} \tag{10}$$

and the functions $I_j^N(\cdot,\cdot)$ and $I_j^R(\cdot,\cdot)$ are defined in (5) and (8) respectively.

Like RTA, RTA-LC is OPA-incompatible due to its dependency on R_j, the response time of a higher priority task τ_j [2]. However, we show that our enhancement over OPA, the MITER-based framework, is still compatible with RTA and its improvement RTA-LC (see Section IV).

EPE-ZLL Test [9]. EPE-ZLL tries to reduce the overestimation of interferences from all parts of the workload (carry-in, body, carry-out), not just those from the carry-in. It derives a lower bound on the accumulative time the target task and higher priority tasks are executed in parallel, which can be excluded from the interference.

To calculate R_i, EPE-ZLL uses an iterative process that gradually increases the value of τ_i's execution time a from one (i.e., $a = 1$) to its real value C_i ($a = C_i$). At each iteration, EPE-ZLL calculates the corresponding response time UR_i^{a+1} based on the value of UR_i^a. Specifically, UR_i^{a+1} is the minimal solution of the following equation

$$x \geq \left\lfloor \frac{\Psi_i(a, x)}{m} \right\rfloor + a + 1 \tag{11}$$

where $\Psi_i(a, x)$ is defined as

$$\Psi_i(a,x) = \min\Big\{ \quad m\left(UR_i^a - a\right) + \sum_{j\in hp(i)\cup\{i\}} W_{j,i}^{\max}(a,x), \\ \Omega_i\left(x + UR_k^a\right) \Big\} \tag{12}$$

Note that the calculation of $\Omega_i\left(x + UR_i^a\right)$ follows (10), and $W_{j,i}^{\max}(a,x)$ is calculated by an algorithm that requires the relative priority order of those tasks in $hp(i)$ to determine the worst case release times of interfering jobs [9]. Similar to RTA-LC, EPE-ZLL is also incompatible with OPA. In addition, even if we assume that the response times of all tasks are known, the calculation of $W_{j,i}^{\max}(a,x)$ and hence EPE-ZLL still require the relative priority order in $hp(i)$.

IV. MITER-guided Optimization Algorithm

In this section, we develop a novel optimization framework that advances OPA, termed Maximal Unschedulable response Time Estimation Range guided optimization algorithm (in short, MITER). Like OPA, MITER is optimal *if the associated schedulability analysis satisfies its compatibility conditions*, but those conditions are more relaxed than OPA, thus it can accommodate more accurate analysis that OPA cannot, including RTA and RTA-LC.

Still, MITER is unable to use some of the sophisticated but more accurate analyses such as EPE-ZLL. *In this section, by schedulability we only mean that there exists a priority assignment that makes the system schedulable with respect to the MITER-compatible analysis (and MITER will be able to find it).* It is possible that due to the inaccuracy of the chosen analysis, even if MITER fails to find a schedulable priority assignment, the system may still be deemed schedulable with another more accurate analysis. Similarly, *optimality in this section is also in regard to the used schedulability analysis.*

A. Response Time Dependency

We first introduce the concept of *response time dependency (in short, RT dependency)*. It is the property that the schedulability analysis shall satisfy in order to be used together with MITER. Specifically, we assume the response time R_i of task τ_i can be written in the following form

$$R_i = f_i\left(hp(i), \mathbf{R}\right), \quad \forall \tau_i \tag{13}$$

where the function $f_i(\cdot)$ takes as inputs $hp(i)$, the set of higher priority tasks, and $\mathbf{R}$, the vector of response time estimations for all tasks. Note that if $hp(i)$ is given, $lp(i)$ is also determined since $lp(i) \bigcup hp(i) = \Gamma\backslash\{\tau_i\}$. Thus, for simplicity, we omit $lp(i)$ in the definition of f_i.

The response time analysis written in the form of (13) directly implies the following two assumptions

- A1: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of higher priority tasks $hp(i)$, but not on their relative order.
- A2: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of lower priority tasks $lp(i)$, but not on their relative order.

We assume (13) further satisfies two more conditions

- A3: the response time R_i of τ_i is monotonically non-decreasing with the set of higher priority tasks $hp(i)$. Specifically, given two sets of tasks $hp(i)$ and $hp'(i)$ such that $hp(i) \subseteq hp'(i)$, the response time R_i with $hp'(i)$ as the higher priority tasks is no smaller than that with $hp(i)$.
- A4: the response time R_i of τ_i is monotonically non-decreasing with the increase of the response time R_j of any other task τ_j.

Now we give the formal definition of RT dependency.

Definition 3. The response time analysis of a real-time system Γ is said to be response time dependent (**RT dependent**) if it satisfies the above four assumptions A1-A4.

Comparably, the compatibility conditions of OPA are

- A1': the response time R_i of τ_i relies on $hp(i)$, the set of higher priority tasks, but not on their relative order.
- A2': the response time R_i of τ_i relies on the set of lower priority tasks $lp(i)$, but not on their relative order.
- A3': the same as A3 (it is rewritten differently than [2]).

It is easy to see that A1-A4 are a weaker requirement than A1'-A3', hence MITER may be optimal with a more accurate analysis than Audsley's OPA. In fact, RTA and RTA-LC are both proven to violate A1'-A3' [2], but they can be combined with MITER as they are RT dependent. We formally state that in the following theorem.

Theorem 1. RTA and RTA-LC tests are RT-dependent.

Proof. We only prove property A4 for the two tests. The rest of the proof is straightforward and can be found in the online Appendix.

We first show that the workload function $W_j^R(\cdot)$ as in Equation (6) will be monotonically non-decreasing with the increase of R_j for any other task τ_j $(j \neq i)$. For any schedulable system, we must have $C_j \leq T_j$. We first show $W_j^R(l)$ is monotonic with R_j when $l + R_j - C_j \in [K \cdot T_j, (K+1)T_j)$, where the value of $N_j^R(l)$ is always equal to K. Let $x = l + R_j - C_j - K \cdot T_j$, then $W_j^R(l) = K \cdot C_j + \min(C_j, x)$. We consider two cases.

- Case 1: $l + R_j - C_j \in [K \cdot T_j, K \cdot T_j + C_j]$. Then the value of x will fall inside the interval $[0, C_j]$. The workload is
$$W_j^R(l) = K \cdot C_j + x, \;\; x \in [0, C_j] \tag{14}$$
which is a linear function of R_j with a positive coefficient 1.
- Case 2: $l + R_j - C_j \in [K \cdot T_j + C_j, (K+1) \cdot T_j)$. x now becomes no smaller than C_j. Hence, the workload is a constant in this case:
$$W_j^R(l) = K \cdot C_j + C_j, \;\; x \in [C_j, T_j) \tag{15}$$

From the above two cases, we can conclude that when $l + R_j - C_j \in [K \cdot T_j, (K+1) \cdot T_j)$, the workload $W_j^R(l)$ is monotonically non-decreasing when R_j increases. Now we prove that when $l + R_j - C_j = (K+1) \cdot T_j$ (i.e., $N_j^R(l) = K+1$), the workload is guaranteed to be larger than or equal to the one when $l + R_j - C_j = (K+1) \cdot T_j - 1$. From the previous discussion, we already know that

$$W_j^R(l) = K \cdot C_j + C_j, \text{ if } l + R_j - C_j = (K+1) \cdot T_j - 1 \tag{16}$$

When R_j increases by one, we have

$$W_j^R(l) = (K+1) \cdot C_j + \min(C_j, 0) \tag{17}$$
$$= (K+1) \cdot C_j, \text{ if } l + R_j - C_j = (K+1) \cdot T_j \tag{18}$$

Therefore, the workload $W_j^R(l)$ is monotonically non-decreasing in the interval $[K \cdot T_j, (K+1) \cdot T_j]$ for arbitrary integer K, and consequently it is monotonically non-decreasing with R_j.

