Mixed criticality systems with varying context switch costs

Robert I. Davis, Sebastian Altmeyer, Alan Burns

Angaben zur Veröffentlichung / Publication details:

Davis, Robert I., Sebastian Altmeyer, and Alan Burns. 2018. "Mixed criticality systems with varying context switch costs." In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 11-13 April 2018, Porto, Portugal*, edited by Rodolfo Pellizzoni, 140–51. Piscataway, NJ: IEEE. https://doi.org/10.1109/rtas.2018.00024.

Mixed Criticality Systems with Varying Context Switch Costs

Robert I. Davis[1], Sebastian Altmeyer[2], and Alan Burns[1]

[1]University of York, York, UK
[2]University of Amsterdam (UvA), Amsterdam, The Netherlands

***Abstract*—In mixed criticality systems, it is vital to ensure that there is sufficient separation between tasks of LO- and HI-criticality applications, so that the behavior or mis-behavior of the former cannot affect the functional or timing correctness of the latter. To ensure appropriate spatial isolation, the memory address spaces and cache use by LO- and HI-criticality tasks must be distinct. A consequence of this separation is that the cost of switching between tasks of the same criticality can be small, whereas the cost of context switching between tasks of different criticality levels can be much larger. In this paper, we focus on integrating the differing context switch costs into fixed priority preemptive scheduling, and the two mixed criticality scheduling schemes based on it: SMC and AMC. We derive simple, refined, and multi-set analyses for each scheme. Further, we show that the refined and multi-set analyses are not compatible with Audsley's Optimal Priority Assignment algorithm, we therefore propose a heuristic priority assignment policy aimed at reducing the number of high cost context switches. Our evaluation is grounded in measurements of context switch times (save and restore costs) from a prototype implementation of an explicitly managed cache on an FPGA. The evaluation shows the effectiveness of the derived analyses and the proposed priority assignment policy.**

I. INTRODUCTION

An important trend in the design of real-time systems is the integration of applications with different levels of criticality onto the same hardware platform. Here, *criticality* is the term used to described the level of assurance against failure needed by each application. A Mixed Criticality System (MCS) is one that comprises a set of applications with two or more criticality levels. Most of the complex embedded real-time systems found in the automotive and avionics industries are evolving into MCS in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation, and power consumption. The fundamental research question underlying MCS is how to reconcile the conflicting requirements of *separation* for assurance and *sharing* for efficient resource usage. This question gives rise to theoretical problems in modeling and verification, and systems problems relating to the design and implementation of the hardware and run-time software.

Since the seminal work of Vestal in 2007 [49] a standard model has emerged (described in Section II), along with a substantial thread of research on analysis of MCS assuming fixed priority preemptive scheduling schemes [9], [10], [12], [14], [21], [27], [30], [45], [57], see the survey on MCS [22] for further details. All of these papers focus on the resource sharing aspect of the mixed criticality scheduling problem.

Separation is however also vitally important. The relevant safety standards (IEC61508, DO-178C, ISO26262) require that either all applications are developed to the standard required for the highest criticality application, or that independence between different applications is achieved and demonstrated in both spatial and temporal domains. In the temporal domain, this can be ensured via appropriate RTOS mechanisms, for example by aborting a LO-criticality task if it has not yet completed when its execution time budget expires. In the spatial domain, the memory address space(s) used by tasks from HI-criticality applications must be inaccessible to tasks of LO-criticality applications, and hence separate from their address spaces. This requires the use of hardware memory protection and memory mapping / virtual address spaces.

One approach to ensuring spatial isolation is to make use of the concepts of *processes* and *threads*, where each process has a separate memory address space. Using a single process for all HI-criticality applications provides a single memory address space, easing the costs of communication and interaction between HI-criticality tasks, which are implemented as threads within the process. Alternatively, individual applications may each be implemented as a distinct process, providing spatial isolation between applications of the same criticality. LO-criticality applications and their tasks can similarly be mapped to processes and threads. For the purposes of spatial isolation, it is a requirement that applications of different criticality levels cannot be mapped to the same process; they must use disparate address spaces.

The use of processes and threads gives rise to varying context switch costs [54]. Switching threads within a process (i.e. the context switch between tasks of the same application) has a low cost, since this involves switching only the resources unique to the threads e.g. the processor state (program counter, stack pointer, processor registers etc.) which is typically very fast and may be assisted by hardware support. By contrast, switching between processes (i.e. the context switch between tasks of different criticality levels) may have a much higher cost. It involves switching the resources related to the processes. In particular, this involves switching the memory address space, and can also involve operations on the cache, making process switches a much more costly operation.

As an exemplar, we assume that there is a requirement to isolate the cache usage of different processes from one another. (Note, depending on the architecture, the cache contents may in any case not be valid when switching between address spaces). Following the work of Whitham et al. [52], we consider explicit cache management, with hardware support for saving and restoring the cache contents on process-level context switches. We further assume that tasks which belong to the same process are allocated different partitions within the cache. Together, this ensures that when switching between tasks, the cache contents of any preempted task are unchanged when it resumes execution. The main advantage of the this approach is that the only impact that a task (e.g. of LO-criticality) from one process can have on the timing behavior of a task (e.g. of HI-criticality) from another process is via interference due to its execution time, which is strictly bounded via budget enforcement by the RTOS, and a fixed context switch cost. It cannot slow subsequent execution of the preempted task by evicting its useful cache blocks, thus causing Cache Related Preemption Delays (CRPD). Further, such separation helps avoid security hazards, since a task belonging to a compromised low security process cannot use the cache contents as a side channel to obtain information about the behavior of a task from a high security process which it has preempted or executes after [8], [40], [41].

In this paper, we derive schedulability analysis for three different fixed priority scheduling schemes for mixed criticality systems, accounting for the differing context switch costs incurred when switching between tasks. The task model is set out in detail in Section II. The three schemes considered are Fixed Priority Preemptive Scheduling (FPPS) – Section III, Static Mixed Criticality (SMC) scheduling [10] – Section IV, and Adaptive Mixed Criticality (AMC) scheduling [9] – Section V. In each case, we derive three forms of analysis: simple, refined, and multi-set. Section VI discusses the dominance relations between the scheduling schemes and their analyses. We show that the refined and multi-set analyses for task models with differing context switch costs are not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [6], [7]. We therefore propose, in Section VII, a heuristic priority assignment technique aimed at improving schedulability by reducing the number of large cost context switches. In Section VIII, we recap on the explicit cache management approach proposed by Whitham et al. [52] and provide representative context switch costs covering the time taken to save and restore data and instruction cache contents on prototype hardware. The evaluation, in Section IX shows the effectiveness of both the analyses and the priority assignment technique. In order to make a systematic appraisal of analysis performance, we used synthetic task sets; however, to ground the evaluation, we used representative process and thread level context switch times taking into account the cache save and restore costs given in Section VIII. Finally, Section X discusses related work, and Section XI concludes with a summary and directions for future research.

II. Task model, Terminology, and Notation

In this paper, we are interested in applications executing under Fixed Priority Preemptive Scheduling (FPPS) schemes on a single processor. The applications are together assumed to comprise a static set of n tasks $(\tau_1, \tau_j, \ldots, \tau_n)$, each assigned a unique fixed priority. We use the notation $\text{hp}(i)$ (and $\text{lp}(i)$) to mean the set of tasks with priorities higher than (lower than) that of task τ_i. Similarly, we use the notation $\text{hep}(i)$ (and $\text{lep}(i)$) to mean the set of tasks with priorities higher than or equal to (lower than or equal to) that of τ_i. Further we use $\text{aff}(i, j)$ to denote the set of *affected* tasks that can execute between the release and completion of task τ_i and also be preempted by higher priority task τ_j, thus $\text{aff}(i, j) = \text{hep}(i) \cap \text{lp}(j)$.

