ILP Representations of Multi-Phase Limited-Preemption Tasks

Benjamin Standaert*, Marion Sudvarg†, Fatima Raadia‡, Christopher Gill*
*Department of Computer Science & Engineering, Washington University in St. Louis, St. Louis, Missouri, United States
†Department of Physics, Washington University in St. Louis, St. Louis, Missouri, United States
‡Department of Computer Science, Wayne State University, Detroit, Michigan, United States
b.g.standaert@wustl.edu, msudvarg@wustl.edu, fatima.fr@wayne.edu, cdgill@wustl.edu

***Abstract*—Prior work considers the multi-phase secure (MPS) task model, in which tasks execute across multiple security domains that have significant preemption costs. Balancing blocking times with preemption overhead to guarantee schedulability requires finding an optimal placement of preemption points, which prior work has addressed for earliest deadline first (EDF) scheduling of MPS tasks. Here, we present fixed-priority (FP) algorithms for this task model, and investigate whether both the FP and EDF problems can be represented and solved as an ILP. We determine that an ILP solution is feasible, but does not improve performance for single-core problems. However, we show that the ILP can be extended to partitioned scheduling on identical multiprocessors, where performance is improved over an iterative algorithm.**

***Index Terms*—real-time systems, limited-preemption scheduling, multi-phase secure tasks, mixed-integer linear programming**

I. Introduction

In modern computing systems, tasks often execute in phases across multiple computational domains. For example, security-sensitive phases may require isolation in a trusted execution environment (TEE); other phases might be offloaded to a GPU or FPGA [1]. In these contexts, task preemption may introduce significant overhead – for example, switching into a TEE has been found to incur a penalty exceeding 100 µs [2]–[4]. Furthermore, the cost of preemption depends on the domain in which the phase executes – for example, if a task is preempted while executing inside a TEE, the preemption cost may include encrypting and copying the data out of the enclave.

If a fully-preemptive scheduler is used, the preemption cost may inflate execution times dramatically, causing the task system to become unschedulable. As such, for domain with high preemption costs (such as TEE execution), many systems schedule phases non-preemptively. However, this may lead to priority inversion, with high-priority tasks blocking on long-executing phases of lower priority tasks and missing deadlines.

In the limited-preemption task model [5] that was introduced to deal with such problems, tasks are allowed to run non-preemptively only part of the time, which reduces the preemption overhead while bounding blocking times. The *multi-phase secure* (MPS) task model extends limited preemption to multi-phase tasks with domain-specific preemption overheads [1]. In the MPS model, the startup and teardown overheads of an execution domain are unavoidable during phase transitions, so these serve as natural preemption points. Additional preemption points can be inserted into the middle of a phase as necessary. Under these assumptions, [1], [6] present iterative algorithms to find (if it exists) the minimum number of preemption points to insert into each task phase such that there is an acceptable tradeoff between preemption overhead and blocking time to guarantee EDF schedulability.

Our goal in this paper is to consider whether ILPs are useful for expressing and solving the problem of finding preemption points to guarantee schedulability in MPS task systems. Indeed, in [7] – in which the traditional limited-preemption model is extended to select from a fixed set of potential preemption points with unique costs – the authors note that while such problems may be solved using ILPs, execution time may scale exponentially with the number of preemption points, hypothesizing that they could rapidly become infeasible. However, under the MPS model, only the *number* of preemption points must be found, suggesting that it may be possible to solve the problem using an ILP with many fewer constraints and consequently better scaling behavior.

Moreover, in the iterative algorithm for EDF scheduling of MPS tasks, the problem of computing the number of preemption points might require evaluation of every time point in an exponentially-sized testing set [1], [6], which could result in poor performance on large task systems. While ILPs in general also have exponential time complexity, off-the-shelf solvers often achieve very good performance on many problems. This paper therefore considers ILPs as an alternative approach for both EDF and fixed-priority scheduling of MPS task sets, implementing them using SCIP [8], an open-source ILP solver, to evaluate their execution time feasibility.

To provide a basis of comparison for the ILPs, we use the algorithm of [6] for limited-preemption EDF scheduling of MPS tasks. For fixed-priority scheduling, we adapt the algorithm of [9] to the MPS model, adjusting it to insert the minimum feasible number of preemption points while accounting for different preemption costs across phases. Though we find for these algorithms that ILPs might not improve execution times, they are still efficiently solvable, making them feasible to use during offline scheduling decisions. Furthermore, the added expressiveness of formulating the problem in this way gives rise to extensions of the model, e.g., to partitioned fixed-priority scheduling on a multiprocessor. We show that solving such an ILP with SCIP gives better execution time performance than an iterative partitioning algorithm.

II. Background and Related Work

Limited-preemption scheduling balances the increased blocking times of non-preemptive execution with the increased overheads caused by context switching. The original model minimizes context switching by finding the longest time each task may execute non-preemptively without compromising schedulability [5], [10], but does not account for the worst-case overhead due to the resulting preemptions. Later approaches account for both blocking time and preemption overheads by defining instants that preemption can occur during task execution (called *preemption points*) to guarantee schedulability. In [11], an algorithm is developed to find preemption points for tasks scheduled with EDF under the assumption that the preemption overhead for each task is constant. A similar model for fixed-priority (FP) scheduling was developed in [9]. In [7], tasks are modeled as sequences of "basic blocks," and algorithms are presented for selecting preemption points between those blocks for both EDF and FP scheduling. Although less flexible than the earlier "floating" preemption point models where a preemption point can be placed anywhere, that model allows different preemption costs between blocks.

