Schedulability using native non-preemptive groups on an AUTOSAR/OSEK platform with caches

Leo Hatvani[a], Reinder J. Bril[a,b], Sebastian Altmeyer[c]

[a]Technische Universiteit Eindhoven (TU/e), The Netherlands, [b]Mälardalen University (MDH), Västerås, Sweden
[c]University of Amsterdam (UvA), The Netherlands

***Abstract*—Fixed-priority preemption threshold scheduling (FPTS) is a limited preemptive scheduling scheme that generalizes both fixed-priority preemptive scheduling (FPPS) and fixed-priority non-preemptive scheduling (FPNS). By increasing the priority of tasks as they start executing it reduces the set of tasks that can preempt any given task.**

A subset of FPTS task configurations can be implemented natively on any AUTOSAR/OSEK compatible platform by utilizing the platform's native implementation of non-preemptive task groups via so called internal resources. The limiting factor for this implementation is the number of internal resources that can be associated with any individual task. OSEK and consequently AUTOSAR limit this number to one internal resource per task.

In this work, we investigate the impact of this limitation on the schedulability of task sets when cache related preemption delays are taken into account. We also consider the impact of this restriction on the stack size when the tasks are executed on a shared-stack system.

I. Introduction

In modern embedded real-time systems, one of the main limitations is the production cost. To reduce these costs, embedded systems manufacturers often utilize commercial off-the-shelf (COTS) programmable platforms. In such platforms, a cache often bridges the speed gap between the processor and the slower main memory. This cache can give rise to additional cache-related preemption delays (CRPD) that increase the worst-case response times (WCRT) of tasks in a given task set and thus may render one or more of the tasks in a given task set unschedulable.

In this work we focus on the AUTOSAR/OSEK[1] standard for real-time embedded vehicular platforms. Among other features, the standard defines a way to group normally preemptable tasks into non-preemptive groups. That is, groups of tasks that will not preempt any other task from the same group, and may be preempted by tasks of a higher priority from another group.

By creating groups of non-preemptive tasks, a native implementation [1] of Fixed Priority preemption Threshold Scheduling algorithm (FPTS) [2, 3, 4, 5] can be constructed. As FPTS can be specialized to fully preemptive or fully non-preemptive scheduling, and it can schedule some task sets that are not schedulable by either approach, it outperforms them in terms of the ability to schedule tasks. General performance of FPTS in the presence of caches has already been analyzed [6]. However, in this work we aim to compare the performance of FPTS to the AUTOSAR restricted variant of FPTS, in the presence of caches, as well as the impact on the stack size for shared-stack systems.

A. AUTOSAR and FPTS

When scheduling tasks using FPTS on an AUTOSAR/OSEK compatible platform it is possible to implement a subset of FPTS configurations using native non-preemptive groups that are realized by means of *internal resources* [1].

Internal resources are a special type of resources that are assigned to tasks at design time and, during runtime, locked and unlocked by the scheduler at the beginning and at the end of the execution of a task instance [7]. The tasks do not interact directly with internal resources. When two or more tasks are assigned the same resource, they cannot preempt each other and thus form a non-preemptive group.

However, the standard describes a limitation that at most one internal resource can be allocated to a task [7]. We call this restriction One Internal Resource per task (OneIR) constraint [1]. A consequence of the OneIR constraint is that only a subset of schedulable FPTS configurations can be implemented on a strictly AUTOSAR/OSEK compliant platform [1]. We refer to this specialized variant of FPTS as FPTS with One Internal Resource per task (FPTS-OneIR).

In our previous work, less than 2% of the examined task sets that are schedulable under FPTS become unschedulable under FPTS-OneIR without caches [1] and the average relative increase in stack size does not exceed 20% [8].

It is worth noting that OneIR constraint has been lifted in some AUTOSAR compliant operating systems, such as ETAS RTA-OSEK and RTA-OS [9].

B. Schedulability

As the main focus of this paper, we investigate the impact of the OneIR constraint on the potential schedulability of a task set when scheduled on an AUTOSAR/OSEK compliant platform with CRPD taken into account. Since we consider hard real-time systems only, if any task in a task set has a response time larger than its deadline, the entire task set is deemed unschedulable.

C. Preemption depth

On a shared-stack system, tasks can occupy the same locations on the stack if they are mutually non-preemptive

This work is supported by the ARTEMIS Joint Undertaking project EMC^2 (grant agreement 621429).

[1]AUTOSAR/OSEK standard can be found at http://www.autosar.org/

τ_i	θ_i
τ_1	π_1
τ_2	π_2
τ_3	π_2
τ_4	π_2
τ_5	π_4

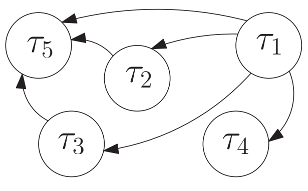

Fig. 1. An example of a potential preemption graph with a depth of 3.