Jobs of a task may arrive either periodically at fixed intervals of time, or sporadically after some minimum inter-arrival time has elapsed. Each task, is characterized by its relative deadline D_i, worst-case execution time C_i, and minimum inter-arrival time or period T_i. Tasks are assumed to have constrained deadlines, i.e. $D_i \leq T_i$. It is assumed that once a task starts to execute it will never voluntarily suspend itself. The processor utilization U_i of task τ_i is given by C_i/T_i. The total utilization U of a task set is the sum of the individual task utilizations. The worst-case response time R_i of a task τ_i, is the longest time from one of its jobs becoming ready to execute to that job completing execution. A task is referred to as schedulable if its worst-case response time is less than or equal to its deadline $(R_i \leq D_i)$. A task set is referred to as schedulable if all of its tasks are schedulable. We use $E_j(R_i)$ to denote the maximum number of times that a task τ_j can execute (i.e. preempt) during the response time R_i of some lower priority task τ_i.

Each task τ_i is categorized into one of two classes according to its criticality L_i which may be either HI or LO. Further, each task is assumed to belong to an application mapped to a specific process and hence address space. A_i indicates the address space that task τ_i is mapped to. If tasks τ_i and τ_j belong to the same process and hence use the same address space, then $A_i = A_j$, otherwise $A_i \neq A_j$. We assume that tasks of different criticality levels will not normally be mapped to the same process and so share the same address space; however, we do not preclude it. The analysis we derive covers the most general case where the process and address space of a task is arbitrary, independent of its criticality level.

We assume that a context switch from one task τ_i to another τ_j has a large cost C^C if it involves switching process and hence a *change* in the address space (i.e. when $A_i \neq A_j$). In contrast, if there is only a switch between threads of the same process (i.e. when $A_i = A_j$), then the context switch cost C^S is small $(C^S < C^C)$. Note that the context switch costs C^C and C^S include the time spent switching to the preempting task and later back to the preempted task.

We consider the standard mixed-criticality task model, where LO-criticality tasks have a single WCET estimate $C_i(LO)$, while HI-criticality tasks have two estimates $C_i(LO)$ and $C_i(HI)$ with $C_i(HI) \geq C_i(LO)$. Here, $C_i(HI)$ is obtained using conservative timing analysis methods appropriate for guaranteeing the timing behavior of the HI-criticality tasks under all conditions, for example as required by a certification authority. By contrast, $C_i(LO)$ is the WCET estimate used by the system designer, which is assumed to be sufficient to ensure the correct behavior of the system during normal operating conditions.

We note that while the various safety standards specify up to five different criticality levels, in practice many systems comprise applications of just two different criticality levels and so conform to the dual-criticality model assumed in this paper.

III. Schedulability Analysis for FPPS

In this section we derive schedulability analysis for the task model set out in Section II, assuming FPPS and varying context switch times. This analysis assumes a single WCET estimate C_i per task; in the case of MCS, for LO-criticality tasks we may substitute $C_i(LO)$ and for HI-criticality tasks we may substitute $C_i(HI)$.

A. Simple Analysis

A simple analysis may be obtained by assuming that all context switches have a large cost C^C. Extending standard response time analysis for fixed priority preemptive scheduling [5], [33], gives:

$$R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + C^C) \tag{1}$$

Equation (1) can be solved using fixed point iteration starting with a suitable initial value such as C_i, and ends either on convergence or when the value exceeds D_i in which case the task is unschedulable.

Example: Consider three tasks: $\tau_A = (10, 50, 100, LO, A^L)$, $\tau_B = (10, 100, 200, HI, A^H)$, $\tau_C = (200, 265, 300, LO, A^L)$ with parameters $(C_i, D_i, T_i, L_i, A_i)$. Further, $C^C = 5$ and $C^S = 0$. Assuming the analysis embodied in (1), then Deadline Monotonic Priority Order (DMPO) is optimal [37]. With the tasks in DMPO, then the response time of task τ_C, $R_C = 280$, and so the task set is deemed unschedulable.

B. Refined Analysis

The above analysis is pessimistic in that it assumes that all jobs of all tasks that execute within the priority level-i busy period equating to the response time of task τ_i incur the maximum context switch time. In reality, the context switch time depends on both the preempting task and the preempted task. Taking this information into account, we may re-write (1) as follows. Note we assume that the first job in the busy period always experiences a large context switch time, since the previously running job may be of a different criticality level and hence associated with a different process and address space. (We assume that at a priority below the hard real-time tasks, soft real-time tasks may run in a background process).

$$R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j}) \tag{2}$$

where $\gamma_{i,j}$ is defined as follows:

$$\gamma_{i,j} = \begin{cases} C^C & \text{if } \exists h \in \text{aff}(i,j) | A_h \neq A_j \\ C^S & \text{otherwise} \end{cases} \tag{3}$$

Note $\gamma_{i,j}$ equates to a large context switch time only if there is some task τ_h that can execute during the busy period (i.e. response time) of task τ_i, be preempted by task τ_j and belongs to a different process and address space to τ_j.

Returning to the example task set, if we consider DMPO $\{A, B, C\}$, then we again have $R_C = 280$, since all context switches can take the maximum value. However, assuming priority ordering $\{B, A, C\}$, then we have $R_C = 265$ since the three preemptions by jobs of task τ_A incur a small context switch cost C^S, while the two preemptions by jobs of task τ_B incur a large cost. This example shows that DMPO is not optimal for FPPS with this task model. Further, the refined analysis is not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [6], [7]. The reason for this is that the response time of the task of interest e.g. τ_C can depend on the relative priority ordering of higher priority tasks, breaking Condition 1 in [29] which is necessary condition for the applicability of Audsley's algorithm.

C. Multi-set Analysis

The analysis given by (2) can be pessimistic, as illustrated by the example task set with priority ordering $\{A, B, C\}$. Here, it is assumed that all three jobs of task τ_A can incur large context switch costs; however, in reality that is not the case. Task τ_B only executes twice within the response time of task τ_C and each time it executes, it can only be preempted at most once by a single job of τ_A, since $R_B = 30$.

The following *multi-set* analysis addresses this source of pessimism by taking into account the number of times that tasks of intermediate priorities may be preempted by task τ_j within the response time of task τ_i, thus limiting the number of large context switch costs (C^C) included in the analysis.

$$R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil C_j + \gamma^M_{i,j} \right) \tag{4}$$

Recognizing the fact that task τ_j can preempt each intermediate task τ_k at most $E_j(R_k)E_k(R_i)$ times during the response time of task τ_i, we first form a multi-set $M_{i,j}$ containing $E_j(R_k)E_k(R_i)$ copies of the context switch time for task τ_j preempting each task $\tau_k | k \in \text{aff}(i,j)$. (Recall that $E_j(R_k) = \lceil \frac{R_k}{T_j} \rceil$).

$$M_{i,j} = \bigcup_{k \in \text{aff}(i,j)} \left(\bigcup_{E_j(R_k)E_k(R_i)} \left\{ \begin{array}{ll} C^C & \text{if } A_k \neq A_j \\ C^S & \text{otherwise} \end{array} \right\} \right) \tag{5}$$

From the multi-set, we then obtain an upper bound $\gamma^M_{i,j}$ on the total context switch cost caused by the maximum number, $E_j(R_i)$, of preemptions that can occur due to jobs of task τ_j executing within the response time of τ_i.

$$\gamma^M_{i,j} = \sum_{q=1}^{E_j(R_i)} F(q, M_{i,j}) \tag{6}$$

where $F(q, M_{i,j})$ returns the q-th largest value from the multi-set $M_{i,j}$.