The **multi-phase secure** (MPS) task model was introduced in [1]. Motivated by tasks that execute in phases sequentially across different security domains (e.g., TEEs) with unique preemption costs, the model considers a set of constrained-deadline, sporadic tasks $\{\tau_i : \tau_i \in \Gamma\}$ characterized by a period (minimum inter-arrival time) T_i and relative deadline $D_i \leq T_i$, both of which are assumed to be integers ≥ 1, and a sequence $\{\tau_{i,j}\}$ of phases. Each phase represents a block of work that occurs in some domain – e.g., an initial phase that runs in non-secure mode on a CPU, followed by a phase that must run in a TEE or co-processor. Each phase $\tau_{i,j}$ has an associated execution time $c_{i,j}$ and a setup/teardown cost $q_{i,j}$ that must be paid at least once when switching into and out of the phase, and again whenever it is preempted.

From the analysis in [1], preemption points spaced evenly within a phase minimize blocking time in systems with sporadic or asynchronous job releases. An integer $x_{i,j} \geq 1$ denotes the number of equally-sized non-preemptive segments in which phase $\tau_{i,j}$ may execute, allowing no more than $x_{i,j}-1$ preemptions. Consequently,

$$\beta_{i,j} = \frac{c_{i,j}}{x_{i,j}} + q_{i,j} \tag{1}$$

represents the maximum time that phase $\tau_{i,j}$ may block execution of higher-priority tasks. Consequently, $\beta_i = \max_j\{\beta_{i,j}\}$ is the maximum blocking time imposed by task τ_i. The total execution time, with overheads, of task τ_i is

$$C_i = \sum_j c_{i,j} + x_{i,j} q_{i,j}. \tag{2}$$

The problem is to assign values (if they exist) to each term $x_{i,j}$ to guarantee schedulability. An algorithm to find *minimum* values for each $x_{i,j}$, thereby minimizing C_i, for EDF scheduling of MPS task systems is presented in [1] and improved in [6] with pseudo-polynomial running time for bounded-utilization task sets. The latter is similar to the algorithm in [11], but supports domain-specific preemption costs rather than a constant overhead per task. In this paper, we similarly extend the algorithm in [9] to FP scheduling of MPS tasks, then construct ILPs to solve both problems.

Limited-preemption also has been extended to **partitioned scheduling** in multi-core systems. By transformation from bin packing, partitioned scheduling is by itself NP-complete in the strong sense [12]. However, heuristics exist that often can find feasible partitions if they exist [13]. These heuristics are extended to limited-preemption tasks in [14]; however, they do not provide an exact solution, and have not been fully integrated with the MPS task model. In this paper, we construct an ILP for exact schedulability analysis of partitioned FP scheduling of MPS tasks.

Other work on limited-preemption scheduling [15], including cache-aware analysis [16], probabilistic [17] or "typical" execution models [18], and conditional control flows [19], [20] are outside the scope of this paper.

III. EDF Scheduling of MPS Tasks

In this paper, we propose an ILP as an alternative to the iterative algorithms in [1], [6] for EDF schedulability analysis of MPS tasks. EDF schedulability analysis for constrained-deadline tasks is coNP-complete in general [21], [22], and requires checking processor demand at a number of time points that may grow exponentially with the number of tasks. An ILP therefore may be an attractive alternative where the testing set (the set of time points to check) is very large.

A. Background

The algorithm in [1] is a two-step approach to solving the limited-preemption EDF problem. First, it iterates over every time point in the testing set $\mathcal{T} = \{t_d \equiv k \cdot T_i + D_i\}$ for $k \in \mathbb{N}$, stopping at $D_{\max} = \max_{\tau_i}\{D_i\}$. At each point, it checks the available "slack":

$$S(t_d) = t_d - \sum_{\tau_i} \text{DBF}(\tau_i, t_d) \tag{3}$$

where the demand-bound function DBF represents the maximum possible execution time required by instances of the task that have deadlines before t_d:

$$\text{DBF}(\tau_i, t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \cdot C_i \tag{4}$$

The algorithm then checks if any lower-priority task can block for longer than the available slack; if so, it inserts additional preemption points until the blocking time of each lower-priority task is less than or equal to the available slack.

Second, once preemption points are fixed, the algorithm tests the remaining time points in $\mathcal{T}$, ensuring that slack remains ≥ 0. As shown in [23], when total utilization $U < 1$, only points in $\mathcal{T}$ not exceeding the following upper bound must be checked:

$$\min\left(\text{lcm}(T_1 ... T_n), \max\left(D_{\max}, \frac{\sum_{i=1}^{n} U_i \cdot (T_i - D_i)}{1 - U}\right)\right).$$

The approach is further refined in [6] by identifying that for systems of implicit-deadline tasks ($D_i = T_i$), the second step can be replaced by a fast utilization check. Otherwise, however, the entire testing set must be checked up to the above bound. This set can become very large for systems with many tasks, particularly as the utilization approaches 1.