[10, 3], thus reducing the total required stack size. This feature is available in several AUTOSAR compliant operating systems, e.g. Unicoi Fusion[2], Erika Enterprise[3], and ETAS RTA-OSEK[4].

For the unconstrained FPTS, the minimum stack size is always achieved by finding the maximum schedulable preemption thresholds [11] (a configuration of preemption threshold for which any increase would make the task set unschedulable). For FPTS-OneIR, there can be multiple such configurations [8] and only some of them provide the minimum stack size. Since an algorithm for generating a minimum stack size preemption threshold configuration for FPTS-OneIR is as of yet unknown, we explore all schedulable maximum preemption threshold configurations and determine the ones with the smallest stack size.

In order to reduce the effects of selecting the stack size for the individual tasks and still be able to draw conclusions about the impact of the OneIR constraint on the stack size, we use the concept of *preemption depth*. For any FPTS or FPTS-OneIR scheduled set of tasks, it is possible to generate a potential preemption graph [12]. If the nodes are the tasks and the directed edges represent potential preemptions, the graph represents any preemptions that can happen in such system based on the information about task priorities and preemption thresholds. As an example, let $\pi_1, \ldots, \pi_5$ be the priorities of tasks $\tau_1, \ldots, \tau_5$, respectively, with π_1 being the highest priority and π_5 the lowest priority. Figure 1 shows a sample configuration of preemption thresholds θ_i and the resulting preemption graph. Based on that graph, we can compute the maximum preemption depth of the task set that is equivalent to the number of nodes on the longest path in the graph, i.e. 3.

The algorithm for efficient computation of the maximum preemption depth is equivalent to the algorithm for determining the minimum stack size [8, 13] with the stack sizes set to 1. In the rest of this paper, we will refer to the maximum preemption depth as the preemption depth, and to this algorithm as PreemptionDepth.

D. Overview

The rest of this document is structured as follows. After introducing our models and notation in Section II, we provide a recap of the algorithm for determining optimal preemption thresholds for FPTS with CRPD in Section III. Then we provide our algorithm for determining preemption thresholds for FPTS with CRPD in Section IV and use it to compare FPTS to FPTS-OneIR in Section V. Finally, we outline our major conclusions in Section VI.

[2]Details about Unicoi Fusion can be found at http://www.unicoi.com/.

[3]Details about Erika Enterprise OS can be found at http://erika.tuxfamily.org/.

[4]Details about ETAS RTA-OSEK can be found at http://www.etas.com/.

II. Models and notation

This section presents the models and notation that we use throughout this paper. We start with a basic, continuous scheduling model for FPPS, i.e., we assume time to be taken from the real domain ($\mathbb{R}$), similar to, e.g., [14, 15, 16]. We subsequently refine this basic model for FPTS [2], and FPTS-OneIR [1]. Next, we introduce a basic memory model and a model for cache-related pre-emption costs.

A. Basic model for FPPS

We assume a single processor and a set $\mathcal{T}$ of n independent sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$, with unique priorities $\pi_1, \pi_2, \ldots, \pi_n$. At any moment in time, the processor is used to execute the highest priority task that has work pending. For notational convenience, we assume that (*i*) tasks are given in order of decreasing priorities, i.e. τ_1 has the highest and τ_n the lowest priority, and (*ii*) a higher priority is represented by a lower value, i.e. $\pi_1 < \pi_2 < \ldots < \pi_n$.

Each task τ_i is characterized by a *minimum inter-activation time* $T_i \in \mathbb{R}^+$, a *worst-case computation time* $C_i \in \mathbb{R}^+$, and a (*relative*) *deadline* $D_i \in \mathbb{R}^+$. We assume that the constant pre-emption costs, such as context switches, are subsumed into the worst-case computation times. We feature arbitrary deadlines, i.e. the deadline D_i may be smaller than, equal to, or larger than the period T_i. The *utilization* U_i of task τ_i is given by C_i/T_i, and the *utilization* U of the set of tasks $\mathcal{T}$ by $\sum_{1\leq i\leq n} U_i$. An activation of a task is also termed a *job*. The first job arrives at an arbitrary time.

We also adopt standard basic assumptions [17], i.e. tasks do not suspend themselves and a job of a task does not start before its previous job is completed.

B. Refined model for FPTS and FPTS-OneIR

In FPTS, each task τ_i has a *pre-emption threshold* θ_i, where $\pi_1 \leq \theta_i \leq \pi_i$. When τ_i is executing, it can only be pre-empted by tasks with a priority higher than θ_i. Note that we have FPPS and FPNS as special cases when $\forall_{1\leq i\leq n}\theta_i = \pi_i$ and $\forall_{1\leq i\leq n}\theta_i = \pi_1$, respectively.