This easily lets us conclude that the interference $I_j^R(\cdot,\cdot)$ in (8), and the response time for RTA in (7) are monotonically non-decreasing with R_j. For RTA-LC, the response time in (9) additionally relies on $I_j^N(\cdot,\cdot)$, but this function, as defined in (5), is independent from R_j. Hence, condition A4 is satisfied for both RTA and RTA-LC. □

On the contrary, EPE-ZLL test [9] does not satisfy the conditions of RT dependency, since even if we assume the response times of all tasks are known beforehand, the relative priority orders are still required to determine the worst-case combination of the release times for the interfering jobs.

Theorem 2. EPE-ZLL test is not RT-dependent.

Proof. We leave the proof to the online Appendix. □

B. Response Time Estimation Range

The response time analysis in (13) violates the conditions of Audsley's OPA. However, if the response times of every task τ_j $(j \neq i)$ is appropriately estimated, then computing R_i in (13) only requires the knowledge of the set of higher/lower priority tasks, and not the relative order in them. This, combined with A3, allows us to leverage OPA *if the response times of all tasks can be estimated appropriately.* We first introduce the concepts and properties related to response time estimations.

Definition 4. A **Response time estimation (RTE)** is defined as a collection of tuples $\langle \tau_i, r_i \rangle$ for all tasks, i.e., $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$. In each tuple $\langle \tau_i, r_i \rangle$, r_i represents the estimated response time of task τ_i where $r_i \in [C_i, D_i]$.

For a given $\mathcal{E}$, the *estimation-inferred response time* of τ_i is calculated with the analysis in (13) assuming the response times of other tasks follow the estimated value in $\mathcal{E}$.

Definition 5. Given a priority assignment **P** and a response time estimation $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$, the **estimation-inferred response time** of task τ_i is the least fixed point solution of the following equation, denoted as R_i^E,

$$R_i^E = f_i\left(hp(i), \mathbf{E}_i\right), \quad \forall \tau_i \tag{19}$$

where $\mathbf{E}_i$ is a vector: the i-th entry of E_i is the variable R_i^E, and the j-th entry for any $j \neq i$ takes the corresponding given value r_j in $\mathcal{E}$,

$$\mathbf{E}_i = \left[r_1, \ldots, R_i^E, \ldots, r_n\right] \tag{20}$$

The vector of estimation-inferred response times R^E is denoted as

$$R^E = \left[R_1^E, R_2^E, \ldots, R_n^E\right] \tag{21}$$

Remark 1. With a given RTE $\mathcal{E}$, the calculation of estimation-inferred response time R_i^E for task τ_i only depends on the set of higher/lower priority tasks, but not on their relative order.

A desired property of an RTE that secures a schedulable priority assignment is given below (Definition 6). Moreover, there must exist an RTE with this property if and only if the system is schedulable (Theorem 3).

Definition 6. An RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$ is defined as **feasible** if and only if there exists a priority assignment **P** such that the estimation-inferred response times are component-wise no larger than $\mathcal{E}$. That is, $\mathcal{E}$ is feasible if and only if

$$\exists \mathbf{P} \;\; s.t. \;\; \forall i = 1..n, \;\; R_i^E = f_i\left(hp(i), \mathbf{E}_i\right) \leq r_i \tag{22}$$

Theorem 3. A system Γ has a schedulable priority assignment if and only if there exists a feasible RTE $\mathcal{E}$ [36].

Theorem 3 claims that instead of directly searching for a feasible priority assignment, an alternative approach is to search for a response time estimation $\mathscr{E}$ that satisfies Equation (22). Checking RTE $\mathscr{E}$ against Equation (22) can be easily done using Audsley's algorithm since the estimation-inferred response depends only on the set of higher priority tasks but not on their relative order.

The total number of response time estimations is $O(\prod_i D_i)$, which is obviously impractical to do an exhaustive search. We propose a search space reduction technique that greatly improves the algorithm efficiency. The idea is that many infeasible RTEs share common reasons that violate definition 6 so we can rule out similar infeasible RTEs from the search space together to improve efficiency.

We first discuss the two conflicting requirements of an RTE before giving a formal definition. For a given RTE $\mathscr{E} = \{\langle\tau_1, r_1\rangle, \ldots \langle\tau_n, r_n\rangle\}$, consider any element $\langle\tau_i, r_i\rangle$, where the estimated response time r_i is used in two distinct places in condition (22): (a) r_i acts as an upper bound for a feasible estimation-inferred response time R_i^E of τ_i; (b) when computing the estimation-inferred response time R_j^E $(j \neq i)$, r_i is a constant entry in vector $\mathbf{E}_j$. Obviously for (a), a larger value of r_i is better but for (b), a smaller value of r_i is desirable since the estimation-inferred response time R_j^E is non-decreasing with r_i (property A4). These two conflicting requirements suggest that an infeasible RTE can be generalized to a range of response time estimations such that any of them falling inside this range will be infeasible too.

Instead of using r_i directly, we now split the estimation r_i into an optimistic estimation r_i^l and a pessimistic one r_i^u, where $r_i^l \leq r_i^u$. As discussed above, now we use r_i^u for (a) and r_i^l for (b), which will result in a weaker condition (26) than (22). Suppose that this weaker condition (26) does not even allow a schedulable priority assignment, then it can be implied that any r_i falling inside the range $[r_i^l, r_i^u]$ would be infeasible for the original condition (22).

We now formalize the idea with the following definitions.

Definition 7. A **response time estimation range** $\mathscr{G}$ is a collection of tuple elements $\langle\tau_i, [r_i^l, r_i^u]\rangle$ for each task τ_i, i.e., $\mathscr{G} = \{\langle\tau_1, [r_1^l, r_1^u]\rangle, ..\langle\tau_n, [r_n^l, r_n^u]\rangle\}$, where $C_i \leq r_i^l \leq r_i^u \leq D_i$. It represents a range of possible estimation values for the actual response time R_i of each task τ_i.

Note that in the definition, we restrict $[r_i^l, r_i^u]$ of each task τ_i to be within $[C_i, D_i]$, as the response time R_i of τ_i for any schedulable system shall be in that range.

Definition 8. A response time estimation $\mathscr{E}$ is said to be **contained** in a response time estimation range $\mathscr{G}$, denoted as $\mathscr{E} \in \mathscr{G}$, if and only if for each $\langle\tau_i, r_i\rangle$ in $\mathscr{E}$, the corresponding range $\langle\tau_i, [r_i^l, r_i^u]\rangle$ in $\mathscr{G}$ satisfies $r_i \in [r_i^l, r_i^u]$.

Definition 9. Given a response time estimation range $\mathscr{G} = \{\langle\tau_1, [r_1^l, r_1^u], ..[r_n^l, r_n^u]\rangle\}$ and a priority assignment $\mathbf{P}$, the **estimation range-inferred response time** of τ_i, denoted as R_i^G, is the least fixed point of the following equation

$$R_i^G = f_i\left(hp(i), \mathbf{G}_i\right) \tag{23}$$

where $\mathbf{G}_i$ is a vector constructed by taking the i-th entry as variable R_i^G and any other j-th entry as the value r_j^l from $\mathscr{G}$

$$\mathbf{G}_i = \left[r_1^l, \ldots, R_i^G, \ldots, r_n^l\right] \tag{24}$$

The **vector of estimation range-inferred response times** is denoted as $\mathbf{R}^G$.