B. ILP Representation

The longer second stage of the described algorithm can be replaced with an ILP, where the objective is to find the minimum slack at any time point t_d, and check whether the result is ≥ 0. After running the first stage, we first check the total utilization. If $U > 1$, we return unschedulable. Otherwise, we construct and solve the following ILP:

1) Define a variable $T \geq D_{\max}$, representing the t_d with the minimum slack. To avoid issues with numerical instability in SCIP, we constrain $T \leq H$, where H is the hyperperiod.
2) Next, construct a representation of Equation 4. For each task τ_i, define an integer variable Z_i that represents the number of job releases in the time up to t_d; in other words,

$$Z_i = \max\left(\left\lfloor \frac{T - D_i}{T_i} \right\rfloor + 1, 0\right).$$

Since $T \geq D_{\max}$, the first term of the max function is non-negative and thus $Z_i = \left\lfloor \frac{T-D_i}{T_i} \right\rfloor + 1$. Since Z_i is an integer, this is enforced by the following constraint:

$$\frac{T - D_i}{T_i} < Z_i \leq \frac{T - D_i}{T_i} + 1$$

SCIP does not support strict inequalities, so the $<$ on the LHS is replaced with $\leq$. However, this does not change the solution produced: if the solver finds a solution where $\frac{T-D_i}{T_i}$ is a valid integer solution to Z_i, then $\frac{T-D_i}{T_i} + 1$ is also a valid integer solution. We have structured our ILP to minimize slack, and increasing Z_i (i.e. increasing the number of times a task runs up to some time point) will always increase the DBF, and therefore reduce slack. Therefore, the ILP solver will always choose $\frac{T-D_i}{T_i} + 1$ as the solution to Z_i given either form of the constraint.
3) From Equation 3, the slack at any time point T then becomes $S = T - \sum_{\tau_i} Z_i C_i$. As we are seeking to minimize the slack across all T, we can turn this into a constraint:

$$S \geq T - \sum_{\tau_i} Z_i C_i$$

Finally, to determine schedulability, we execute the solver to **minimize** $\boldsymbol{S}$ and check that $S \geq 0$.

IV. FP Scheduling of MPS Tasks

From [24], a fixed-priority, constrained-deadline task system (indexed in decreasing priority order) that experiences blocking (as in the MPS model) is schedulable if and only if:

$$\forall \tau_i,\ \exists t \leq D_i : t \geq C_i + \left(\sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil \times C_k\right) + \max_{\forall k > i}\{\beta_k\} \quad (5)$$

where β_j represents the maximum amount of time that τ_j can block other tasks, e.g., by running non-preemptively.

Algorithm 1 Fixed-Priority MPS Task Scheduling

```
Input: Set $\Gamma$ of $n$ constrained-deadline MPS tasks
Output: Values $x_{i,j}$ indicating the number of equal-sized
    non-preemptive regions in each phase $\tau_{i,j}$

$\beta_1 = \infty$                              ▷ Max blocking time by $\tau_1$
$C_1 = 0$                                       ▷ Execution time of $\tau_1$
for all $\tau_{1,j} \in \tau_1$ do
    $x_{1,j} = 1$
    $C_1 = C_1 + c_{1,j} + q_{1,j}$
end for
$\hat{B}_1 = D_1 - C_1$                         ▷ Blocking tolerance of $\tau_1$

if $\hat{B}_1 < 0$ then return Not Schedulable

for $i = 2, \ldots, n$ do                       ▷ Remaining tasks
    $\beta_i = \min(\beta_{i-1}, \hat{B}_{i-1})$    ▷ Max blocking time by $\tau_i$
    $C_i = 0$                                   ▷ Execution time of $\tau_i$
    for all $\tau_{i,j} \in \tau_i$ do          ▷ Assign $x_{i,j}$ for each phase
        if $q_{i,j} \geq \beta_i$ then return Not Schedulable
        $x_{i,j} = \left\lceil \frac{c_{i,j}}{\beta_i - q_{i,j}} \right\rceil$
        $C_i = C_i + c_{i,j} + x_{i,j} \cdot q_{i,j}$
    end for
    $\hat{B}_i = \max_{t \in \mathcal{T}_i}\left(t - C_i - \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil \times C_k\right)$
    if $\hat{B}_i < 0$ then return Not Schedulable
end for
return $\{x_{i,j}\}$
```

In this section, we present two algorithms to assign values $x_{i,j}$ (the number of non-preemptive chunks in each phase) to guarantee FP schedulability of MPS tasks.

A. Iterative Solution

[9, Algorithm 2] is an iterative procedure for finding the maximum length of time β_i that each task τ_i may run without preemption while allowing the system to remain schedulable. Here, we adapt that algorithm to MPS task systems, making a few key changes. Our procedure is outlined in Algorithm 1, using the notation introduced earlier in this paper, which follows the more recent notation in [1].

The algorithm iterates over tasks in descending priority order, computing for each task τ_i the maximum time β_i that it can block higher-priority tasks without compromising schedulability. It also computes the blocking tolerance $\hat{B}_i$ for each task τ_i. This is defined in [9] as the maximum time τ_i can be blocked by lower priority tasks while still meeting its deadline, i.e., the maximum value that $\max_{j>1}\{\beta_j\}$ can take such that the recurrence in Equation 5 is satisfied.

For τ_1, the blocking tolerance is just its slack $D_1 - C_1$, and $\beta_1 = \infty$ since there are no higher-priority tasks for it to block. For subsequent tasks, the maximum allowed blocking time β_i is just the minimum blocking tolerance of all higher-priority tasks; the iteration order guarantees these values have already been computed. Unlike [9, Algorithm 2], lines 17–20 of our algorithm compute the resulting number of non-preemptive regions for *each task phase*, as well as the resulting task execution times. The blocking tolerance $\hat{B}_i$ is then computed on line 22 by testing times $t_d \leq D_i$ in the testing set $\mathcal{T}_i$, i.e., those times where the RHS of Equation 5 can change: $\mathcal{T}_i = \{t_d \equiv m \cdot T_k + D_k\}$ for $m \in \mathbb{N}$ and $k < i$.