In our previous work [1], we show that any FPTS schedulable task set that satisfies the following expression can be implemented using at most one AUTOSAR internal resource per task.

$$\nexists 1 < i < j \leq n : (\theta_j = \pi_i) \wedge (\theta_i \neq \pi_i) \tag{1}$$

Note that the example in Figure 1 does not conform to this constraint since task τ_5 has preemption threshold $\theta_5 = \pi_4$ and task τ_4 has threshold $\theta_4 = \pi_2$.

There are two alternative approaches to make the example task set OneIR compatible: (a) change the preemption threshold θ_4 to π_4, or (b) change the preemption threshold θ_5 to π_1, π_2, or π_5 (any value for which $\pi_i = \theta_i$). If the original task set was a maximum threshold configuration, threshold θ_5 can be only lowered to π_5. Thus, starting from an FPTS maximum threshold configuration, an FPTS-OneIR configuration can be derived by iterating over tasks from the highest priority, branching every time a task is detected that invalidates Equation 1, and introducing corrections in each of the branches [8].

C. A memory model

We consider two types of memory, (main) memory and (instruction) cache (memory). Memory and cache are assumed to contain (memory) blocks of a fixed size, where memory contains N^{M} blocks and cache N^{C} blocks, and typically $N^{\mathrm{M}} \gg N^{\mathrm{C}}$. Memory blocks and cache blocks are numbered from 0 until $N^{\mathrm{M}} - 1$ and from 0 to $N^{\mathrm{C}} - 1$, respectively. Similar to [6], we assume direct-mapped caches [18], i.e. a memory block is mapped to exactly one cache block. The worst-case block-reload time (BRT) is assumed to be a constant that upper bounds the time to load a block from main memory to cache.

The set of memory blocks of task τ_i is denoted by MB_i. This set contains natural numbers and each number refers to a certain memory block.

The *cache utilization* of a task τ_i is given by $U_i^{\mathrm{C}} = |\mathrm{MB}_i|/N^{\mathrm{C}}$, where $|\mathrm{MB}_i|$ denotes the cardinality of the set MB_i. The cache utilization of an individual task can therefore be larger than one, i.e. when $|\mathrm{MB}_i| > N^{\mathrm{C}}$. The *cache utilization* U^{C} of the set of tasks $\mathcal{T}$ is given by $U^{\mathrm{C}} = \sum_{1 \le i \le n} U_i^{\mathrm{C}}$.

The set of cache blocks of task τ_i is determined by MB_i and the surjective memory to cache mapping function $MapM2C : \{0, \ldots, N^{\mathrm{M}} - 1\} \rightarrow \{0, \ldots, N^{\mathrm{C}} - 1\}$.

D. A model for cache-related pre-emption costs

Similar to [19], we also use the concepts of *evicting cache blocks* (ECBs) and *useful cache blocks* (UCBs) in order to analyze CRPDs. The ECBs of a task τ_i are denoted by the set ECB_i; the UCBs of a task τ_i are denoted by the set UCB_i. Just like MB_i, these sets are also represented as sets of natural numbers. By definition, the set UCB_i is a subset of the set ECB_i, i.e. $\mathrm{UCB}_i \subseteq \mathrm{ECB}_i$. The set ECB_i is determined as the union of all MB_i mapped onto the cache using memory to cache mapping function $MapM2C$. The relation between the ECBs of a task (ECB_i), the UCBs of a task (UCB_i) and the BRT is demonstrated in Example 1, below.

Example 1: We assume a direct-mapped cache with 4 cache blocks and two tasks τ_1 and τ_2. The memory blocks of τ_1 map to cache blocks 0, 1 and 2. Only τ_1's memory block mapping to cache block 1 is useful, i.e. $\mathrm{ECB}_1 = \{0, 1, 2\}$ and $\mathrm{UCB}_1 = \{1\}$. The memory blocks of τ_2 map to cache blocks 1, 2, and 3 and all three are useful, i.e. $\mathrm{ECB}_2 = \{1, 2, 3\}$ and $\mathrm{UCB}_2 = \{1, 2, 3\}$. The cache-related pre-emption cost of task τ_1 pre-empting task τ_2 is thus given as follows:

$$|\mathrm{ECB}_1 \cap \mathrm{UCB}_2| \cdot \mathrm{BRT} = |\{1, 2\}| \cdot \mathrm{BRT} = 2 \cdot \mathrm{BRT}.$$

III. Recap of optimal threshold assignment algorithm

In this section, we first introduce the relevant FPTS response time analysis concepts and then review the algorithm for determining the maximum preemption thresholds for FPTS when CRPDs are taken into account, originally published by Bril et. al [6].