Using the above multi-set analysis (4), then task τ_C in the example task set has a response time of $R_C = 275$ (rather than $R_C = 280$) when the tasks are in priority order $\{A, B, C\}$. This is because although there are 3 possible preemptions by task τ_A, only two of them can cause large context switch costs by preempting task τ_B. This is reflected in the multi-set $M_{C,A}$, which contains the value C^C twice, as $E_A(R_B) = 1$ and $E_B(R_C) = 2$, and the value C^S three times, as $E_A(R_C) = 3$. The largest 3 values are then taken as the overall context switch cost due to preemptions by task τ_A.

IV. Schedulability Analysis for SMC

In this section we derive schedulability analysis for Static Mixed Criticality (SMC) scheduling [10], assuming varying context switch times.

SMC is based on Vestal's original approach [49] using fixed priorities, but extended with run-time monitoring. Thus, if a job of a LO-criticality task τ_i does not complete execution by $C_i(LO)$, then it is aborted. Further, if any HI-criticality task executes for its $C_i(LO)$ WCET estimate without completing execution, then the system enters HI-criticality mode. Under SMC, LO-criticality tasks continue to be released and to execute in HI-criticality mode; however, they are not required to meet their deadlines in that mode. (Note the difference from classical FPPS, which effectively requires that LO-criticality tasks meet their deadlines in HI-criticality mode).

A. Simple Analysis and Refined Analysis

Under SMC, the worst-case response times for all tasks in the LO-criticality mode may be computed using the following fixed point iteration, adapted to account for context switch costs. Note in the simple analysis $\gamma_{i,j} = C^C$, whereas in the refined analysis $\gamma_{i,j}$ is defined by (3).

$$\begin{aligned} R_i(LO) &= C_i(LO) + C^C \\ &+ \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil (C_j(LO) + \gamma_{i,j}) \end{aligned} \tag{7}$$

Similarly, the worst-case response times for all tasks in the HI-criticality mode may be computed using the following fixed point iteration. Note only HI-criticality tasks are required to be schedulable in HI-criticality mode.

$$\begin{aligned} R_i(HI) &= C_i(L_i) + C^C \\ &+ \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil (C_j(L_j) + \gamma_{i,j}) \end{aligned} \tag{8}$$

Since $C_i(HI) \geq C_i(LO)$, it follows that $R_i(HI) \geq R_i(LO)$ and so schedulability of HI-criticality tasks can be checked using only (8), while that of LO-criticality tasks can be checked using only (7).

B. Multi-set Analysis

We now apply the techniques used in the multi-set approach (Section III-C) to SMC. Schedulability of each LO-criticality task τ_i is determined by computing its worst-case response time in LO-criticality mode as follows:

$$\begin{aligned} R_i(LO) &= C_i(LO) + C^C \\ &+ \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) + \gamma^M_{i,j}(LO) \right) \end{aligned} \tag{9}$$

where $\gamma^M_{i,j}(LO)$ is defined according to (6), with all of the response time values used in both (6) and (5) taking their $R(LO)$ values (i.e. $R_i(LO)$ in (6) and $R_k(LO)$ and $R_i(LO)$ in (5)).

Similarly, schedulability of each HI-criticality task τ_i is determined by computing its worst-case response time in HI-criticality mode:

$$\begin{aligned} R_i(HI) &= C_i(L_i) + C^C \\ &+ \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(L_j) + \gamma^M_{i,j}(HI) \right) \end{aligned} \tag{10}$$

where $\gamma^M_{i,j}(HI)$ is defined according to (6), with all of the response time values used in both (6) and (5) taking their $R(HI)$ values. Note this requires that $R(HI)$ values are computed for higher priority LO-criticality tasks. Although such tasks are not required to be schedulable in HI-criticality mode, under SMC they still execute and *crucially* they will typically have longer response times in that mode, which may increase the number of costly preemptions that can occur within the response time of the lower priority HI-criticality task τ_i.

While the response time of a LO-criticality task τ_k in HI-criticality mode can be computed using (10) this can be problematic. Iteration must *not* be stopped when the deadline is reached, since the task may be unschedulable in HI-criticality mode but continue to execute anyway. Further, if the response time computed via (5) exceeds the task period, then the value calculated may be optimistic; in general analysis that is suitable for arbitrary deadlines ($D_i \geq T_i$) [48] would be required in that case. Fortunately, such complex analysis is unnecessary here due to the way in which the response time $R_k(HI)$ for a LO-criticality task is used in the multi-set calculation. Instead, we may limit the maximum value of $R_k(HI)$ to the task's period T_k (i.e. if (10) converges on a value of $R_k(HI) < T_k$ we use that value, otherwise iteration stops as soon as $R_k(HI) \geq T_k$ and a value of $R_k(HI) = T_K$ is assumed).

We now show why this is sufficient to account for the maximum possible number of preemptions. Consider the term $E_j(R_k)E_k(R_i)$ in (5). This term bounds the maximum number of times that task τ_j could potentially preempt task τ_k during the response time of task τ_i. (Note R_k is $R_k(HI)$ and R_i is $R_i(HI)$ in the case we are interested in). Further, the maximum number of preemptions by task τ_j during the response time of task τ_i is limited to $E_j(R_i(HI))$ in (6). Thus the maximum possible number of preemptions of τ_k by τ_j that could contribute to the context switch cost is given by:

$$\begin{aligned} &\min(E_k(R_i(HI))E_j(R_k(HI)), E_j(R_i(HI))) \\ &= \min\left(\left\lceil \frac{R_i(HI)}{T_k} \right\rceil \left\lceil \frac{R_k(HI)}{T_j} \right\rceil, \left\lceil \frac{R_i(HI)}{T_j} \right\rceil \right) \end{aligned} \tag{11}$$

Since the following property holds for the ceiling function $\lceil x/y \rceil \lceil y/z \rceil \geq \lceil x/z \rceil$, then for any value of $R_k(HI) \geq T_k$ in (11), the minimum value is given by the term on the right hand side of the $\min()$ function. Hence $R_k(HI) = T_k$ suffices to correctly account for any value of $R_k(HI) \geq T_k$. Intuitively, once the response time $R_k(HI)$ reaches the task's period, then the task could be active at any time and thus the bound on the number of preemptions is only limited by the number of releases of the higher priority task.

V. Schedulability Analysis for AMC

In this section we derive schedulability analysis for Adaptive Mixed Criticality (AMC) scheduling [9], assuming varying context switch times. With AMC, if a job of a HI-criticality task τ_i executes for its $C_i(LO)$ WCET estimate without completing, then the system enters HI-criticality mode. AMC differs from SMC in that in HI-criticality mode, previously released jobs of LO-criticality tasks are aborted (or have their priorities reduced so that they no longer interfere with HI-criticality tasks) and subsequent releases of LO-criticality tasks are not started. Similar to SMC, only HI-criticality tasks are required to be schedulable in HI-criticality mode.

The behavior of all tasks in LO-criticality mode is the same for both AMC and SMC, hence the analysis of LO-criticality mode presented in sections IV-A and IV-B also applies to AMC. Below we present analysis for HI-criticality tasks covering HI-criticality mode and the transition to it.