Intuitively, due to property A4, the estimation range-inferred response time is essentially the smallest estimation inferred response time that can possibly be obtained for $\mathscr{E} \in \mathscr{G}$, as shown in the following equation

$$\begin{aligned} & \forall \mathscr{E} \in \mathscr{G},\ \forall i,\ \forall j \neq i,\ r_j \geq r_j^l \\ \Rightarrow\ & \forall \mathscr{E} \in \mathscr{G},\ \forall i,\ R_i^E \geq R_i^G \qquad \text{(by property A4)} \end{aligned} \tag{25}$$

Also, given an estimation range, the analysis of estimation range-inferred response times depends only on the set of higher priority tasks but not on their relative order, hence it is compliant with Audsley's algorithm.

We now define the schedulability of a response time estimation range as follows.

Definition 10. A response time estimation range $\mathscr{G} = \{\langle\tau_1, [r_1^l, r_1^u]\rangle, \ldots \langle\tau_n, [r_n^l, r_n^u]\rangle\}$ is said to be **feasible** if

$$\exists \mathbf{P}\ s.t.\ \ \forall i = 1..n,\ R_i^G = f_i(hp(i), \mathbf{G}_i) \leq r_i^u \tag{26}$$

Condition (26) is weaker than (22): it allows a smaller r_i^l than r_i^u, hence easier to be satisfied than (22). Like (22), (26) can be checked efficiently using Audsley's algorithm.

The usefulness of the concept is shown in the following theorem, which demonstrates that an infeasible response time estimation range implies all its contained response time estimations are infeasible. Unlike the case of *infeasible* response time estimation range, its *feasible* version is less useful in the sense that the contained RTE may or may not be feasible.

Theorem 4. Given an infeasible response time estimation range $\mathscr{G} = \{\langle\tau_1, [r_1^l, r_1^u]\rangle, \ldots \langle\tau_n, [r_n^l, r_n^u]\rangle\}$, any response time estimation $\mathscr{E} \in \mathscr{G}$ is infeasible [36].

Definition 11. A response time estimation range $\mathscr{G}_1 = \{\langle\tau_1, [r_1^{l1}, r_1^{u1}]\rangle, \ldots \langle\tau_n, [r_n^{l1}, r_n^{u1}]\rangle\}$ is a **subset** of another range $\mathscr{G}_2 = \{\langle\tau_1, [r_1^{l2}, r_1^{u2}]\rangle, \ldots \langle\tau_n, [r_n^{l2}, r_n^{u2}]\rangle\}$ if

$$\forall i = 1..n,\ r_i^{l1} \geq r_i^{l2}\ and\ r_i^{u1} \leq r_i^{u2} \tag{27}$$

$\mathscr{G}_1$ is said to be a **strict subset** of $\mathscr{G}_2$ if and only if $\mathscr{G}_1$ is a subset of $\mathscr{G}_2$ and $\mathscr{G}_1 \neq \mathscr{G}_2$.

We now define a class of infeasible response time estimation ranges that are not a strict subset of any other infeasible ones. This can maximize its contained infeasible RTEs.

Definition 12. A response time estimation range $\mathscr{U}$ is a **Maximal Infeasible response Time Estimation Range (MITER)**[1] if and only if it satisfies the following conditions

[1]With a slight abuse of terminology, we use MITER to also refer to the MITER-guided optimization framework. The meaning of MITER should be clear from the context.

Algorithm 1: Algorithm for Computing MITER

1 **function** MITER(*(infeasible RTE* $\mathscr{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$*)*)
2 $\quad \mathscr{G} = \{\langle \tau_1, [r_1, r_1] \rangle, \ldots \langle \tau_n, [r_n, r_n] \rangle\}$
3 $\quad$ **for** *each* $\langle \tau_i, [r_i^l, r_i^u] \rangle \in \mathscr{G}$ **do**
4 $\quad\quad$ Use binary search to find out the smallest value that r_i^l can be decreased to while keeping $\mathscr{G}$ infeasible.
5 $\quad\quad$ Use binary search to find out the largest value that r_i^u can be increased to while keeping $\mathscr{G}$ infeasible.
6 $\quad$ **return** $\mathscr{G}$

- $\mathscr{U}$ is infeasible by Definition 10;
- For all $\mathscr{G}$ such that $\mathscr{U} \subset \mathscr{G}$, $\mathscr{G}$ is feasible.

Remark 2. For the concept of response time estimation range, we shall treat the subset relationship as a partial order, i.e., $\mathscr{G}_1$ is no larger than $\mathscr{G}_2$ if $\mathscr{G}_1 \subseteq \mathscr{G}_2$. Let $\mathbb{S}$ be the set of all *infeasible* response time estimation ranges. A MITER $\mathscr{U}$ by Definition 12 is essentially a "maximal" element of $\mathbb{S}$. Since the order among response time estimation ranges is only partial, there may be multiple maximal elements of $\mathbb{S}$, i.e., multiple MITERs.

Intuitively, consider two infeasible response time estimation ranges $\mathscr{G}_1$ and $\mathscr{G}_2$ such that $\mathscr{G}_1 \subseteq \mathscr{G}_2$. $\mathscr{G}_1$ is redundant in the presence of $\mathscr{G}_2$, since the latter contains all infeasible RTEs contained in $\mathscr{G}_1$. In this sense, a MITER $\mathscr{U}$ is more useful than any of its subset $\mathscr{G}$ (i.e., $\mathscr{G} \subseteq \mathscr{U}$), since it is more efficient than $\mathscr{G}$ in capturing infeasible RTEs. We leverage this property to rule out the most infeasible RTEs with the fewest number of infeasible response time estimation ranges.

An infeasible RTE $\mathscr{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$ can be generalized into a MITER by Algorithm 1. We assume that the initial input $\mathscr{E}$ satisfies $r_i \in [C_i, D_i]$ for each task τ_i. We will later show in Section IV-C how it can be guaranteed. The algorithm first converts the RTE to a response time estimation range $\mathscr{G} = \{\langle \tau_1, [r_1, r_1] \rangle, \ldots \langle \tau_n, [r_n, r_n] \rangle\}$ containing only $\mathscr{E}$ itself (Line 2). Then it leverages the property that condition (26) is monotonic w.r.t. each r_i^l and r_i^u: increasing r_i^u or decreasing r_i^l can only make (26) easier to be satisfied. It uses binary search to find out the minimum value that r_i^l can be decreased to (or the maximum value r_i^u can be increased to) while maintaining the unschedulability of $\mathscr{G}$ (Lines 3–6).

Specifically, Line 4 preserves the values of r_i^u and all other response time estimation ranges $\left\langle \tau_j, [r_j^l, r_j^u] \right\rangle, i \neq j$, and uses binary search to decrease r_i^l as much as $\mathscr{G}$ is infeasible. By Definition 5, r_i^l must be no smaller than C_i. Also, r_i^l has an initial value r_i which is known to be infeasible. Thus, the initial lower and upper bounds for the binary search of r_i^l are C_i and r_i respectively. The binary search stops when the lower and upper bounds converge (i.e., their difference is no more than one, which is sufficient since all response time estimations are integers). Line 5 is similar except that (a) it increases r_i^u, while keeping r_i^l at the value determined by Line 4; (b) the initial lower and upper bounds for r_i^u are r_i and D_i respectively.