Given a set of tasks scheduled using FPTS, a task's worst-case response time may be increased by higher and lower priority tasks [2, 5]. Higher priority tasks may preempt the task, given that their priority is higher than the task's preemption threshold. At the same time, a lower priority task may delay the start of execution of a task given the right preemption thresholds. We call delays introduced by higher priority tasks *interference*, and the ones introduced by lower priority tasks *blocking*. A single job of a task can experience interference from multiple jobs of higher priority tasks, but it can be blocked by only one lower priority job. Therefore, for the purposes of presenting the algorithms, let ComputeR($\mathcal{T}, \tau_a, \tau_b$) be a function that returns the worst-case response time of a task τ_a in the context of the task set $\mathcal{T}$ with blocking by the task τ_b due to its preemption threshold θ_b.

Algorithm 1 OptimalThresholdAssignment [6]

```
Input: A set of n tasks $\mathcal{T}$, and $\{C_i, T_i, D_i, \pi_i\}$ $\forall \tau_i \in \mathcal{T}$.
Output: The schedulability of the task set and $\theta_i, \forall \tau_i \in \mathcal{T}$.
 1: function OptimalThresholdAssignment($\mathcal{T}$)
 2:     for all ($\tau_i \in \mathcal{T}$) do
 3:         $\hat{\theta}_i \leftarrow \pi_1$                    ▷ Initial max. thresholds.
 4:         $\theta_i \leftarrow \pi_i$                          ▷ Initial thresholds.
 5:     end for
 6:     for ($\tau_i : \tau_1$ to $\tau_n$) do
 7:         $\theta_i \leftarrow \hat{\theta}_i$          ▷ Assign max. as current threshold.
    ▷ Compute worst-case response time without blocking.
 8:         $R_i \leftarrow$ ComputeR($\mathcal{T}, \tau_i, \emptyset$)
 9:         if ($R_i > D_i$) then return unschedulable end if
    ▷ Adjust max. threshold of lower priority task $\tau_{i+1}, \ldots, \tau_n$.
10:         AdjustMaxThresholds($\mathcal{T}, \hat{\Theta}, i$)
11:     end for
12:     return schedulable
13: end function
```

Algorithm 2 AdjustMaxThresholds

```
Input: A set of n tasks $\mathcal{T}$, and $\{C_i, T_i, D_i, \pi_i, \theta_i\}$ $\forall \tau_i \in \mathcal{T}$, the
    set of the maximum preemption thresholds $\hat{\Theta}$, and the
    index i specifying the range of tasks to be adjusted.
Output: The adjusted set of max. preemption thresholds $\hat{\Theta}$.
 1: procedure AdjustMaxThresholds($\mathcal{T}, \hat{\Theta}, i$)
 2:     for ($\tau_j : \tau_{i+1}$ to $\tau_n$) do
    ▷ Temporary assignment of the preemption threshold.
 3:         $\theta_j \leftarrow \hat{\theta}_j$
    ▷ Compute the response time of task $\tau_i$ blocked by $\tau_j$.
 4:         $R_i \leftarrow$ComputeR($\mathcal{T}, \tau_i, \tau_j$)
    ▷ If task $\tau_i$ cannot tolerate blocking of task $\tau_j$, then lower
    the maximum preemption threshold of $\tau_j$.
 5:         if ($R_i > D_i$) then $\hat{\theta}_j = \pi_{i+1}$ end if
 6:         $\theta_j \leftarrow \pi_j$              ▷ Restore the initial threshold.
 7:     end for
 8: end procedure
```

We present the OptimalThresholdAssignment algorithm [6] in two parts: the main body of the algorithm as Algorithm 1, and the supporting procedure AdjustMaxThresholds as Algorithm 2.

The core idea of OptimalThresholdAssignment is that, given a set of schedulable tasks of a higher priority (e.g. $\{\tau_1, \tau_2\}$) with predefined preemption thresholds (θ_1 and θ_2), a task of a lower priority (τ_3) can have preemption threshold equal to the highest priority task ($\theta_3 = \pi_1$) only if all intermediate tasks, and the higher priority task, can tolerate the blocking of the given task (ComputeR($\mathcal{T}, \tau_1, \tau_3$) $\le D_1$ and ComputeR($\mathcal{T}, \tau_2, \tau_3$) $\le D_2$).

OptimalThresholdAssignment iterates over the task set from the highest priority to the lowest. Besides the current preemption thresholds $\{\theta_1, \theta_2, \ldots\}$, it maintains a second array of preemption thresholds $\hat{\Theta} = \{\hat{\theta}_1, \hat{\theta}_2, \ldots\}$ that tracks the maximum preemption threshold that can be assigned to a task that keeps all of the higher priority tasks schedulable.