We note that [9, Algorithm 2] computes the maximum length of non-preemptible regions of each task under the assumption that task execution times remain unchanged, i.e., the addition of preemption points does not affect a task's blocking tolerance. This means that the given task set has to be checked a priori for feasibility when scheduled fully preemptively without overheads, but schedulability does not have to be reconfirmed while the procedure executes. In contrast, our algorithm has to compute $\hat{B}_i$ for each task after preemption points are assigned to each phase, since the resulting overhead impacts the task's execution time. In line 23, if the blocking tolerance is negative, the task set is deemed unschedulable – this check removes the need for a prior feasibility test while also ensuring that the task set remains schedulable as preemption points are inserted.

B. ILP Solution

Starting with the problem definition above, we build on the ILP representation of FP response-time analysis in [25] and the representation of the FP constrained-deadline elastic scheduling problem in [26]. An integer variable $x_{i,j} \geq 1$, as defined in Section II, indicates how many times phase $\tau_{i,j}$ of task τ_i may run non-preemptively. The worst-case execution time of τ_i is given by Equation 2, and the blocking time β_i of the task is thus the maximum blocking time among its phases, $\beta_i = \max_j\{\beta_{i,j}\}$, with $\beta_{i,j}$ given by Equation 1.

The following steps construct an ILP for Equation 5:

1) For each task τ_i, define a real-valued variable $0 < t_i \leq D_i$ representing a value of t for which Equation 5 holds.
2) For each pair of tasks τ_i, τ_k with $k < i$, define an integer variable $Z_{i,k} \geq \frac{t_i}{T_k}$. Because $Z_{i,k}$ is an integer, it will respect the ceiling operator ($\lceil\ \rceil$) in Equation 5. This represents the number of jobs of the higher-priority task τ_k that can interfere with a job of τ_i. Note that we do not provide any upper bound on $Z_{i,k}$; however, for some t, decreasing $Z_{i,k}$ can only reduce the right-hand side of Equation 5. As the ILP solver must make the right-hand side of Equation 5 small enough for some t in order to form a valid solution, it will naturally minimize this variable if needed.
3) For every task τ_i, define a real variable β_i that represents the time τ_i can block higher-priority tasks. This is the maximum blocking time among its phases, so for each phase, add a constraint $\beta_i \geq \frac{c_{i,j}}{x_{i,j}} + q_{i,j}$.
4) For every task τ_i, define a variable B_i representing the maximum time it can be blocked by lower-priority tasks. Then for every pair of tasks τ_i, τ_j with $j > i$, we add the constraint $B_i \geq \beta_j$.
5) For every task τ_i, add a final constraint for Equation 5, where n_i represents the number of phases in τ_i:

$$\sum_{j=1}^{n_i} c_{i,j} + \sum_{j=1}^{n_i} q_{i,j} x_{i,j} + \sum_{k=1}^{i-1} Z_{i,k} C_k + \sum_{k=1}^{i-1} \sum_{j=1}^{n_k} q_{k,j} Z_{i,k} x_{k,j} + B_i \leq t_i \tag{6}$$

If the task system is schedulable when introducing preemption points, then the ILP solver will be find a set of values $x_{i,j}$ that satisfy all constraints at some t_i for each task.

Our ILP can be configured either to find a feasible solution – some combination of $x_{i,j}$ that ensures the system is schedulable – or an optimal solution, which additionally minimizes the total preemption cost. To find an optimal solution, construct the ILP to additionally **minimize** $\sum_{i,j} x_{i,j} \frac{q_{i,j}}{T_i}$ since each term is proportional to the amount by which increasing $x_{i,j}$ increases utilization.

V. Partitioned Fixed-Priority Scheduling

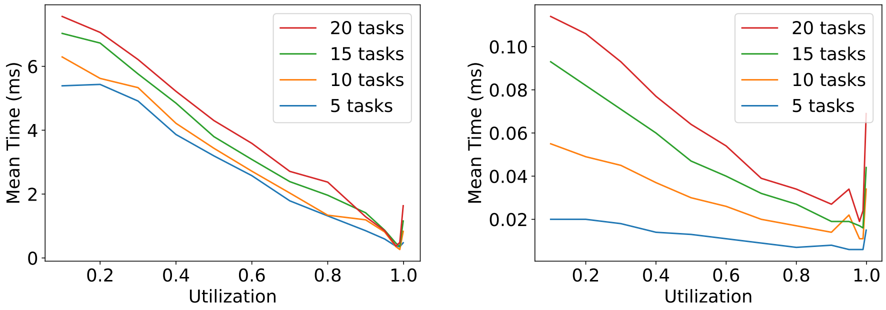
In this section, we illustrate one of the benefits of using an ILP to express scheduling problems for MPS tasks – namely, that the expressive power of ILPs allows straightforward extensions with additional constraints, e.g., for alternative scheduling models. Here, using an approach inspired by [27], we extend our ILP from Section IV-B to partitioned FP scheduling across m identical processors.