A. Simple Analysis and Refined Analysis

The analysis for AMC first computes the worst-case response time $R_i(LO)$ for each task τ_i in LO-criticality mode via (7). The response time $R_i(HI)$ of a HI-criticality task, covering HI-criticality mode and the transition to it, can be determined by the following fixed point iteration, where $\gamma_{i,j}$ is defined as C^C for the simple analysis and by (3) for the refined analysis.

$$\begin{aligned} R_i(HI) &= C_i(HI) + C^C \\ &+ \sum_{\forall j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil (C_j(HI) + \gamma_{i,j}) \\ &+ \sum_{\forall j \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil (C_j(LO) + \gamma_{i,j}) \end{aligned} \tag{12}$$

where $hpH(i)$ is the set of HI-criticality tasks with priority higher than that of task τ_i and $hpL(i)$ is the set of LO-criticality tasks with priority higher than that of task τ_i.

Equation (12) limits the interference from LO-criticality tasks by noting that no further jobs of these tasks can be released after the change to the HI-criticality mode which must occur at or before $R_i(LO)$.

B. Multi-set Analysis

We now apply the techniques used in the multi-set approach (Section III-C) to AMC. The analysis for AMC first computes the worst-case response time $R_i(LO)$ for each task τ_i in LO-criticality mode via (9).

Since under AMC, LO-criticality tasks do not execute in HI-criticality mode, we only need to consider the response time $R_i(HI)$ of each HI-criticality task τ_i in that mode. In the calculation of $R_i(HI)$ for a HI-criticality task τ_i, given in (13) below, we separately consider the context switch costs due to preemptions by some higher priority HI-criticality task τ_j, given in total by $\gamma_{i,j}^M(HI)$ and those due to preemptions by some higher priority LO-criticality task τ_j, given by $\gamma_{i,j}^M(LO)$. Note, the latter are limited to occurring during $R_i(LO)$.

$$\begin{aligned} R_i(HI) &= C_i(HI) + C^C \\ &+ \sum_{\forall j \in hpH(i)} \left(\left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \gamma_{i,j}^M(HI) \right) \\ &+ \sum_{\forall j \in hpL(i)} \left(\left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_j(LO) + \gamma_{i,j}^M(LO) \right) \end{aligned} \tag{13}$$

We first derive an upper bound on $\gamma_{i,j}^M(HI)$ by considering the potential for large context switch costs C^C due to the preemptions of task τ_i and intermediate priority tasks by jobs of a HI-criticality task τ_j, within the response time of task τ_i. Here we need only consider those tasks, in aff(i,j), that can execute during the busy period of task τ_i and can be preempted by τ_j. We consider the HI- and LO-criticality tasks in this set separately. We use affL$(i,j) = hepL(i) \cap lpL(j)$ to denote the LO-criticality tasks, and similarly affH$(i,j) = hepH(i) \cap lpH(j)$ to denote the HI-criticality tasks.

HI-criticality task τ_j can preempt each job of a lower priority LO-criticality task τ_k at most $E_j(R_k(LO))$ times. This is the case since the response time of τ_k cannot exceed $R_k(LO)$ in LO-criticality mode, and if the system enters HI-criticality mode then τ_k would be aborted. Further, LO-criticality task τ_k can execute at most $E_k(R_i(LO))$ times during the response time of HI-criticality task τ_i. This is the case since if the response time of τ_i exceeds $R_i(LO)$, then the system must have entered HI-criticality mode and so no more releases of LO-criticality task τ_k are permitted. Thus the number of preemptions of τ_k by τ_j, within the response time of τ_i is upper bounded by $E_j(R_k(LO))E_k(R_i(LO))$. (By comparison the upper bound on the number of preemptions is $E_j(R_k(HI))E_k(R_i(HI))$ for SMC).

By contrast, HI-criticality task τ_j can preempt each job of a lower priority HI-criticality task τ_h at most $E_j(R_h(HI))$ times. Further, HI-criticality task τ_h can execute at most $E_h(R_i(HI))$ times during the response time of task τ_i. Thus the number of preemptions of τ_h by τ_j, within the response time of τ_i is upper bounded by $E_j(R_h(HI))E_h(R_i(HI))$. (This is the same as for SMC).

To compute $\gamma_{i,j}^M(HI)$, we first construct a multi-set $M_{i,j}^{HI}$ which contains all the possible context switch costs due to preemptions by HI-criticality task τ_j of jobs of LO-criticality tasks $\tau_l | l \in$ affL(i,j) and of jobs of HI-criticality tasks $\tau_h | h \in$ affH(i,j) which could potentially occur during the response time of HI-criticality task τ_i.

$$\begin{aligned} &M_{i,j}(HI) = \\ &\bigcup_{h \in \text{affH}(i,j)} \left(\bigcup_{E_j(R_h(HI))E_h(R_i(HI))} \left\{ \begin{array}{ll} C^C & \text{if } A_h \neq A_j \\ C^S & \text{otherwise} \end{array} \right\} \right) \\ &\cup \bigcup_{l \in \text{affL}(i,j)} \left(\bigcup_{E_j(R_l(LO))E_l(R_i(LO))} \left\{ \begin{array}{ll} C^C & \text{if } A_l \neq A_j \\ C^S & \text{otherwise} \end{array} \right\} \right) \end{aligned} \tag{14}$$

From the multi-set, we then obtain an upper bound $\gamma_{i,j}^M(HI)$ on the total context switch costs caused by the maximum number, $E_j(R_i(HI))$, of preemptions that can occur due to jobs of task τ_j executing within the response time of τ_i.

$$\gamma^M_{i,j}(HI) = \sum_{q=1}^{E_j(R_i(HI))} F(q, M_{i,j}(HI)) \tag{15}$$

where $F(q, M_{i,j}(HI))$ returns the q-th largest value from the multi-set $M_{i,j}(HI)$.

The derivation of an upper bound on $\gamma^M_{i,j}(LO)$ follows a similar approach. Here, we are interested in the potential for large context switch costs C^C due to the preemptions of task τ_i and intermediate priority tasks by jobs of a LO-criticality task τ_j, within the response time of HI-criticality task τ_i. Since τ_j is a LO-criticality task, it can only execute and hence only preempt when the system is in LO-criticality mode. $\gamma^M_{i,j}(LO)$ is therefore defined according to (6), with all of the response time values used in both (6) and (5) taking their $R(LO)$ values (i.e. $R_i(LO)$ in (6) and $R_k(LO)$ and $R_i(LO)$ in (5)). (By comparison, the larger $R(HI)$ values are used for SMC).

VI. Dominance Relationships

A scheduling policy X is said to *dominate* a policy Y if all tasks sets that are schedulable under Y are also schedulable under X, and there are also task sets that are schedulable under X, but not under Y.

We observe that, by construction, for each scheduling policy (FPPS, SMC, and AMC) the multi-set analysis dominates the refined analysis which in turn dominates the simple analysis. Further, for each approach to accounting for context switch costs (multi-set, refined, simple) the AMC analysis dominates the SMC analysis which in in turn dominates the FPPS analysis.

VII. Priority Assignment

As shown in section III, Deadline Monotonic Priority Ordering (DMPO) is not optimal for the refined or multi-set analysis for FPPS, and nor are those analyses compatible with Audsley's OPA algorithm. Further, since the counter-examples presented in Section III do not require different execution time estimates $C(LO)$ and $C(HI)$, then those negative results also apply to the refined and multi-set analyses for SMC and AMC. (Note that Audsley's algorithm is optimal for SMC and AMC with no context switch costs, and also with a simple analysis of context switch costs which inflates execution times).

Below, we introduce a priority assignment heuristic for a simplified task model which assumes that all of the LO-criticality tasks are mapped to a single process and share a common address space A^L, and similarly that all of the HI-criticality tasks are mapped to a single process and share an address space A^H. (This is an efficient arrangement since it means that only context switches between criticality levels incur a large cost).