At Lines 4–5, Audsley's algorithm is used to check if the resulting $\mathscr{G}$ is feasible, i.e., to see if it permits a priority assignment that satisfies (26). Note that Audsley's algorithm only needs to explore $O(n^2)$ priority orders, r_i^l is bounded below by $C_i \leq D_i$, and r_i^u is bounded above by D_i, hence Algorithm 1 checks a total of $O(n^2 \log D)$ priority orders to calculate a MITER, where $D = \prod_i D_i$, and n is the number of tasks.

C. MITER-Guided Framework

By leveraging the concepts of RTE and MITER, we present an optimization algorithm for systems with an RT-dependent response time analysis. Finding a schedulable priority assignment for such systems, including G-FP, is particularly difficult since there is no known tractable procedure like Audsley's algorithm. As in Theorem 3, finding a schedulable priority assignment is equivalent to finding a feasible RTE. The latter has the promise to be more scalable for the following unique capability from Algorithm 1: it can efficiently generalize a given infeasible RTE to a set of MITERs, each of which contains a maximal range of infeasible RTEs.

We design the optimization algorithm that leverages the power of Algorithm 1. Instead of directly solving the original problem $\mathbb{O}$, we start with a relaxed problem $\mathbb{X}$ that leaves out all the system schedulability constraints. If the obtained RTE by solving $\mathbb{X}$ is infeasible, we use Algorithm 1 to generalize it to a set of MITERS. The corresponding constraints are then added to problem $\mathbb{X}$ to rule out similar infeasible RTEs, and the updated problem will be solved again. The iterative procedure will terminate (i) if the obtained RTE is feasible, which is guaranteed to be an optimal solution of $\mathbb{O}$, or (ii) $\mathbb{X}$ is deemed infeasible, which implies $\mathbb{O}$ is also infeasible (see Theorem 5).

The procedure is illustrated in Figure 1. Step 1 checks if the most relaxed response time estimation range is schedulable. If yes, it enters the loop between Step 2 (solving the relaxed problem $\mathbb{X}$) and Step 3 (computing MITERs). The details for each step is explained as follows.

Step 1. Let R_i^L and R_i^U denote the smallest and greatest values of the response time of each task τ_i in any schedulable priority assignment. In this paper, we assume $R_i^L = C_i$ and $R_i^U = D_i$. Step 1 evaluates whether the response time estimation range $\zeta^M = \{\langle \tau_1, [C_1, D_1] \rangle, \ldots \langle \tau_n, [C_n, D_n] \rangle\}$ is schedulable. If not, then the systems must be unschedulable by any priority assignment, and the algorithm reports unschedulability and terminates.

Step 2. The second step searches for a response time estimation that has not been deemed unschedulable by the currently computed MITERs. This is done by solving a relaxed problem $\mathbb{X}$ consisting of no schedulability conditions but the implied constraints by the computed MITERs. Specifically for each MITER $\mathscr{U} =$

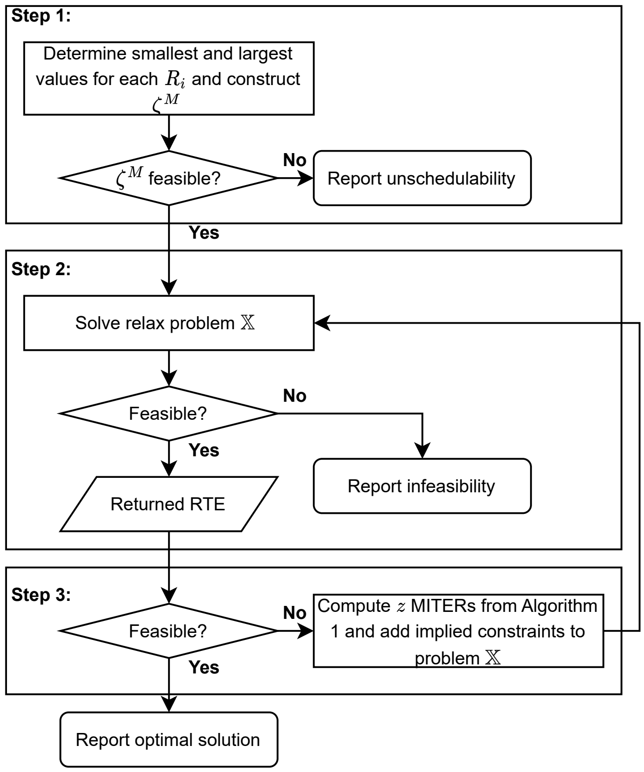


Fig. 1. MITER-guided priority assignment algorithm

$\{\langle \tau_1, [r_1^l, r_1^u] \rangle, ... \langle \tau_n, [r_n^l, r_n^u] \rangle\}$, by Theorem 4 any feasible RTE $\mathscr{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ cannot be contained in $\mathscr{U}$. This implies the following constraint

$$\mathscr{E} \notin \mathscr{U} \;\Leftrightarrow\; \neg \begin{cases} r_1^l \le r_1 \le r_1^u \\ ... \\ r_n^l \le r_n \le r_n^u \end{cases} \;\Leftrightarrow\; \left\| \begin{array}{l} r_1 < r_1^l \| r_1 > r_1^u \\ ... \\ r_n < r_n^l \| r_n > r_n^u \end{array} \right. \tag{28}$$

where $\neg$, $\{$, and $\|$ represent the logic-NOT, logic-AND, and logic-OR operations, respectively. Thus, $\mathbb{X}$ just consists of the objective and all the other constraints in $\mathbb{O}$, while replacing the system schedulability constraints with those of (28) implied by all currently computed MITERs. $\mathbb{X}$ also includes the response time estimations of all tasks $[r_1, ... r_n]$ as the additional design variables, as well as their initial bounding constraints $C_i \le r_i \le D_i, \forall i$. In the problem of priority assignment for G-FP, $\mathbb{O}$ has no objective and other constraints, but only the system schedulability constraints. Hence, problem $\mathbb{X}$ can be expressed as

$$\begin{aligned} \mathbb{X}: \min \quad & 1 \\ \text{s.t.} \quad & \text{Implied constraints of } \mathscr{U} \text{ as in (28)}, \;\forall\; \mathscr{U} \in \mathbb{U} \\ & C_i \le r_i \le D_i, \forall i \end{aligned} \tag{29}$$

where $\mathbb{U}$ is the set of currently known MITERs.

$\mathbb{X}$ can be solved using ILP solvers such as CPLEX. Note that the logical disjunction constraint in (28) can be formulated in ILP using the "big-M" method. For example,

$$\left\| \begin{array}{l} r_1 < r_1^l \\ r_1 > r_1^u \end{array} \right. \Leftrightarrow \begin{cases} r_1 < r_1^l + M \cdot b_1 \\ r_1 > r_1^u - M \cdot (1 - b_1) \end{cases} \tag{30}$$

Here b_1 is an auxiliary *binary* variable, defined as

$$b_1 = \begin{cases} 0, & \text{if } r_1 < r_1^l \\ 1, & \text{if } r_1 > r_1^u \end{cases} \tag{31}$$

M is a sufficiently large constant such as D_1.

If $\mathbb{X}$ is infeasible (i.e., no solution satisfies all constraints in $\mathbb{X}$), then the system cannot be schedulable by any RTE (see Theorem 5 below). Otherwise, solving $\mathbb{X}$ will return an RTE (composed of the solution values assigned to the decision variables $[r_1, ... r_n]$) that respects all the implied constraints by the set of known MITERs $\mathbb{U}$.