During iteration over the task set, it assigns the maximum preemption threshold found so far to each task (Algorithm 1:7), checks whether it is schedulable without any blocking from the lower priority tasks (Algorithm 1:8-10) and then adjusts all of the lower priority tasks so that they do not block the observed task if it will make it unschedulable (Algorithm 1:10 and Algorithm 2).

IV. Preemption thresholds for FPTS-OneIR with CRPD

In this section we present the algorithm for finding schedulable FPTS-OneIR preemption threshold configurations that have the smallest preemption depth while accounting for CRPD.

A. Problem description

When considering FPTS-OneIR with CRPD in comparison to Algorithm 1, there are two aspects that need to be accounted for: (a) any threshold configuration that does not satisfy the OneIR constraint is considered invalid, and (b) similar to searching for the configuration with the smallest stack size [8], there may be multiple threshold configurations that need to be explored to determine the one with the smallest preemption depth.

B. Proposed algorithm

Our algorithm for finding FPTS-OneIR schedulable preemption threshold configurations which accounts for CRPD and has the smallest preemption depth is listed in two parts: (*i*) the initialization and result interpretation as Algorithm 3, and (*ii*) the main recursive body of the algorithm as Algorithm 4.

Algorithm 3 FPTS-OneIR-CRPD

Context: Global variables: *minDepth* representing the smallest discovered preemption depth for a schedulable configuration, and $\vec{\Theta}$ set of schedulable preemption threshold configurations that produce it.
Input: A set $\mathcal{T}$ of n tasks, and $\{C_i, T_i, D_i, \pi_i, \theta_i\}\ \forall \tau_i \in \mathcal{T}$ with θ_i set to maximum preemption thresholds for FPTS.
Output: The schedulability of the task set under FPTS-OneIR, the minimum discovered preemption depth as *minDepth* and the set of FPTS-OneIR schedulable threshold configurations as $\vec{\Theta}$.

```
1:  function FPTS-OneIR-CRPD($\mathcal{T}$)
        ▷ Initialize the global variables.
2:      $minDepth \leftarrow +\infty$
3:      $\vec{\Theta} \leftarrow \emptyset$
        ▷ Start the recursion.
4:      Recurse($\mathcal{T}$, 1)
        ▷ Interpret the results.
5:      if ($|\vec{\Theta}| > 0$) then
6:          return SCHEDULABLE
7:      else
8:          return UNSCHEDULABLE
9:      end if
10: end function
```

Algorithm 4 Recurse

Context: Global variables: *minDepth*, and $\vec{\Theta}$ as in Algorithm 3.
Input: A set $\mathcal{T}$ of n tasks, and $\{C_i, T_i, D_i, \pi_i, \theta_i\}\ \forall \tau_i \in \mathcal{T}$. An index p such that all tasks τ_i with a priority higher than π_p do not invalidate the OneIR constraint and are schedulable at their current thresholds.
Output: Upon completion, the global variables *minDepth*, and $\vec{\Theta}$ are updated if any schedulable configurations with preemption depth smaller or equal to *minDepth* have been discovered.

```
1:  procedure Recurse($\mathcal{T}$, $p$)
2:      if ($p \leq n$) then
3:          if ($\theta_p \neq \pi_p \wedge (\exists \tau_i \in \mathcal{T} : \theta_i = \pi_p)$) then
        ▷ Store the current preemption threshold configuration.
4:              $\Theta_{orig} \leftarrow (\theta_1, \ldots, \theta_n)$
5:              $\theta_p \leftarrow \pi_p$                    ▷ First adjustment.
6:              Recurse($\mathcal{T}$, $p$)
        ▷ Restore the original preemption threshold configuration.
7:              $(\theta_1, \ldots, \theta_n) \leftarrow \Theta_{orig}$
8:              for all ($\tau_i \in \mathcal{T} : \theta_i = \pi_p$) do      ▷ Second adj.
9:                  $\theta_i \leftarrow \pi_{p+1}$
10:             end for
11:             Recurse($\mathcal{T}$, $p$)
12:         else
        ▷ The task $\tau_p$ is not involved in OneIR constraint violation.
13:             $R_p \leftarrow$ ComputeR($\mathcal{T}$, $\tau_p$, $\emptyset$)
14:             if ($R_p \leq D_p$) then
15:                 for all ($\tau_i : \pi_i > \pi_p \wedge \theta_i \leq \pi_p$) do
16:                     if (ComputeR($\mathcal{T}$, $\tau_p$, $\tau_i$)$> D_p$) then
17:                         $\theta_i \leftarrow \pi_{p+1}$
18:                     end if
19:                 end for
20:                 Recurse($\mathcal{T}$, $p + 1$)
21:             end if
22:         end if
23:     else
        ▷ The task set is schedulable and satisfies OneIR constraint.
24:         if (PreemptionDepth($\mathcal{T}$)$= minDepth$) then
25:             $\vec{\Theta} \leftarrow \vec{\Theta} \cup \{\theta_1, \ldots, \theta_n\}$
26:         else if (PreemptionDepth($\mathcal{T}$)$< minDepth$) then
27:             $minDepth \leftarrow$ PreemptionDepth($\mathcal{T}$)
28:             $\vec{\Theta} \leftarrow \{\theta_1, \ldots, \theta_n\}$
29:         end if
30:     end if
31: end procedure
```