As in [27], for every pair of tasks τ_i, τ_k, we define a zero-one variable $s_{i,k}$ with the intended interpretation that it takes the value 1 if the two tasks are scheduled on the same processor, or 0 otherwise. The tasks are schedulable if and only if there exists, for each task τ_i, a $t_i \leq D_i$ that satisfies

$$t_i \geq C_i + \left(\sum_{k<i} \left\lceil \frac{t_i}{T_k} \right\rceil C_k \cdot s_{i,k} \right) + \max_{\forall k>i}\{\beta_k \cdot s_{i,k}\} \tag{7}$$

In other words, for each task, only other tasks that run on the same processor will contribute towards the execution time and blocking time portions of the recurrence. Similarly to [27], we first set up constraints to enforce our definition of $s_{i,k}$:

1) For each task τ_i and processor p, define a zero-one variable $z_{i,p}$, with the interpretation that it takes the value 1 if and only if τ_i executes on processor p. To represent that each task should be assigned to exactly one processor, we add a constraint for each τ_i that $\sum_{p=1}^{m} z_{i,p} = 1$.
2) For each pair of tasks τ_i, τ_k with $i < k$, and each processor p, define a zero-one variable $y_{i,k,p}$ with the intended interpretation that it takes the value 1 if and only if τ_i and τ_k both execute on processor p. We enforce this using a SCIP AND constraint, $y_{i,k,p} = z_{i,p} \wedge z_{k,p}$. For solvers that do not support boolean expressions, this can instead be enforced with linear constraints using established linearization techniques [28].
3) Then, construct a constraint for $s_{i,k}$ by summing over all possible processor cores: $s_{i,k} = \sum_{p=1}^{m} y_{i,k,p}$.



(a) EDF ILP solution

(b) EDF iterative solution

Fig. 1: Mean times to determine schedulability and find preemption points. Note the different y-axis scales. The vertical ordering of the series matches the ordering of the legend.

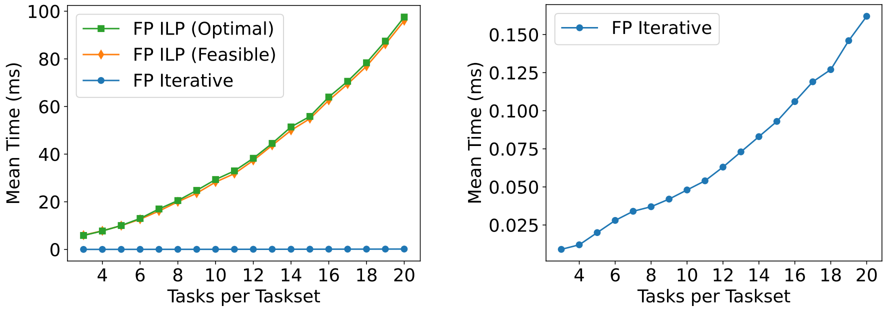


Fig. 2: Mean times for FP scheduling. Note the different y-axis scales.

Then, we add the same constraints as our formulation in section IV-B, while making the following changes:

1) We update our definition of $Z_{i,k}$ so that it represents the term $\left\lceil \frac{t_i}{T_k} \right\rceil \cdot s_{ik}$ with only linear constraints. As in [27], it is constrained as $Z_{i,k} \geq 0$ and

$$Z_{i,k} \geq \frac{t_i}{T_k} - M_1(1 - s_{i,k}) \tag{8}$$

where the constant $M_1 = D_{\max} + 1$. If $s_{i,k} = 0$, indicating that the tasks run on different cores, then $M_1(1 - s_{i,k}) = M_1$. Since $T_k \geq 1$ and t_i is constrained as $t_i \leq D_{\max}$, this is sufficient to guarantee that the RHS of (8) evaluates to a negative value, and so $Z_{i,k}$ takes the value 0. Otherwise, if $s_{i,k} = 1$, then $M_1(1 - s_{i,k}) = 0$, and since $Z_{i,k}$ is an integer it takes the value $\left\lceil \frac{t_i}{T_k} \right\rceil$.

2) Similarly, we must update our definition of B_i so that it represents the term $\max_{k>i}\{\beta_k \cdot s_{i,k}\}$ with only linear constraints. Here, we let $M_2 = D_{\max} + 1$; this value will be larger than the WCET of any task, and consequently, larger than any task's maximum possible blocking time. Then for every pair of tasks τ_i, τ_k with $k > i$, we add constraints of the form $B_i \geq 0$ and

$$B_i \geq \beta_k - M_2(1 - s_{i,k}) \tag{9}$$

By the same logic as before, $B_i \geq \beta_k$ if and only if τ_i and τ_k execute on the same core.

With these changes, we add a constraint of the form in Equation 6. An MPS task set is partitioned-FP schedulable on m identical cores if and only if a set of values $x_{i,j}$ describing the preemption points for each task phase and $z_{i,p}$ describing processor assignments are found to satisfy the constructed ILP.

VI. Evaluation

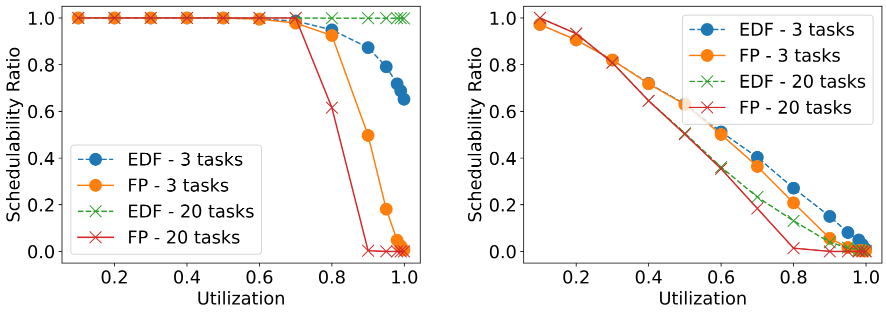
We evaluate the performance of our algorithms using a C++ simulation on randomly-generated synthetic MPS task sets, into which we link version 8 [29] of SCIP [8] to solve the ILP. Following the methodology in [1], the number of phases for each task is selected uniformly from 1–4, and task periods are selected uniformly from the integers 10–30.