Step 3. This step evaluates the schedulability of the RTE $\mathscr{E}$ returned in Step 2, i.e., whether it satisfies Equation (22). If yes, then $\mathscr{E}$ is optimal, and the associated priority assignment **P** is an optimal priority assignment (proven in Theorem 5). Here the priority assignment **P** can be obtained as a byproduct of applying Audsley's algorithm in checking $\mathscr{E}$ against the condition (22).

If $\mathscr{E}$ is infeasible, it is generalized into a number of MITERs using Algorithm 1. The implied constraints from these newly discovered MITERs are then added to the problem $\mathbb{X}$, and we enter the next iteration. Here we set an upper bound z on the number of MITERs generated from a single infeasible RTE. In our experiments, we find that $z = 5$ is a good setting in most cases. A set of examples that explains the definitions and calculation processes of the proposed concepts is included in our previous work (i.e., Examples 1–8) [36].

The following theorem formally proves the correctness of the proposed algorithm in Figure 1.

Theorem 5. The algorithm in Figure 1 guarantees to **terminate**. Upon termination, it reports **infeasibility/unschedulability** if the original problem $\mathbb{O}$ is infeasible, otherwise it returns a **schedulable** priority assignment that is **optimal** with respect to the given objective [36].

Although the algorithm in Figure 1 is guaranteed to terminate, in the worst case it may still require to compute all MITERs. Consequently it needs $O(\prod_i D_i^2)$ number of iterations between Steps 2 and 3, as each iteration computes at most z number of distinct MITERs, where z is a predefined constant. Also, in each iteration it needs to solve an ILP problem $\mathbb{X}$. In the end, the algorithm still has exponential worst-case complexity.

In the problem of priority assignment for G-FP, the main advantage of MITER-guided framework compared to OPA is its optimality with a more accurate analysis RTA-LC than DA-LC, the most accurate analysis that can be used with OPA. In our experiments, we show that MITER with RTA-LC is consistently outperforming OPA with DA-LC (up to 30% in acceptance ratio), despite that we limit the runtime of the algorithms (see Section VI).

V. Hybrid Algorithm with MITER

Each of the two categories of priority assignment algorithms for G-FP now has its own advantages: MITER with RTA-LC, our advancement to OPA-based algorithms, can schedule more tasksets when the system utilization is low, and DkC with EPE-ZLL, the best in the second category (heuristic with any analysis that may be more accurate than RTA-LC), works better for high system utilizations. We prudently combine their strengths and present a hybrid algorithm that significantly outperforms both.

We first briefly introduce the priority assignment process in OPA [5]. All tasks are not assigned priorities at the beginning. For each priority level, from lowest to the highest, OPA selects a task τ_i and assigns the current priority level to it. Assuming that all unassigned tasks have a higher priority than τ_i, the schedulability of τ_i is tested with an OPA-compatible test. If it is schedulable, the priority assignment is successful and OPA proceeds to the next higher priority level. If τ_i is unschedulable, the algorithm iteratively selects another unassigned task and repeats the schedulability test. If the whole set of unassigned tasks is traversed and no schedulable task is found, then OPA returns unschedulable. If all tasks are assigned priority levels at the end, the OPA returns schedulable.

Our observation is that OPA is a powerful and efficient algorithm, but it has one major limitation from its compatibility condition A1' (A2' and A3' are actually satisfied by most, if not all, schedulability tests including EPE-ZLL). The proposed idea is that in OPA, when deciding if τ_i is schedulable at a particular priority level, we may find a way to temporarily approximate the priority order of higher priority tasks $hp(i)$. This allows us to estimate the response time R_i of τ_i, even if we use the OPA-incompatible analysis EPE-ZLL.

Our new algorithm, called hybrid priority assignment with MITER (or in short HP-MITER), is detailed in Algorithm 2. Like OPA, it iteratively assigns a task to a priority level starting from the lowest to the highest (Lines 2–24). At each level p, it calculates the response time of each unassigned task τ_i using EPE-ZLL (Line 12). However, EPE-ZLL requires the total order among the set of higher priority tasks $hp(i)$. Hence, we leverage the strengths of the two categories of priority assignments for G-FP and use a hybrid of MITER with RTA-LC and DkC with EPE-ZLL to estimate the priority order among tasks in $hp(i)$. When the total utilization of tasks in $hp(i)$ is lower than a predefined threshold Θ (see next paragraph), MITER with RTA-LC is utilized to get the total order in $hp(i)$ (Lines 6–9). Otherwise, DkC is applied (Lines 10–11). Note this order $\mathbf{P}_{hp(i)}$ within $hp(i)$ is temporary and only used for the purpose of determining the response time R_i of τ_i (Line 12) or system schedulability (Line 14) while trying to assign τ_i at priority level p.

Algorithm 2: Hybrid Priority Assignment with MITER

Input: Set of tasks Γ, Utilization threshold Θ
Use: Set of unassigned tasks Δ, Set of schedulable tasks Γ_s at the given priority level

1 $\Delta = \Gamma$
2 **for** *each priority level p from lowest to highest* **do**
3 $\quad \Gamma_s = \emptyset$
4 $\quad$ **for** *each unassigned task τ_i in Δ* **do**
5 $\quad\quad hp(i) = \Delta \backslash \{\tau_i\}$
6 $\quad\quad$ **if** *Utilization of $hp(i)$ is lower than Θ* **then**
7 $\quad\quad\quad$ Use MITER to find priority order $\mathbf{P}_{hp(i)}$ for tasks in $hp(i)$
8 $\quad\quad\quad$ **if** *Unsuccessful* **then**
9 $\quad\quad\quad\quad$ continue to next τ_i
10 $\quad\quad$ **else**
11 $\quad\quad\quad$ Use DkC to find priority order $\mathbf{P}_{hp(i)}$ for tasks in $hp(i)$
12 $\quad\quad$ Calculate R_i of τ_i at priority p using EPE-ZLL
13 $\quad\quad$ **if** $R_i \le D_i$ **then**
14 $\quad\quad\quad$ Check schedulability of Γ using EPE-ZLL
15 $\quad\quad\quad$ **if** Γ *is schedulable* **then**
16 $\quad\quad\quad\quad$ Return **schedulable**
17 $\quad\quad\quad$ **else**
18 $\quad\quad\quad\quad$ Add τ_i to Γ_s
19 $\quad$ **if** $\Gamma_s \neq \emptyset$ **then**
20 $\quad\quad$ Assign p to the task τ_i in Γ_s with the largest DkC value
21 $\quad\quad \Delta = \Delta \backslash \{\tau_i\}$
22 $\quad$ **else**
23 $\quad\quad$ Assign p to the task τ_i in Δ with the smallest lateness $R_i - D_i$ value
24 $\quad\quad \Delta = \Delta \backslash \{\tau_i\}$
25 Return **unschedulable**

Threshold. The threshold Θ is acting as a dividing line between MITER with RTA-LC and DkC with EPE-ZLL. As shown in [9, Fig. 13], EPE-ZLL is significantly more accurate than RTA-LC when the system utilization is higher than 40%, otherwise, the difference between the two methods is negligible. Since MITER is optimal and better than DkC for MITER-compatible analysis such as RTA-LC, we set the threshold Θ at 40%, with the hope that MITER can compensate the small disadvantage of RTA-LC. Also, on varying the threshold from 0% to 100%, it is found that 40% gives the overall best performance in both implicit and constrained deadline cases. Note that this threshold is chosen for the combination of MITER with RTA-LC and DkC with EPE-ZLL, and it may not be the same for other combinations of schedulability tests and priority assignment policies.