The algorithm is based on our algorithm for finding the preemption threshold configuration with the smallest stack size [8] and integrates some elements of Algorithms 1 and 2 from Section III to remove the dependency on the blocking tolerance. The main idea of the algorithm is that, starting from the maximum preemption threshold configuration for FPTS, it progresses from the highest priority task to the lowest and resolves any OneIR constraint violations using two alternative approaches outlined in Section II-B.

The initialization phase of our algorithm is listed as Algorithm 3:2-3. During this phase, we set the two global variables, *minDepth*, and $\vec{\Theta}$ to positive infinity and empty set, respectively. Then we call Algorithm 4 for the task at the highest priority. As Algorithm 4 executes, it compares the newly discovered preemption depth values to *minDepth* and adds the relevant preemption threshold configurations to $\vec{\Theta}$. Once the recursion is completed, if the cardinality of the set $\vec{\Theta}$ is greater than zero, the task set is schedulable.

The recursive procedure, listed as Algorithm 4, starts by detecting whether all of the tasks have been processed. If the index p is smaller or equal to the number of tasks, the procedure checks whether the task τ_p breaks the OneIR constraint (Algorithm 4:3) and then tries two alternative adjustments to correct it.

First, a temporary copy of the preemption threshold configuration is created as Θ_{orig} (Algorithm 4:4). Then, the preemption threshold θ_p of task τ_p is lowered to the priority π_p. This negates the right side of the OneIR constraint conjunction (1), and makes the task τ_p OneIR compatible. Since the task is now OneIR compatible, we can call the recursive procedure for the same index (Algorihtm 4:6) to process the task as OneIR compatible task. The alternative adjustment begins by restoring the preemption threshold configuration from Θ_{orig} (Algorithm 4:7). Then the procedure identifies all tasks τ_i with a preemption threshold equal to π_p and lowers their preemption thresholds to π_{p+1}. Since the left side of the OneIR constraint conjunction (1) is now false, we call the recursive procedure again for the same index (Algorithm 4:11).

If task τ_p does not invalidate the OneIR constraint (Algorithm 4:13), we check if it is schedulable without any lower priority tasks blocking it. If it is not schedulable, the task cannot be made schedulable without altering preemption thresholds of higher priority tasks and the recursion ends. If it is schedulable without blocking, the procedure identifies tasks τ_i that can block it. For each task τ_i, the response time R_p with the blocking from task τ_i is calculated. If it is greater than the deadline D_p, the preemption threshold θ_i is lowered to π_{p+1}.

If line Algorithm 4:20 is reached, the set of tasks $\pi_1, \dots, \pi_p$ is OneIR compatible and schedulable. When the procedure recurses across all tasks from $\mathcal{T}$, the algorithm compares the current preemption depth to the ones already found and adjusts the global variables *minDepth* and $\vec{\Theta}$ (Algorithm 4:24-29).

V. Evaluation

We have evaluated the schedulability and preemption depth of FPTS compared to FPTS-OneIR with and without caches. The evaluation was done by generating a number of synthetic task sets using UUnifast [20] algorithm with a default of 10 tasks, implicit deadlines, and the task periods randomly drawn from the interval [10, 1000]ms. To have comparable results to the previous work [6], we have set the cache size to $N^C = 512$ cache sets, with utilization $U^C = 4$, resulting in $N^C \times U^C = 2048$ ECBs. The individual cache utilizations U_i^C were selected using UUnifast with 40% of ECBs as UCBs. For each task set and the algorithm, we applied both ECB-Union Multiset and UCB-Union Multiset [6, 19] approach to get the largest possible schedulability for all algorithms.