All tests were performed on a server with two Intel Xeon Gold 6130 (Skylake) processors running at 2.1 GHz, and with 64GB of memory. Multiple task sets were evaluated in parallel; each task set was given a single thread in which to run.

A. EDF ILP

To test the performance of our EDF ILP in Section III-B, we generate task sets of size n from 3–20 with total utilizations $U = 0.1 \ldots 0.9$ in increments of 0.1. [6] hypothesized that the iterative algorithm would run more slowly for task sets with utilizations close to 1; therefore, we also test $U = \{0.95, 0.98, 0.99, 0.999\}$. Total utilization is distributed among tasks using the UUniFast algorithm [30], which follows a uniform random distribution. As in [6], after periods are also generated, UUniFast is invoked again for each task to distribute its resulting total execution time among the execution times $c_{i,j}$ and setup/teardown costs $q_{i,j}$ for each of its phases.

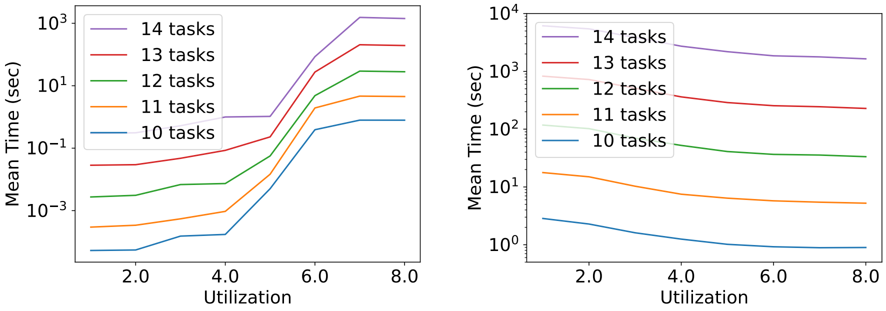
As the ILP step is unnecessary for implicit-deadline tasks, we randomly select a deadline uniformly from the integers between the task's execution time and period. For each combination (n, U), we generate 1000 task sets. For each test, we



(a) Implicit deadlines

(b) Constrained deadlines

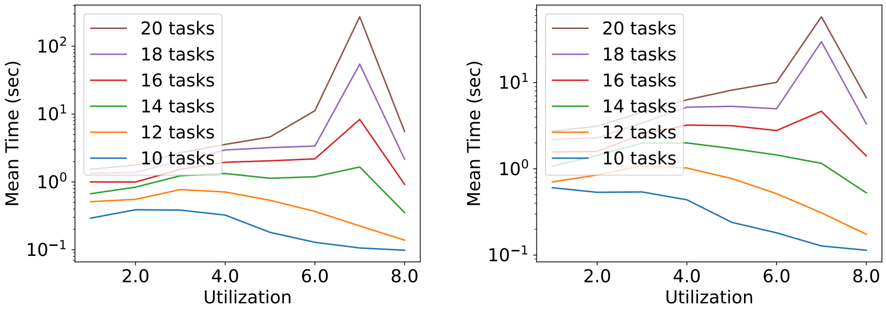
Fig. 3: Comparison of EDF and FP schedulability.



(a) Feasible

(b) Optimal

Fig. 4: Mean times for partitioned FP iterative solution. Note the log scale of the y-axis. The vertical ordering of the series matches the ordering of the legend.



(a) Feasible

(b) Optimal

Fig. 5: Mean times for partitioned FP ILP solution. Note the log scale of the y-axis. The vertical ordering of the series matches the ordering of the legend.

run the EDF ILP from Section III-B and the iterative procedure in [6, Algorithm 1] and record their execution times.

Results are shown in Figure 1. Although the execution time of the iterative algorithm does increase when U is very close to 1, the increase is not as large as hypothesized. One contributing factor to this is that in a constrained-deadline system, the schedulability ratio of tasks where U is close to 1 is very low – for example, with 20 tasks, less than 1% of the task systems are schedulable when $U \geq 0.98$. As the iterative algorithm terminates once it deems a task set unschedulable, it rarely evaluates the entire testing set. We further observe that the ILP generally exhibits execution times that are orders of magnitude higher than the iterative algorithm. Nonetheless, it completes in under 8ms on average, remaining feasible as an approach to schedulability analysis.

B. *Fixed-Priority Uniprocessor ILP*

To evaluate our FP uniprocessor implementations, we consider sets of n tasks from 3–20 with total utilizations U from 0.1–1.0 in increments of 0.1. For each combination (n, U), we generate 1000 task sets according to the above methodology. Tasks are assigned rate-monotonic (RM) priorities.

Figure 2 compares the performance of the ILP and iterative approaches. We observed no significant performance differences when evaluating tasks with implicit deadlines compared to constrained deadlines generated as before. Figure 2 therefore displays just the results for implicit-deadline tasks with $U = 0.8$ as a representative example. We observe that the *feasible* (finding values $x_{i,j}$ to guarantee schedulability) and *optimal* (finding schedulable values of $x_{i,j}$ that also minimize utilization) versions of the ILP have almost equivalent average execution time behavior. Furthermore, although both the ILP and iterative approaches exhibit similar scaling behavior with the number of tasks, the ILP solution is much slower in all cases than the iterative solution.