Algorithm 2 has a couple of noticeable designs that are different from OPA, considering that the priority order $\mathbf{P}_{hp(i)}$ may not be the same as in the final solution and the estimated response time R_i may not be accurate. *First*, in Lines 13–14, in the case that τ_i is schedulable at level p, we opportunistically check whether the current priority assignment (the temporary order $\mathbf{P}_{hp(i)}$ for the set $hp(i)$, τ_i at level p, and all other tasks follow the previously fixed priority levels) happens to be schedulable according to EPE-ZLL. If so, the algorithm terminates and returns the current priority assignment. *Second*, if we find a schedulable task at priority level p, instead of terminating immediately and returning schedulability, we simply add this task to Γ_s, the set of schedulable tasks at level p (Line 18). Again, this is because $\mathbf{P}_{hp(i)}$ is not necessarily the order in the final solution. *Third*, if at least one candidate task is schedulable at the current priority level (i.e., Γ_s is not empty), then we follow the DkC policy and select the one in Γ_s with the largest DkC value (Line 19–21). If none of them is schedulable, instead of reporting unschedulability as in OPA, we select the one with the smallest lateness $R_i - D_i$ among them and continue to the next priority level (Lines 22–24), in the hope that the small lateness may be erased by a different order in $hp(i)$. *Fourth*, in the rare case that MITER is unable to find any schedulable priority order for the set $hp(i)$, we continue on to the next task in the set of unassigned tasks. (Lines 8–9). *Finally*, the loop terminates at Line 25 returning unschedulable in two cases: (1) if for a particular priority level, MITER is always used to obtain the initial ordering for $hp(i)$ and it is not able to find a feasible ordering for any them, or (2) none of the priority levels lead to Line 14 returning true.

Another point we would like to clarify is the use of schedulability analysis in Algorithm 2. Throughout the algorithm we use EPE-ZLL, the most accurate analysis that is still practical. The only exception is at Line 7, where the MITER-guided framework is leveraged to find the temporary priority order for $hp(i)$. We use RTA-LC at Line 7, the most accurate analysis satisfying the compatibility conditions of MITER.

VI. Experimental Evaluation

In this section, we present the experimental results of the proposed priority assignment algorithms as well as other representative combinations of schedulability tests and priority assignment algorithms. We perform two sets of experiments, the first is to include most of the state-of-the-art except the machine learning framework [25]. Due to the lack of source code and training data from [25], we were unable to duplicate its results. Hence, we use a second set of experiments following the settings in [25] and make an indirect comparison with it.

A. First Experiment

As summarized in Section I, the state-of-the-art can be classified into two categories: an optimal priority assignment algorithm with an analysis that is compatible, and a heuristic algorithm with a more accurate analysis. Hence, we consider the following methods and compare their performances in terms of acceptance ratio, i.e., the percentage of tasksets that are deemed schedulable by each method and the average runtime.

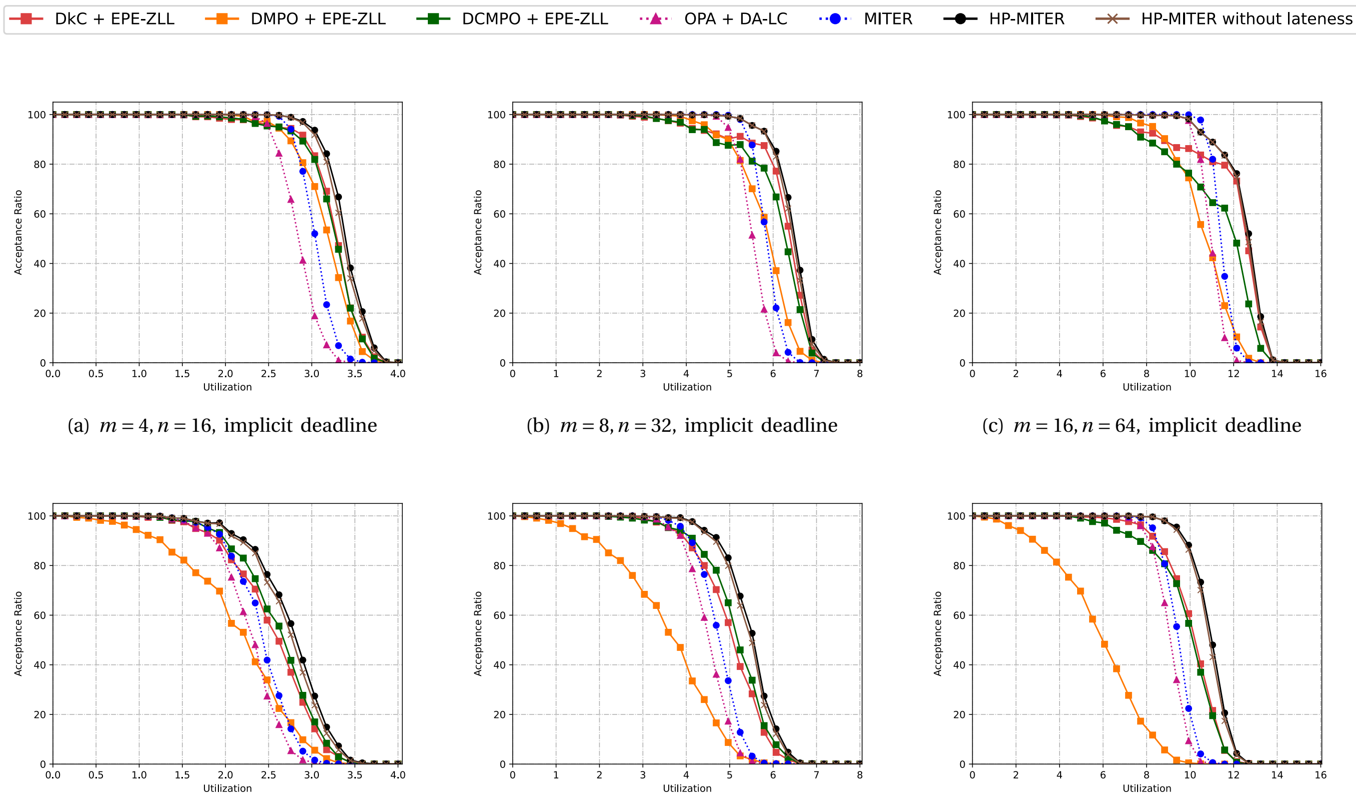
- OPA + DA-LC: Deadline Analysis with Limited Carry-in (DA-LC test from [2]) with Audsley's Optimal Priority Assignment. This is the previous state-of-the-art in the first category.
- MITER: This is our advancement in the first category. It is optimal with RTA-LC, a more accurate analysis than DA-LC.
- EPE-ZLL combined with three different heuristics DkC, DMPO, and D-CMPO. They represent the state-of-the-art in the second category.
- HP-MITER: The hybrid priority assignment algorithm presented in Section V.
- HP-MITER without lateness: This is the version of Algorithm 2 without Lines 22–24. This is done to evaluate the effectiveness of using lateness when all tasks are unschedulable at a priority level.

Since MITER has an exponential worst-case time complexity, to avoid excessive runtime while ensuring a fair comparison, we set the time limit of MITER to the same value of 600 seconds both when used in HP-MITER and as a stand-alone approach for comparison. When a timeout occurs, the taskset is deemed unschedulable by MITER.