For Figure 2, we have compared 1000 task sets of 10 tasks for every utilization from 0.6 to 0.975 in 0.025 increments. From the graph we can see that FPTS and FPTS-OneIR almost overlap when they are compared without CRPD influence,

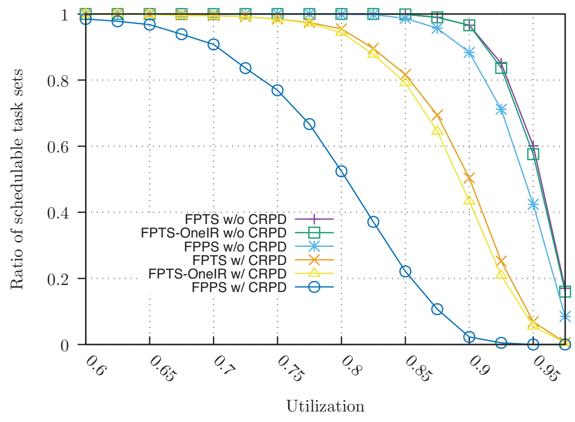

Fig. 2. Ratio of schedulable task sets for varying utilizations. Larger schedulability ratio is better.

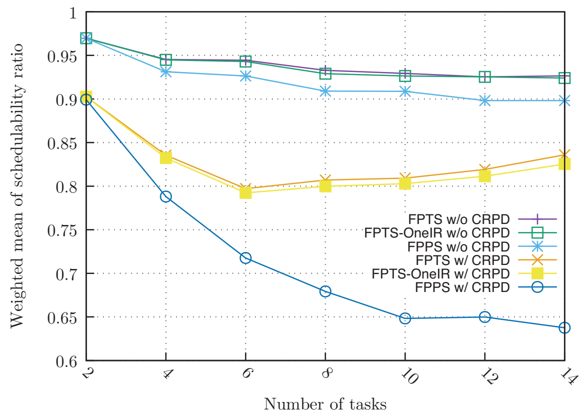

Fig. 3. Weighted mean of schedulability ratio for varying number of tasks. Larger schedulability ratio is better.

which corresponds to the earlier schedulability results [1]. However, the difference becomes slightly more pronounced with CRPD. The largest mean difference of FPTS and FPTS-OneIR in the presence of caches can be found for the utilization 0.9 and is 7%.

The trend of increasing difference in schedulability is best observed in Figure 3. To generate the graph, we have produced 100 task sets for every utilization in the range [0.025, 0.975] in 0.025 increments, resulting in 3900 task sets per data point. The experiment results were then aggregated using weighted means based on the formula $(\sum_{i=1}^{E} x_i U_i)/(\sum_{i=1}^{E} U_i)$ [21], where U_i are the utilizations, E total number of experiments, and x_i schedulability results encoded as 0 or 1.

For preemption depths in Figures 4 and 5, we have used the same generation approaches as for their utilization counterparts in Figures 2 and 3 respectively. For each task set, we computed a relative increase in preemption depth as $\frac{x-y}{y}$ comparing FPTS to FPTS-OneIR with and without caches, as well as the performance of each scheduling approach to itself when caches are added. From both figures, we can see that adding CRPD to FPTS or FPTS-OneIR has only minor influence on the preemption depth.

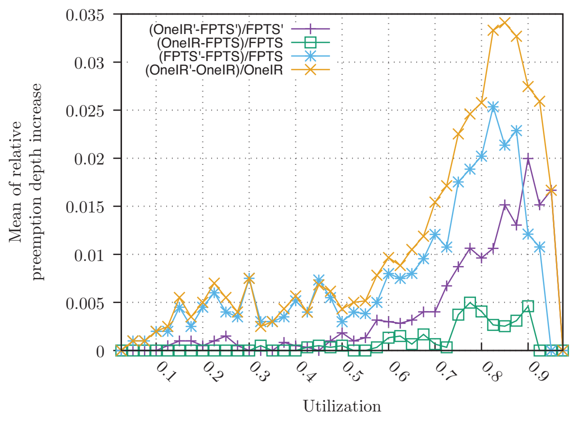

Fig. 4. Mean values of relative increase in preemption depth for varying utilizations. FPTS and OneIR correspond to the results without CRPD, and FPTS' and OneIR' correspond to the results with CRPD. Smaller increase in preemption depth is better.

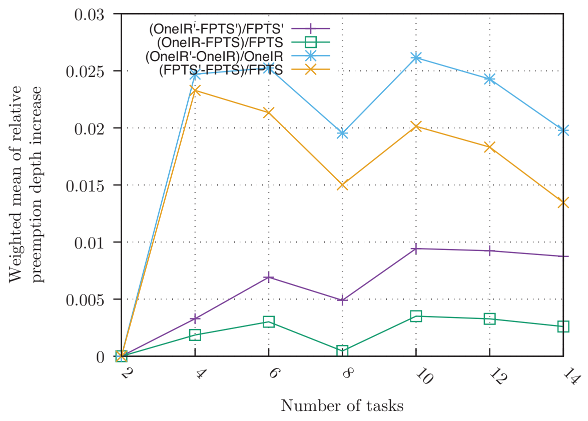

Fig. 5. Weighted mean values of relative increase in preemption depth for varying number of tasks. FPTS and OneIR correspond to the results without CRPD, and FPTS' and OneIR' correspond to the results with CRPD. Smaller increase in preemption depth is better.