For practitioners deciding between EDF and rate-monotonic FP scheduling for systems of MPS tasks, we also compare the schedulability of these approaches. Figure 3 shows the schedulability ratios as total utilization U approaches 1 for both constrained-deadline and implicit-deadline task sets, using the same total utilizations tested in Section VI-A. With implicit deadlines, schedulability ratios remains close to 1 for task sets with $U \leq 0.7$. For larger utilizations, EDF schedules more sets of 20 tasks than 3, whereas FP schedules more sets of 3 tasks than 20. This makes sense, since our method for generating tasks typically assigns larger preemption costs to systems with fewer tasks. Since the utilization bound of preemptive EDF is unaffected by the number of tasks, it makes sense that lower preemption costs yield better schedulability. Preemptive rate-monotonic scheduling, however, has its schedulable utilization decrease with larger task sets [31], an effect that we observe to dominate the improvements of smaller preemption overheads. For constrained deadlines, on the other hand, EDF schedulability is lower for 20 than 3 tasks.

C. Fixed-Priority Partitioned Multicore ILP

To evaluate the performance benefits of our partitioned FP multiprocessor solution, we compare it to a simple iterative algorithm that generates every possible partition of a given set of tasks onto a given number of cores, then uses Algorithm 1 to test the schedulability of each individual subset. When configured to find a feasible solution, our algorithm begins with more uniformly-distributed partitions, terminating once it finds a partition for which all subsets are schedulable. When configured to find an optimal solution, it iterates over all partitions to find the solution with the lowest preemption overhead. We note that there are many heuristics that may find schedulable partitions more quickly [14]; however, these approaches generally do not provide exact schedulablility analysis, as does our ILP approach.

We generate task sets of size n from 10–20 with total utilizations U from 1–8 in steps of 1. For each combination (n, U), we generate 50 sets using the above methodology. This time, however, we distribute utilizations uniformly using the Dirichlet Rescale (DRS) algorithm [32], [33], which allows us to limit each individual task's utilization to 0.8, ensuring that it can be partitioned successfully onto a CPU core.

As shown in Figure 4, iteratively testing all partitions quickly becomes infeasible as the number of tasks is increased. When finding a feasible solution, the solver can execute quickly when utilization is low, as the first few tested partitions are likely to be feasible. However, once utilization is increased, finding a feasible solution can take over 1000 seconds. When searching for an optimal solution, which requires testing all partitions, the solver took over an hour to run on sets of 14 tasks. For sets of 15 tasks, the solver was not able to finish running in a reasonable amount of time.

Figure 5 shows the ILP execution times, which are significantly faster. For sets of 14 tasks, the ILP can find a solution in just a few seconds. Even for sets of 20 tasks, the execution time of the ILP is on the order of hundreds of seconds – significantly faster than that of the iterative solution with just 14 tasks. Additionally, the gap in execution times between the feasible- and optimal-solution configurations of the ILP is much smaller; its execution times are reasonable for sets of 20 tasks even when configured to find an optimal solution.

VII. Conclusion

In this work, we evaluate the applicability of ILPs to the multi-phase secure task model defined in [1]. We demonstrate feasibility of an ILP representation for schedulability analysis under this model. Although these suffer from higher execution times than corresponding iterative algorithms for uniprocessor scheduling problems, we demonstrate that ILPs can be extended easily to account for additional scheduling constraints. Moreover, for partitioned FP scheduling on an identical multiprocessor, an ILP has significant performance advantages over an exact iterative solution.

As future work, we plan to investigate whether the EDF ILP can be further optimized by structuring it as a feasibility problem (i.e., identifying a time point with negative slack) rather than an optimization problem (finding the minimum slack). We also intend to investigate whether the partitioning heuristics in [14] can be used to guide the ILP and/or the iterative solution toward feasibility in less time while still finding an exact solution. We also intend to investigate whether, under this task model, analysis of upper bounds on the number of times a task can actually be preempted can benefit the model by reducing the accounted overhead, and consequently improving schedulability. Finally, we intend to investigate extensions to conditional execution models.

VIII. Acknowledgements

This work was supported by NSF grants CPS-2229290, CNS-2141256, CPS-2038609 and CNS-2211641 and a seed grant from Washington University in St. Louis. The authors would like to thank Dr. Sanjoy Baruah, Dr. Thidapat Chantem, and Dr. Nathan Fisher for their contributions towards developing the multi-phase secure model. The authors would also like to thank Dr. Ning Zhang, Ao Li, Jinwen Wang, and Tanmaya Mishra for their assistance in understanding the execution model of TEEs. Finally, the authors would like to thank Dr. Filip Marković for his assistance in providing information on task partitioning heuristics.

References

[1] S. Baruah, T. Chantem, N. Fisher, and F. Raadia, "A Scheduling Model Inspired by Security Considerations," in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 32–41. DOI: 10.1109/ISORC58943.2023.00016.

[2] A. Mukherjee, T. Mishra, T. Chantem, N. Fisher, and R. Gerdes, “Optimized trusted execution for hard real-time applications on COTS processors,” in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, 2019, pp. 50–60.

[3] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, “RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 352–369.

[4] M. F. Babar and M. Hasan, “DeepTrustˆ RT: Confidential Deep Neural Inference Meets Real-Time!” In *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[5] S. Baruah, “The limited-preemption uniprocessor scheduling of sporadic task systems,” in *17th Euromicro Conference on Real-Time Systems (ECRTS’05)*, 2005, pp. 137–144. DOI: 10.1109/ECRTS.2005.32.