We consider the following number of processors and tasks: (1) $m = 4, n = 16$, i.e., 4 processors and 16 tasks; (2) $m = 8, n = 32$; (3) $m = 16, n = 64$. For each case, we choose 30 system utilizations in the range of $[0, m]$. For each utilization, 1000 random tasksets are generated, with the following way to set the task parameters:

- Utilization: Task utilizations are generated using the UUnifast-Discard algorithm [2].
- Period: Task periods are generated according to a log-uniform distribution in the range [10, 1000].
- Execution time: Task execution times are calculated based on the generated utilizations and periods: $C_i = U_i \cdot T_i$.
- Deadline: We test the algorithms for two task models: implicit deadline and constrained deadline. In the latter case, task deadlines are set according to a uniform distribution in the range $[C_i, T_i]$.

The experimental results are plotted in Figure 2, from which we make a few interesting observations. *First*, MITER with RTA-LC is always able to schedule (as much as 30%) more tasksets than OPA + DA-LC. Since both are optimal with respect to their schedulability analyses satisfying the respective compatibility conditions, MITER's superiority comes from its ability to adopt a more accurate analysis than OPA. *Second*, DkC is still a better heuristic than the other two (DMPO and D-CMPO) with the new analysis EPE-ZLL, consistent with the conclusion in [2] that uses RTA-LC as the analysis. However, different than [2], the approaches in the second category, in particular, DkC + EPE-ZLL is now significantly better than the previous state-of-the-art OPA + DA-LC in the first category. Even compared to the new frontier MITER with RTA-LC in the first category, DkC + EPE-ZLL is still able to schedule more tasks at high system utilization. This is mainly because the new advancement in schedulability analysis, EPE-ZLL, is substantially more accurate than RTA-LC or DA-LC, easily compensating the disadvantage of a heuristic priority assignment policy like DkC compared to optimal priority assignment algorithms. *Third*, HP-MITER, taking advantages of both categories, is outperforming the other methods across all settings. For example, it improves OPA + DA-LC by up to 70% for implicit deadline tasks, and by up to 80% for constrained deadline



(a) $m = 4, n = 16$, implicit deadline

(b) $m = 8, n = 32$, implicit deadline

(c) $m = 16, n = 64$, implicit deadline

(d) $m = 4, n = 16$, constrained deadline

(e) $m = 8, n = 32$, constrained deadline

(f) $m = 16, n = 64$, constrained deadline

Fig. 2. Acceptance ratio of each method with varying utilization

Table I. Average Runtime for each Priority Assignment (+ Schedulability Test) in seconds

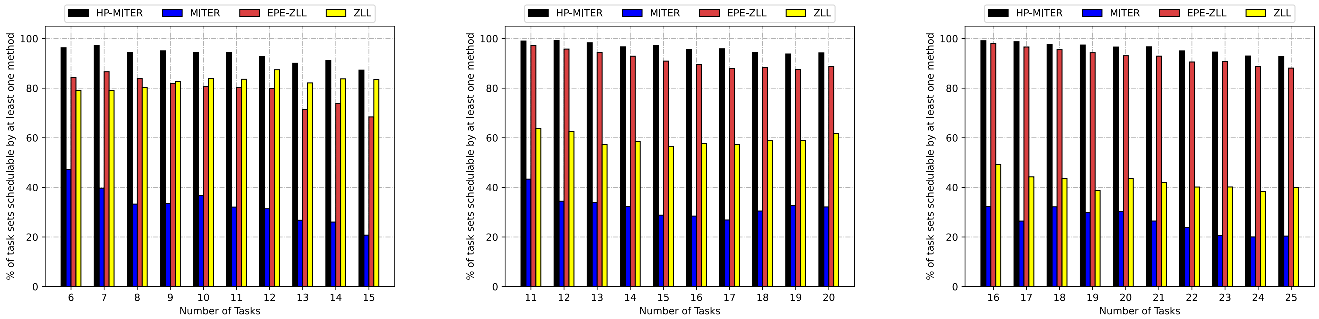
Type of Deadline	m	DkC + EPE-ZLL	DMPO + EPE-ZLL	DCMPO + EPE-ZLL	OPA + DA-LC	MITER	HP-MITER	HP-MITER without lateness
Implicit	4p	0.7939	1.1108	0.8734	0.0029	42.4201	81.9924	27.1600
	8p	4.7934	6.8357	5.6165	0.0110	96.2297	738.4391	182.1743
	16p	23.4517	29.6310	26.5116	0.0395	118.4041	9824.9771	2299.8825
Constrained	4p	0.2587	0.2995	0.3221	0.0016	21.8565	129.9478	28.5039
	8p	1.8508	1.7981	2.7669	0.0080	67.5505	961.5012	193.7253
	16p	12.4310	8.1328	17.8814	0.0363	93.0550	10274.4882	2284.1387

tasks. Compared to DkC + EPE-ZLL, HP-MITER can be better by a margin of 18% for tasks with implicit deadlines. For tasks with constrained deadlines, the improvement of HP-MITER over DkC + EPE-ZLL is even higher, with a maximum difference of 25%. On comparing HP-MITER with the version where lateness is not used, there is a small but noticeable difference which decreases as the number of processors increases. This implies that either (1) the case where none of the tasks have a response time less than deadline is very rare, or (2) whenever that case does occur, the choice of using lateness does not have a large impact on the schedulability.

The average runtimes of all the algorithms are presented in Table I. It can be seen from the table that the combination of OPA and DA-LC is the fastest at the cost of the lowest acceptance ratio. The heuristics with the EPE-ZLL run slower due to the complexity of EPE-ZLL. Our proposed methods, MITER and HP-MITER run several magnitudes slower than other methods. This is expected as in the worst case, the MITER will be called n^2 number of times. Omitting Lines 22–24 (lateness) could result in a better runtime but sacrifice the acceptance for up to 9% when compared to the original version of HP-MITER. As the proposed algorithm is used in an offline setting and the aim is to find more schedulable tasksets that are deemed unschedulable by existing methods, the runtime of HP-MITER remains acceptable for the acceptance ratio that it provides.

B. Second Experiment

In the second experiment, we make an effort to compare with the ML framework [25]. We follow the same experimental settings as in [25] and use ZLL [15] as a common method for comparison. Specifically, we consider systems with $m = 2, 4, 6$ processors, and vary the number of tasks n as follows: $m = 2, n \in [6, 15]$; $m = 4, n \in [11, 20]$; and $m = 6, n \in [16, 25]$. The task parameters are generated as follows:



(a) $m = 2$ with $6 \le n \le 15$ (b) $m = 4$ with $11 \le n \le 20$ (c) $m = 6$ with $16 \le n \le 25$

Fig. 3. Percentage of schedulable tasksets with varying number of tasks

- Utilization: When setting the task utilizations in a taskset, first one of the following ten distributions is selected randomly: binomial and exponential distributions, each with constants 0.1, 0.3, 0.5, 0.7, and 0.9. Then for each of the tasks, its utilization is generated according to the same selected distribution.
- Period: it is randomly generated following a log-uniform distribution in the range of [10,1000].
- Execution time: it is calculated by the multiplication of $T_i \times U_i$ for each task τ_i.
- Deadline: all tasks have implicit deadlines, i.e., $D_i = T_i$.

After these task parameters are created, the taskset is tested and discarded if it is (1) schedulable by RTA-LC with the heuristics DkC [24], D-CMPO [11] or DMPO [10], (2) not schedulable according to the C-RTA condition [2], which is a necessary but not sufficient condition for a taskset to be schedulable by RTA-LC [8], and (3) schedulable by DA-LC test [2] with Audsley's OPA [5]. For each (m, n) pair, 1000 tasksets are generated.