VI. Conclusions

In this paper we have analyzed the impact of the cache related preemption delays on the schedulability and minimum stack size on FPTS compared to FPTS-OneIR, an AUTOSAR/OSEK restricted native implementation of FPTS.

To achieve these results, we have first defined the preemption depth as an abstract measure of the stack size which is independent of the generated stack sizes used in earlier work. Next, we presented a new algorithm for FPTS-OneIR preemption threshold assignment that is compatible with the restrictions of the CRPD analysis and that finds a solution with the smallest preemption depth.

Even though the difference in schedulability between FPTS and FPTS-OneIR increases when CRPD are taken into account, both approaches still significantly outperform FPPS. Within our observed task sets, FPTS outperformed FPTS-OneIR by at most 7% for 90% system utilization, 10 tasks, and implicit deadlines. It is worth noting that the gap increases with larger number of tasks and it is an open question if this trend continues. Evaluation for arbitrary deadlines is part of our future work. For preemption depth, we have observed that the largest mean difference that we have detected is less than 4%.

Finally, we conclude that even in contexts where CRPDs need to be included in the response time analysis, FPTS-OneIR is a suitable alternative to FPTS.

References

[1] L. Hatvani and R. J. Bril, "Schedulability using native non-preemptive groups on an AUTOSAR/OSEK platform," in *Proc. 20th Conf. on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–8.

[2] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Proc. 6th Int. Conf. on Real-Time Computing Systems and Applications (RTCSA)*, Dec. 1999, pp. 328–335.

[3] M. Saksena and Y. Wang, "Scalable real-time system design using preemption thresholds," in *Proc. 21st IEEE Real-Time Systems Symposium (RTSS)*, Nov. 2000, pp. 25–34.

[4] J. Regehr, "Scheduling tasks with mixed preemption relations for robustness to timing faults," in *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2002, pp. 315–326.

[5] U. Keskin, R. J. Bril, and J. J. Lukkien, "Exact response-time analysis for fixed-priority preemption-threshold scheduling," in *IEEE Conf. on Emerging Tech. and Factory Automation (ETFA)*, Sep. 2010, pp. 1–4.

[6] R. J. Bril, S. Altmeyer, M. M. H. P. v. d. Heuvel, R. I. Davis, and M. Behnam, "Integrating cache-related pre-emption delays into analysis of fixed priority scheduling with pre-emption thresholds," in *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2014, pp. 161–172.

[7] OSEK group, *OSEK/VDX operating system, Version 2.2.3*, February 2005. [Online]. Available: http://portal.osek-vdx.org/files/pdf/specs/os223.pdf

[8] L. Hatvani and R. J. Bril, "Minimizing stack usage for AUTOSAR/OSEK's restricted fixed-priority preemption threshold support," in *Proc. 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[9] D. Buttle, "Real-time in the prime-time," Keynote speech given at the 24th Euromicro Conf. on Real-Time Systems (ECRTS), July 2012, Presentation available from http://ecrts.eit.uni-kl.de/index.php?id=69.

[10] R. Davis, N. Merriam, and N. Tracey, "How embedded applications using an RTOS can stay within on-chip memory limits," in *Work in Progress and Industrial Experience Session, 12th Euromicro Conference on Real-Time Systems (ECRTS)*, Jun. 2000.

[11] R. Ghattas and A. G. Dean, "Preemption threshold scheduling: Stack optimality, enhancements and analysis," in *Proc. 13th Real Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2007, pp. 147–157.

[12] M. Bohlin, K. Hänninen, J. Mäki-Turja, J. Carlson, and M. Nolin, "Bounding shared-stack usage in systems with offsets and precedences," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, Jul. 2008, pp. 276–285.

[13] G. Yao and G. Buttazzo, "Reducing stack with intra-task threshold priorities in real-time systems," in *Proc. 10th ACM International Conference on Embedded Software (EMSOFT)*, Oct. 2010, pp. 109–118.

[14] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, Nov. 1990.

[15] R. Bril, J. Lukkien, and W. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, no. 1-3, pp. 63–119, Aug. 2009.

[16] M. Bertogna, G. Buttazzo, and G. Yao, "Improving feasibility of fixed priority tasks using non-preemptive regions," in *Proc. 32nd IEEE Real-Time Systems Symposium (RTSS)*, Nov. 2011, pp. 251–260.

[17] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[18] D. Patterson and J. Hennessy, *Computer organization and design (5th edition)*. Morgan Kaufman, 2014.

[19] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, Sep. 2012.

[20] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[21] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *OSPERT*, pp. 33–44, Jul. 2010.