The intuition for the priority ordering heuristic is that deadlines have the largest impact on priority assignment, and so in the cases where DMPO does not provide a feasible priority ordering, it is likely that if a feasible ordering exists it will be similar to DMPO (i.e. it may be obtained from DMPO by swapping just a few tasks in the priority order). Preliminary experiments using exhaustive search for a feasible priority ordering showed this to be the case.

The heuristic shown in Algorithm 1 therefore starts with DMPO. If DMPO is not schedulable, then the algorithm swaps the priority of two neighboring tasks and determines if the system is schedulable with the new priority order. If not, then another pair of neighboring tasks will have their priorities swapped. If this is not successful, then DMPO will be restored and the algorithm then proceeds with the next pair of neighboring tasks. The algorithm thus explores at most n^2 priority orderings, compared to $n!$ with an exhaustive search. This keeps the runtime tractable, while providing effective performance. (Note, for the multi-set experiments with 10 tasks, shown in Fig. 2 in Section IX, analysis using exhaustive search took over 70 minutes, whereas with the priority assignment heuristic it took less than 30 seconds. With more than 10 tasks, exhaustive search quickly becomes intractable).

Algorithm 1: PriorityAssignmentHeuristic($\{\tau_1 \dots \tau_n\}$)

```
1: bool isSchedulable = false;
2: int i = 1;
3: while (¬ isSchedulable ∧ i < $n - 1$) do
4:     {Outer loop, swaps the priority of two consecutive tasks};
5:     swapPriority($i, i + 1$);
6:     isSchedulable = checkSchedulability();
7:     if (isSchedulable) then
8:         break;
9:     end if
10:    int j = i;
11:    while (¬ isSchedulable ∧ j < $n - 1$) do
12:        {Inner loop, swaps the priority of two consecutive tasks};
13:        swapPriority($j, j + 1$);
14:        isSchedulable = checkSchedulability();
15:        if (isSchedulable) then
16:            break;
17:        end if
18:        swapPriority($j + 1, j$);
19:        $j = j + 1$;
20:    end while
21:    {If not successful, roll back};
22:    swapPriority($i + 1, i$);
23:    $i = i + 1$;
24: end while
25: return isSchedulable;
```

VIII. Explicit Cache Management and Context Switch Costs

In this section, we recap on the approach of Whitham et al. [52] to explicit cache management, which saves and restores the cache contents on context switches.

Whitham et al. assume a direct mapped cache (write-through for data cache), which is supplemented by a *Cache Budget Register* (CBR) that records the number of cache lines to be saved/restored, a *save/restore stack* (SRS) and *control logic*. The SRS is used to store the tag values[1] for the cache lines used by preempted tasks. The control logic implements a state machine to control cache filling and a control port to allow the RTOS software to initiate save and restore operations.

On a context switch, the save operation pushes the CBR and the tags for the specified number of cache blocks on to the SRS. The associated restore operation undoes the effects

[1]A tag encodes the address of the memory block stored in a cache line.

of the save. It uses the CBR to determine the number of tag values to pop from the SRS. For each tag value popped, it loads the associated memory block from main memory into the cache. Finally, it pops the CBR value from the SRS. Note that since save operations involve only accesses to the local SRS, as opposed to main memory, they are much faster than restore operations.

Whitham et al. [52] prototyped explicit cache management on a Xilinx Spartan- 6 FPGA, using a Xilinx Microblaze IP core. The design is illustrated in Figure 1, reproduced from [52].

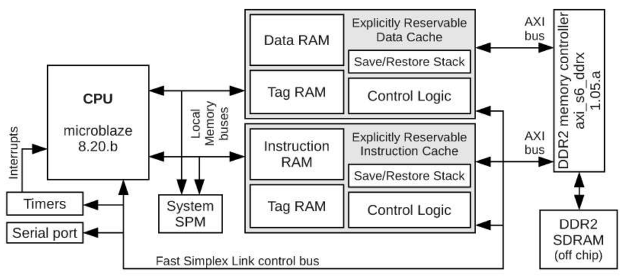

Fig. 1. FPGA prototype explicit cache management

The prototype implementation runs at 75 MHz (i.e. a 13.3ns clock cycle). On this system, context switching, to and from a task without saving and restoring the data and instruction cache contents, takes 28 μs. The time to save and restore the cache contents depends on the total size of the cache and the size of each cache line. Assuming data and instruction caches of equal size and 32 byte cache lines, the total save and restore time in nanoseconds (ns) can be estimated from measurements of the system according to the following formula:

$$overhead = 651ns + ((N * 2) * 147)ns \qquad (16)$$

where N is the number of 32 byte cache lines in each of the 2 caches (data and instruction). The total save and restore overhead, covering both caches, is given in Table I for a variety of different cache sizes.

TABLE I
EXPLICIT CACHE MANAGEMENT OVERHEADS

Data cache and instruction cache	Save and restore overhead
2 KBytes	19.47 μs
4 KBytes	38.29 μs
8 KBytes	75.92 μs
16 KBytes	151.80 μs
32 KBytes	301.70 μs
64 KBytes	602.76 μs
128 KBytes	1204.88 μs

Note that the prototype implementation does not include management of virtual memory (it only uses a single address space).

While we used the prototype implementation to provide realistic values for different context switch costs, the analysis presented in Sections III, IV, and V is not restricted to this implementation or values. It can be applied to any system using FPPS, SMC, or AMC scheduling, where tasks can be classified into different groups with different costs for inter- and intra-group context switches. In practice, the actual costs incurred would depend on the set of thread-level and process-level resources that need to be switched and the time taken to save and restore their state.

IX. Experimental Evaluation

In this section we provide a systematic evaluation of the performance of the three different forms of analysis: simple, refined, and multi-set, for each of the three scheduling policies: FPPS, SMC, and AMC, using synthetic task sets. The aim of these experiments is to show the improvement in schedulability (i.e. guaranteed performance) that can be obtained by accounting for two different context switch costs rather than a single large cost i.e. the improvement that the refined and multi-set approaches using the priority assignment heuristic provide over the simple approach which is representative of the current state-of-the-art.

A. Task set parameter generation

The task set parameters used in our experiments were generated as follows:

- The number of tasks n in each task set was 10.
- Task utilizations ($U_i = C_i/T_i$) were generated using the UUnifast algorithm [17], giving an unbiased distribution of utilization values.
- Task periods were generated according to a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible task period. This represents a spread of task periods from 10ms to 1 second, as found in many hard real-time applications.
- Task deadlines were set equal to their periods.
- The LO-criticality execution time of each task was set based on the utilization and period: $C_i(LO) = U_i/T_i$.
- The HI-criticality execution time of each task was a fixed multiplier of the LO-criticality execution time, $C_i(HI) = CF \cdot C_i(LO)$, where $CF = 2.0$.
- The probability that a generated task was a HI-criticality task was given by the parameter CP, where $CP = 0.5$.
- All LO-criticality tasks were mapped to a single process and address space A^L. Similarly, all HI-criticality tasks were mapped to another single process and address space A^H. Thus large context switch costs are only incurred when switching between tasks of different criticality levels.

All task parameters were computed in integer units of microseconds (μs), for example task periods ranged from 10^4 to 10^6 μs.

The default values used for small and large context switch costs were $C^S = 30\mu s$ and $C^C = 600\mu s$ respectively. These values correspond approximately to the cost of a basic context switch on the prototype hardware (see Section VIII), and to the cost of a context switch where 64 KByte data and instruction caches are saved and restored. The effect of varying the large context switch cost is examined in Section IX-C.