Below is the list of methods compared in this experiment:

- ZLL: The baseline in [25] using the three heuristic priority assignment policies (DkC, DMPO, D-CMPO) with ZLL as the schedulability analysis [15]. That is, a taskset is schedulable by this baseline if any of the three policies with ZLL deems it schedulable.
- EPE-ZLL: The same as the above method except using EPE-ZLL [9] as the schedulability analysis instead of ZLL [15].
- MITER: The same as in the previous experiment.
- HP-MITER: The same as in the previous experiment.

We present the results in Figure 3, where the y-axis gives the percentage of schedulable tasksets among those that are deemed schedulable by at least one of the four methods. In all cases, across all values of n, HP-MITER is able to find a feasible priority ordering for at least 85% of the time, consistently outperforming all the other methods. The *average* difference between HP-MITER and ZLL is around 10% for $m = 2$, 35% for $m = 4$, and 55% for $m = 6$. As reported in [25], the ML framework is outperformed by ZLL when (1) $m = 2, n \in [11, 15]$; (2) $m = 4, n = 20$; (3) $m = 6, n = 21, 24, 25$. Even when ML is better than ZLL, the *maximum* advantage is about 40%. These results provide an indirect proof that HP-MITER is potentially able to find schedulable priority assignments for more tasksets than the ML framework. It must be noted that this result is expected as [25] uses RTA-LC, which is a less accurate schedulability test than EPE-ZLL, as its basis.

VII. Conclusion and Future Work

In this paper, we consider the problem of priority assignment for global fixed priority scheduling on a multiprocessor platform. We first propose a framework, MITER, that leverages the concept of response time estimation range. It remains optimal for a broader range of response time analysis techniques than Audsley's optimal priority assignment algorithm. We then present an algorithm that judiciously takes advantage of both MITER and heuristic algorithms. Experimental results with various synthetic tasksets show that the proposed approach significantly outperforms the state-of-the-art algorithms. For future work, we consider extending our optimization framework to different scheduling algorithms and task models, such as global dynamic priority scheduling and the arbitrary deadline task model.

Acknowledgment

The authors acknowledge Advanced Research Computing at Virginia Tech for providing the necessary computational resources to conduct the experiments. This paper serves as an extended work of the conference version at RTAS'18 [36].

References

[1] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[2] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[3] R. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016.

[4] V. Bonifaci and A. Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63:763–780, 2012.

[5] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, 2009.

[7] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE International Real-Time Systems Symposium*, pages 149–160, 2007.
[8] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *30th IEEE Real-Time Systems Symposium*, pages 387–397, 2009.
[9] Q. Zhou, J. Li, and G. Li. Excluding parallel execution to improve global fixed priority response time analysis. *ACM Transactions on Embedded Computing Systems*, 20(5s):1–24, 2021.
[10] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.
[11] M. Bertogna. *Real-Time Scheduling Analysis for Multiprocessor Platforms.* PhD thesis, Scuola Superiore Sant'Anna, Pisa, 2007.
[12] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys*, 43(4):1–44, 2011.
[13] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *IEEE Real-Time Systems Symposium, Work-in-Progress Session*, pages 19–23, 2000.
[14] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 119–128, 2007.
[15] Q. Zhou, G. Li, and J. Li. Improved carry-in workload estimation for global multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2527–2538, 2017.
[16] Y. Sun and M. Di Natale. Assessing the pessimism of current multicore global fixed-priority schedulability analysis. In *33rd ACM Symposium on Applied Computing*, pages 575–583, 2018.
[17] Y. Sun and M. Di Natale. Pessimism in multicore global schedulability analysis. *Journal of Systems Architecture*, 97:142–152, 2019.
[18] Liliana Cucu and Joel Goossens. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Design, Automation & Test in Europe Conference*, pages 1–6, 2007.
[19] A. Burmyakov, E. Bini, and C. Lee. Towards a tractable exact test for global multiprocessor fixed priority scheduling. *IEEE Transactions on Computers*, 71(11):2955–2967, 2022.
[20] T. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *International Conference On Principles Of DIstributed Systems*, pages 62–75. Springer, 2007.
[21] A. Burmyakov, E. Bini, and E. Tovar. An exact schedulability test for global fp using state space pruning. In *23rd International Conference on Real Time and Networks Systems*, pages 225–234, 2015.
[22] Y. Sun and G. Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*, 52:323–355, 2016.
[23] Liliana Cucu. Optimal priority assignment for periodic tasks on unrelated processors. In *Euromicro Conference on Real-Time Systems, WIP session*, 2008.
[24] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *30th IEEE Real-Time Systems Symposium*, pages 398–409, 2009.
[25] S. Lee, H. Baek, H. Woo, K. Shin, and J. Lee. Ml for rt: Priority assignment using machine learning. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 118–130, 2021.
[26] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with symta/s - symbolic timing analysis for systems. In *IEEE Real-Time Systems Symposium*, 2004.
[27] M. Saksena and Yun Wang. Scalable real-time system design using preemption thresholds. In *IEEE Real-Time Systems Symposium*, 2000.
[28] Haibo Zeng, Marco Di Natale, and Qi Zhu. Minimizing stack and communication memory usage in real-time embedded applications. *ACM Trans. Embed. Comput. Syst.*, 13(5s):1–25, July 2014.
[29] C. Wang, Z. Gu, and H. Zeng. Global fixed priority scheduling with preemption threshold: Schedulability analysis and stack size minimization. *IEEE Trans. Parallel and Distributed Systems*, 27(11):3242–3255, Nov. 2016.
[30] Yun Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *International Conference on Real-Time Computing Systems and Applications*, 1999.
[31] 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 Trans. Embed. Comput. Syst.*, 11(4):1–30, January 2013.
[32] Y. Zhao and H. Zeng. The concept of unschedulability core for optimizing priority assignment in real-time systems. In *Conference on Design, Automation and Test in Europe*, 2017.
[33] Y. Zhao and H. Zeng. The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In *IEEE Real-Time Systems Symposium*, 2017.
[34] Yecheng Zhao, Vinit Gala, and Haibo Zeng. A unified framework for period and priority optimization in distributed hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2188–2199, 2018.
[35] Shamit Bansal, Yecheng Zhao, Haibo Zeng, and Kehua Yang. Optimal implementation of simulink models on multicore architectures with partitioned fixed priority scheduling. In *IEEE Real-Time Systems Symposium*, pages 242–253, 2018.
[36] Y. Zhao and H. Zeng. The concept of response time estimation range for optimizing systems scheduled with fixed priority. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 283–294, 2018.

Xuanliang Deng is currently a Ph.D. student in the Electrical and Computer Engineering department at Virginia Tech. He received his master's degree in Electrical and Computer Engineering from Georgia Institute of Technology. His research interests include real-time system scheduling and optimization.

Shriram Raja completed his Bachelor's in Electrical and Electronics Engineering at PSG College of Technology, India and received his M.Eng. degree in Computer Engineering from Virginia Tech in 2023. He is currently pursuing a Ph.D. at Boston University. His area of interest is real-time operating systems.

Yecheng Zhao received his B.E. in Electrical Engineering from Harbin Institute of Technology, Harbin, China, and his Ph.D. degree in Computer Engineering from Virginia Tech. His main research focus is design optimization for real-time embedded systems. He is currently working as a software engineer with Google.

Haibo Zeng is with the Department of Electrical and Computer Engineering at Virginia Tech, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from the University of California at Berkeley. He was a senior researcher at General Motors R&D until October 2011, and an assistant professor at McGill University until August 2014. His research interests are embedded systems, cyber-physical systems, and real-time systems. He received five paper awards in the above fields.