[6] B. Standaert, F. Raadia, M. Sudvarg, *et al.*, *A Limited-Preemption Scheduling Model Inspired by Security Considerations*. [Online]. Available: https://openscholarship.wustl.edu/cse_facpubs/2/.

[7] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, “Optimal selection of preemption points to minimize preemption overhead,” in *2011 23rd Euromicro Conference on Real-Time Systems*, IEEE, 2011, pp. 217–227.

[8] T. Achterberg, “SCIP: Solving constraint integer programs,” *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, Jul. 2009, ISSN: 1867-2957. DOI: 10.1007/s12532-008-0001-1.

[9] G. Yao, G. Buttazzo, and M. Bertogna, “Feasibility analysis under fixed priority scheduling with limited preemptions,” *Real-Time Systems*, vol. 47, no. 3, pp. 198–223, 2011, Publisher: Springer.

[10] M. Bertogna and S. Baruah, “Limited Preemption EDF Scheduling of Sporadic Task Systems,” *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 579–591, Nov. 2010, ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2010.2049654.

[11] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, “Preemption Points Placement for Sporadic Task Sets,” in *2010 22nd Euromicro Conference on Real-Time Systems*, 2010, pp. 251–260. DOI: 10.1109/ECRTS.2010.9.

[12] R. I. Davis, A. Burns, J. Marinho, V. Nelis, S. M. Petters, and M. Bertogna, “Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Preemption,” *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 3, Apr. 2015, Place: New York, NY, USA Publisher: Association for Computing Machinery, ISSN: 1539-9087. DOI: 10.1145/2739954.

[13] J. M. López, J. L. Díaz, and D. F. García, “Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems,” *Real-Time Systems*, vol. 28, no. 1, pp. 39–68, Oct. 2004, ISSN: 1573-1383. DOI: 10.1023/B:TIME.0000033378.56741.14.

[14] F. Marković, J. Carlson, and R. Dobrin, “A Comparison of Partitioning Strategies for Fixed Points Based Limited Preemptive Scheduling,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1070–1081, 2019. DOI: 10.1109/TII.2018.2848879.

[15] G. C. Buttazzo, M. Bertogna, and G. Yao, “Limited Preemptive Scheduling for Real-Time Systems. A Survey,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, 2013. DOI: 10.1109/TII.2012.2188805.

[16] F. Marković, J. Carlson, and R. Dobrin, “Cache-aware response time analysis for real-time tasks with fixed preemption points,” in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 30–42. DOI: 10.1109/RTAS48715.2020.00-19.

[17] F. Markovic, J. Carlson, R. Dobrin, B. Lisper, and A. Thekkilakattil, “Probabilistic response time analysis for fixed preemption point selection,” in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018, pp. 1–10. DOI: 10.1109/SIES.2018.8442099.

[18] S. Baruah and N. Fisher, “Choosing preemption points to minimize typical running times,” in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, ser. RTNS ’19, Toulouse, France: Association for Computing Machinery, 2019, pp. 198–208, ISBN: 9781450372237. DOI: 10.1145/3356401.3356407.

[19] B. Peng, N. Fisher, and M. Bertogna, “Explicit preemption placement for real-time conditional code,” in *2014 26th Euromicro Conference on Real-Time Systems*, 2014, pp. 177–188. DOI: 10.1109/ECRTS.2014.25.

[20] F. Raadia, N. Fisher, T. Chantem, and S. Baruah, “An improved security-cognizant scheduling model,” in *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2024, pp. 1–8.

[21] P. Ekberg and W. Yi, “Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete,” in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, Jul. 2015, pp. 281–286. DOI: 10.1109/ECRTS.2015.32.

[22] P. Ekberg and W. Yi, “Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization,” in *Proceedings of the 36th Real-Time Systems Symposium (RTSS)*, 2015, pp. 87–95. DOI: 10.1109/RTSS.2015.16.

[23] S. K. Baruah, L. E. Rosier, and R. R. Howell, “Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor,” *Real-time systems*, vol. 2, no. 4, pp. 301–324, 1990, Publisher: Springer.

[24] E. Bini and G. C. Buttazzo, “Schedulability analysis of periodic fixed priority systems,” *IEEE Transactions*

on Computers, vol. 53, no. 11, pp. 1462–1473, 2004, Publisher: IEEE.

[25] S. Baruah and P. Ekberg, *An ILP representation of Response Time Analysis*, 2021. [Online]. Available: https://research.engineering.wustl.edu/~baruah/Submitted/2021-ILP-RTA.pdf.

[26] M. Sudvarg, S. Baruah, and C. Gill, "Elastic Scheduling for Fixed-Priority Constrained-Deadline Tasks," in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 11–20. DOI: 10.1109/ISORC58943.2023.00014.

[27] M. Sudvarg, D. Wang, J. Buhler, and C. Gill, "Subtask-Level Elastic Scheduling," in *Proceedings of the 45th Real-Time Systems Symposium (RTSS)*, Dec. 2024.

[28] L. A. Wolsey, *Integer Programming*, 2nd. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2021.

[29] K. Bestuzheva *et al.*, "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, Dec. 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[30] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1, pp. 129–154, 2005, Publisher: Springer.

[31] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[32] D. Griffin, I. Bate, and R. I. Davis, *Dgdguk/drs*, version v2.0.0. DOI: 10.5281/zenodo.4264857. [Online]. Available: https://github.com/dgdguk/drs.

[33] D. Griffin, I. Bate, and R. I. Davis, "Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests," in *IEEE Real-Time Systems Symposium, RTSS 2020, Houston, Texas, USA*.