B. Baseline experiment

In our experiments, the task set utilization was varied from 0.025 to 0.975[2]. For each utilization value, 1000 task sets were

[2]Note utilization was computed from the $C(LO)$ values only.

generated and the schedulability of those task sets determined using each of the three scheduling policies: FPPS, SMC, and AMC, and four forms of analysis: simple, refined, multi-set, and no-cost, which assumes (optimistically) that all context switch costs are zero. Results were also obtained for the multi-set analysis combined with the priority assignment heuristic and combined with exhaustive priority optimization. In all other cases, Deadline Monotonic Priority Order (DMPO) was used. The graphs are best viewed via an electronic copy of the paper in color.

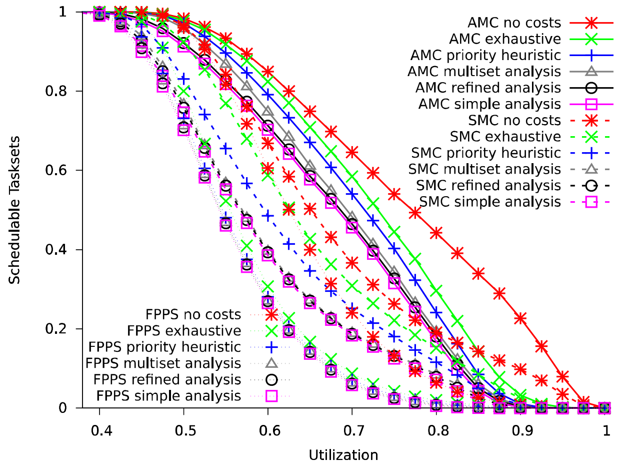

Fig. 2. Success ratio for baseline configuration.

The results of the baseline experiment are shown in Figure 2, which plots the percentage of task sets generated, using the default parameters specified in Section IX-A, that were deemed schedulable by each of the analyses at utilization levels from 0.4 to 1.0.

Observe that for AMC, the improvement obtained by using multi-set analysis as opposed to refined or simple analysis is much larger than it is for SMC or FPPS. This is because the multi-set analysis for AMC is able to account for the fact that after the LO-criticality response time $R_h(LO)$ of a HI-criticality task of interest τ_h, LO-criticality tasks can no longer execute and hence any further preemptions by higher priority HI-criticality tasks can only incur a small context switch cost. With SMC and FPPS, LO-criticality tasks are eligible to execute during all of $R_h(HI)$, and so preemptions by higher priority HI-criticality tasks can incur a large cost over that longer interval.

Both the priority assignment heuristic and exhaustive priority optimization further improve the performance of the multi-set analysis for SMC and AMC. With the heuristic providing close to optimal performance, when compared to exhaustive search on small task sets.

C. Weighted schedulability experiments

In this section, we provide additional experimental results showing how the performance of the analysis techniques varies with: (i) the number of orders of magnitude range between the minimum and maximum task period, (ii) the task set cardinality, and (iii) the large context switch cost C^C. In these experiments, all parameters, with the exception of the one being varied, took their default values as described in Section IX-A. Note that due to the long runtime, we do not show results for exhaustive priority optimization on these graphs.

In Figures 3, 4, and 5, we show the weighted schedulability measure $W_y(p)$ [13] for schedulability test y as a function of some parameter p. For each value of p, this measure combines results for the task sets τ generated for all of a set of equally spaced utilization levels (0.025 to 1.0 in steps of 0.025).

Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for a task set τ with parameter value p:

$$W_y(p) = \Big(\sum_{\forall \tau} U(\tau) \cdot S_y(\tau, p)\Big) / \sum_{\forall \tau} U(\tau) \qquad (17)$$

where $U(\tau)$ is the utilization of task set τ.

The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [13]. Weighting the results by task set utilization reflects the higher value placed on being able to schedule higher utilization task sets.

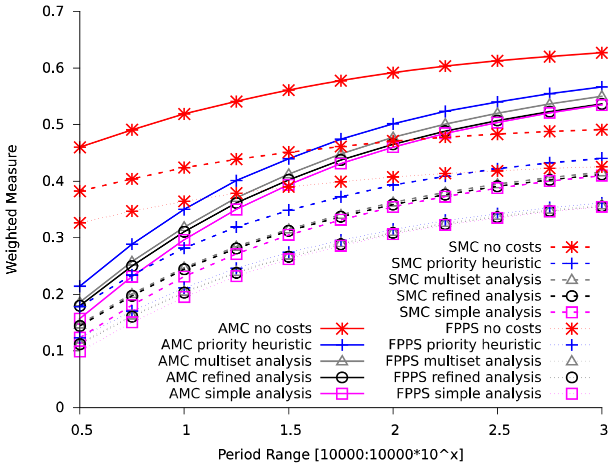

Fig. 3. Weighted schedulability for varying range of task periods.

Figure 3 shows how schedulability varies with the range of task periods from a factor of $10^{0.5} \approx 3$ difference between the maximum and minimum period to a factor of $10^3 = 1000$ difference. Observe that as the minimum period remains fixed, schedulability is reduced in all cases as the range of periods becomes smaller. There are two reasons for this. Firstly, task sets with a wider range of periods are intrinsically easier to schedule with fixed priorities, as shown by the lines for no context switch costs. Secondly, when context switch costs are taken into account, then the reduction in task periods also reduces schedulability since the ratio of context switch time to task execution times increases. Note that as the range of task periods reduces, the relative performance of the priority assignment heuristic improves. This happens because with a smaller range of task periods, the tasks have similar deadlines, and so there is more scope to adjust the priority ordering away from DMPO to reduce the overall context switch costs.

Figure 4 shows how schedulability varies with task set cardinality. Here, as the number of tasks increases, so the impact of context switch costs becomes larger and so

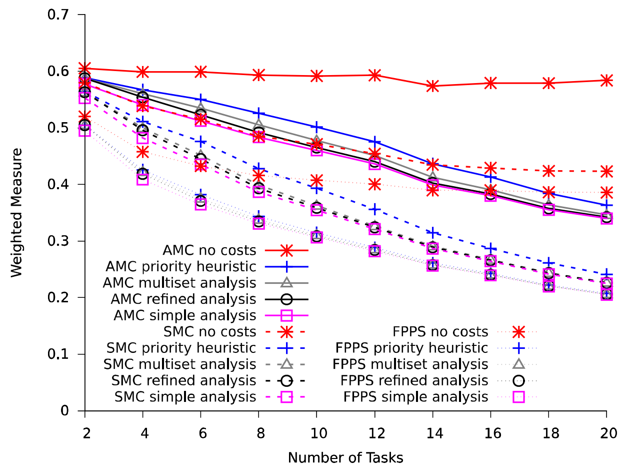

Fig. 4. Weighted schedulability for varying task set cardinality.

schedulability decreases for all analyses except those marked "no costs" which do not account for such costs. As the number of tasks increase, so does the relative improvement obtained by using the multi-set analysis combined with the priority assignment heuristic for SMC and AMC.

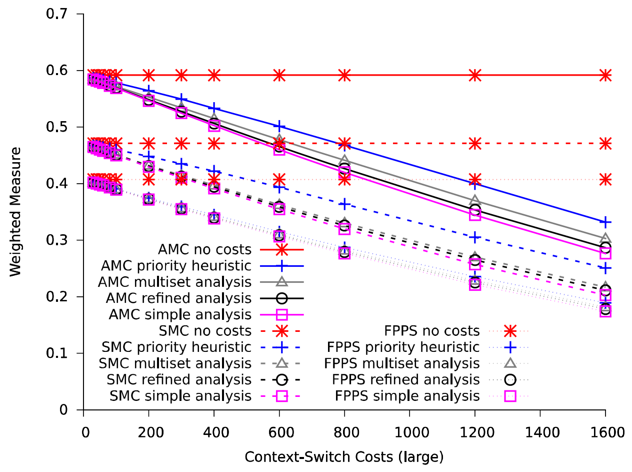

Fig. 5. Weighted schedulability for varying context switch cost (large).

Figures 5 shows how schedulability varies with increasing context switch costs. As expected, weighted schedulability decreases approximately linearly with increasing context switch costs. As the context switch costs increase, so the improvement obtained via the multi-set analysis combined with the priority assignment heuristic becomes more pronounced.

X. Related Work

Early work on accounting for scheduling overheads in Fixed Priority Preemptive Scheduling (FPPS) by Katcher et al. [34] in 1993, Burns [20] in 1994, and Burns et al. [23] in 1995, focused on scheduler overheads and processor context switch costs, accounting for the maximum number of preemptions that could potentially occur. Echague et al. [31] aimed to solve the problem more precisely for periodic task sets by finding the exact number of preemptions; however, they did not include the duration of the context switches which can lead to an increased number of preemptions. In 2007 Yomsi and Sorel [56] showed that for periodic task sets, the critical instant does not occur on simultaneous release of the tasks when context switch costs are considered, and derived the exact number of preemptions for each job in the hyper-period.

Much of the subsequent work has focused on providing analyses for specific types of preemption related overheads caused by hardware resources that are in use at the time of preemption, for example delays relating to the sharing of caches or scratch-pads, or on ways of ameliorating these effects, for example via cache partitioning.

Analysis of Cache Related Preemption Delays (CRPD) and their integration into schedulability analyses for FPPS used the concepts of Useful Cache Blocks (UCBs) and Evicting Cache Blocks (ECBs). In 1996, Busquets et al. introduced the ECB-Only approach [24], which considers just the preempting task; while in 1998, Lee et al. developed the UCB-Only approach [36], which considers just the preempted task(s). Both the UCB-Union approach [47] developed by Tan and Mooney in 2007, and the ECB-Union approach [1] derived by Altmeyer et al. in 2011 consider both the preempted and preempting tasks. As does an alternative approach [46] developed by Staschulat et al. in 2005. These approaches were later superseded by multiset based methods (ECB-Union Multiset and UCB-Union Multiset) which dominate them [2]. These methods have subsequently been adapted to EDF scheduling by Lunniss et al. [39] [38].

Cache partitioning is one way of eliminating CRPD; however, this results in inflated WCETs due to the reduced cache partition size available to each task. In 2014, Altmeyer et al. [3], [4] derived an optimal cache partitioning algorithm for the case where each task has its own partition. They showed that the trade off between longer WCETs and CRPD often favors sharing the cache rather than partitioning it.

An alternative form of local memory to cache is a scratch pad. If a scratch pad is to be shared between tasks in a preemptive system, then either the scratch pad contents need to be static (i.e. shared by all tasks) which can be ineffective, or the scratch pad contents need to be dynamic, i.e. saved and restored on preemptions. In 2012, Whitham et al. [52], [53] developed a dynamic scratch pad memory reuse scheme and showed how Scratch pad Related Preemption Delays (SRPD) could be integrated into schedulability analysis for FPPS.

Other work has focused on adaptations to the FPPS policy that reduce the number of context switches and their impact, for example scheduling using preemption thresholds and limited or deferred preemption.

Preemption thresholds [35], [43], [44], [51] provide a means of reducing the number of preemptions by making certain groups of tasks non-preemptable with respect to each other. In 2014, Bril et al. [18] integrated CRPD into analysis for fixed-priority scheduling with preemption thresholds. Further work in this area by Wang et al. [50] in 2015 showed that by using preemption thresholds, groups of tasks can share a cache partition while still avoiding CRPD.

Two different models of fixed priority scheduling with deferred preemption have been developed. In the *fixed model*, introduced by Burns in 1994 [20], preemption is only permitted at fixed *preemption points* within the code of each task. This method is also referred to as *co-operative scheduling*. In the floating model [11], [55], an upper bound is given on the length of the longest non-preemptive region of each task, at run-time non-preemptive behavior is then controlled by the operating system. Exact schedulability analysis for the fixed model was derived by Bril et al. in 2009 [19]. In 2010, Bertogna et al. integrated preemption effects into analysis of the fixed model, considering both fixed [15] and variable [16] costs. In 2012, Davis and Bertogna [28], derived an optimal method of assigning both priorities and the length of the final non-preemptive region of each task in order to maximize schedulability. In 2015, Cavicchio et al. [26] derived an optimal method of placing preemption points that minimizes CRPD. For further information on limited preemption scheduling see the survey by Buttazzo et al. [25].

In 2014, Mohan et al. [40], [41] presented analysis for fixed priority non-preemptive scheduling, assuming that a Flush Task (FT) is required to run whenever execution switches from a high security task to a low security task. A min-cost flow graph formulation was used to upper bound the maximum number of flushes required within the response time of a task. The upper bound is found by considering (in polynomial time) possible permutations of the order of the jobs within the busy period of the task, without regard to their actual arrival times. Further work in this area considered both non-preemptive and preemptive scheduling [42]. The approach has also been adapted to mixed criticality systems scheduled using non-preemptive AMC [8].

The analysis derived in this paper builds on the schedulability tests for FPPS [5], [33], SMC [10], and AMC [9]. It and employs some of the multi-set techniques first used for CRPD analysis in [2], tailored to the analysis of context switch costs.

XI. Conclusions

In mixed criticality systems, it is vital to ensure that there is sufficient separation between tasks of LO- and HI-criticality applications, so that the behavior or mis-behavior of the former cannot affect the functional or timing behavior of the latter. To ensure appropriate spatial isolation, the memory address spaces used by LO- and HI-criticality tasks must be distinct. In this paper, we modeled tasks as belonging to different processes. Tasks within the same process share an address space that is distinct from the address spaces used by tasks belonging to other processes. A consequence of this separation is that the cost of switching between tasks of the same same process (e.g. the same criticality level) is low, whereas the cost of context switching between tasks of different processes (e.g. different criticality levels) can be much higher, due to the additional resources that need to be switched.

We assumed that *process-level* context switches, which switch address space, also require that the cache contents are saved and subsequently restored via an explicit cache management mechanism. Combined with cache partitioning between tasks belonging to the same process, this mechanism eliminates Cache Related Preemption Delays (CRPD). This ensures that the only impact of a LO-criticality task (belonging to one process) on a HI-criticality task (belonging to another process) is due to the task execution time which is enforced by the RTOS, and a fixed context switch cost. This approach also has the advantage that it prevents security hazards due to a compromised low security process obtaining information from the cache contents of a high security process which it either preempts or follows.

Measurements from an existing prototype implementation on a 75 MHz FPGA showed that the context switch costs are of the order of $30\mu s$ for a thread-level context switch and $600\mu s$ for a process-level context switch, which additionally saves and restores the contents of 64 Kbyte data and instruction caches.

The main contribution of this paper is in integrating the differing context switch costs into schedulability analysis for fixed priority preemptive scheduling, and the two mixed criticality scheduling schemes, SMC and AMC, based on it. We derived simple, refined, and multi-set analyses for each scheme. Further, our analysis covers the general case where multiple applications (of either HI- or LO-criticality) are each mapped to a different process. It assumes that there are two different context switch costs, corresponding to inter-process context switches (a large cost) and intra-process context switches (a small cost). Many different processes may be considered with no increase in complexity in the analysis.

We showed that the refined and multi-set analyses are not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [6], [7]. We therefore proposed a heuristic priority assignment technique aimed at improving schedulability by reducing the number of high cost context switches for systems where all HI-criticality tasks are mapped to one process, and all LO-criticality tasks to another. Our systematic evaluation showed the effectiveness of the different analyses and the priority assignment technique, using representative context switch costs obtained from the prototype system.

In practice, the cost of context switches must be accounted for in any valid schedulability analysis used to verify the timing correctness of a real-time system using pre-emptive scheduling. Accounting for two different context switch costs provides a more precise analysis than subsuming a single large context switch cost into task execution times or ignoring these costs altogether. Compared to simple analysis, the more precise multi-set approach can provide the headroom necessary to add more functionality to a system without requiring costly hardware upgrades or software optimization. Alternatively, it may show that a system is schedulable when the simple analysis does not, avoiding the need for unnecessary and costly changes.

One of the disadvantages of employing fully preemptive scheduling is the large number of context switches that may occur. There are a number of techniques that have been developed that can reduce the number of context switches and hence improve schedulability. In future, we aim to explore the integration of different context switch costs into mixed criticality scheduling with deferred preemption [21] and with preemption thresholds [57], as well as extension of the approach to systems with more than two criticality levels [32].

Acknowledgements

The research in this paper is partially funded by the ESPRC grant, MCCps (EP/K011626/1) and by the NWO Veni Project "The time is now: Timing Verification for Safety-Critical Multi-Cores". EPSRC Research Data Management: No new primary data was created during this study.

References

[1] S. Altmeyer, R. I. Davis, and C. Maiza. Cache related pre-emption aware response time analysis for fixed priority pre-emptive systems. In *RTSS*, pages 261–271, December 2011.

[2] 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*, 48(5):499–526, 2012.

[3] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. Evaluation of cache partitioning for hard real-time systems. In *ECRTS*, pages 15–26, July 2014.

[4] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. On the effectiveness of cache partitioning in hard real-time systems. *Real-Time Systems*, pages 1–46, Jan 2016.

[5] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[6] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Dept. Computer Science, University of York, UK, 1991.

[7] N.C. Audsley. On priority assignment in fixed priority scheduling. pages 39–44, May 2001.

[8] Hyeongboo BAEK and Jinkyu LEE. Incorporating security constraints into mixed-criticality real-time scheduling. *IEICE Transactions on Information and Systems*, E100.D(9):2068–2080, 2017.

[9] S. Baruah, A. Burns, and R.I. Davis. Response-Time Analysis for Mixed Criticality Systems. In *Real-Time Systems Symposium (RTSS), IEEE*, pages 34–43, 2011.

[10] S. Baruah and S. Vestal. Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 147–155, 2008.

[11] S. K. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 6-8 July 2005, Palma de Mallorca, Spain, Proceedings*, pages 137–144, 2005.

[12] S.K. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky, editor, *Proc. of Reliable Software Technologies - Ada-Europe 2011*, pages 174–188. Springer, 2011.

[13] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *OSPERT*, pages 33–44, July 2010.

[14] I. Bate, A. Burns, and R.I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[15] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, pages 251–260. IEEE, 2010.

[16] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 217–227. IEEE, 2011.

[17] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30:129–154, May 2005.

[18] R. J. Bril, S. Altmeyer, M. M. H. P. van den Heuvel, R.I. Davis, and M. Behnam. Integrating cache-related pre-emption delays into analysis of fixed priority scheduling with pre-emption thresholds. In *RTSS*, pages 161–172, 2014.

[19] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42(1):63–119, 2009.

[20] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[21] A. Burns and R. I. Davis. Adaptive mixed criticality scheduling with deferred preemption. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 21–30. IEEE, 2014.

[22] A. Burns and R. I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, November 2017.

[23] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480, May 1995.

[24] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *RTAS*, pages 204–212, June 1996.

[25] G. C. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013.

[26] J. Cavicchio, C. Tessler, and N. Fisher. Minimizing cache overhead via loaded cache blocks and preemption placement. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 163–173, July 2015.

[27] Y. Chen, K.G. Shin, and H. Xiong. Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems. *Information Processing Letters*, 116(8):508–512, 2016.

[28] R. I. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 39–50. IEEE, 2012.

[29] R.I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Real-Time Systems, Volume 47, Issue 1*, pages 1–40, 2010.

[30] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems Journal*, 46(3):305–331, 2010.

[31] J. Echague, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proceedings Seventh Euromicro Workshop on Real-Time Systems*, pages 184–190, Jun 1995.

[32] T. Fleming and A. Burns. Extending mixed criticality scheduling. In *Proc. WMC, RTSS*, pages 7–12, 2013.

[33] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, May 1986.

[34] D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 19, 1993.

[35] U. Keskin, R.J. Bril, and J.J. Lukkien. Exact response-time analysis for fixed-priority preemption-threshold scheduling. In *Work-in-Progress Session ETFA*, 2010.

[36] C.-G. Lee, J. Hahn, Y.-M. Seo, S.L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998.

[37] J.Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. In *Performance Evaluation 2(4)*, pages 237–250, 1982.

[38] W. Lunniss, S. Altmeyer, and R. I. Davis. A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays. *Leibniz Transactions on Embedded Systems*, 1(1):01–1–01:24, 2014.

[39] W. Lunniss, S. Altmeyer, C. Maiza, and R. I. Davis. Integrating cache related pre-emption delay analysis into edf scheduling. In *RTAS*, pages 75–84, April 2013.

[40] S. Mohan, M. K. Yoon, R. Pellizzoni, and R. Bobba. Real-time systems security through scheduler constraints. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 129–140, July 2014.

[41] S. Mohan, M-K. Yoon, R. Pellizzoni, and R. B. Bobba. Integrating security constraints into fixed priority real-time schedulers. *Real-Time Syst.*, 52(5):644–674, September 2016.

[42] R. Pellizzoni, N. Paryab, M. K. Yoon, S. Bak, S. Mohan, and R. B Bobba. A generalized model for preventing information leakage in hard real-time systems. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 271–282, April 2015.

[43] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *RTSS*, pages 315–25, December 2002.

[44] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds. In *Proceeding of the IEEE Real-Time Systems Symposium (RTSS)*, pages 25–34, December 2000.

[45] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 155–165, 2012.

[46] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *ECRTS*, July 2005.

[47] Y. Tan and V. Mooney. Timing analysis for preemptive multi-tasking real-time systems with caches. *Trans. on Embedded Computing Sys.*, 6(1), 2007.

[48] K. W. Tindell. Extendible approach for analysing fixed priority hard real-time tasks. *Journal of Real-Time Systems*, 6, 1992.

[49] S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Real-Time Systems Symposium (RTSS), IEEE*, 2007.

[50] C. Wang, Z. Gu, and H. Zeng. Integration of cache partitioning and preemption threshold scheduling to improve schedulability of hard real-time systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 69–79, 2015.

[51] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 328–335, 1999.

[52] J. Whitham, N. C. Audsley, and R. I. Davis. Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):120, 2014.

[53] J. Whitham, R.I. Davis, N.C. Audsley, S. Altmeyer, and C. Maiza. Investigation of scratchpad memory for preemptive multitasking. In *RTSS*, pages 3–13, December 2012.

[54] S. Yamada and S. Kusakabe. Effect of context aware scheduler on tlb. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[55] Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on*, pages 351–360. IEEE, 2009.

[56] P. M. Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, pages 280–290, July 2007.

[57] Q. Zhao, Z. Gu, and H. Zeng. Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 141–146, March 2013.