Mixed Criticality on Multi-cores Accounting for Resource Stress and Resource Sensitivity

Robert I. Davis  
rob.davis@york.ac.uk  
Department of Computer Science, University of York  
York, UK

Iain Bate  
ibane@york.ac.uk  
Department of Computer Science, University of York  
York, UK

ABSTRACT
The most significant trend in real-time systems design in recent years has been the adoption of multi-core processors and the accompanying integration of functionality with different criticality levels onto the same hardware platform. This paper integrates mixed criticality aspects and assurances within a multi-core system model. It bounds cross-core contention and interference by considering the impact on task execution times due to the stress on shared hardware resources caused by co-runners, and each task’s sensitivity to that resource stress. Schedulability analysis is derived for four mixed criticality scheduling schemes based on partitioned fixed priority preemptive scheduling. Each scheme provides robust timing guarantees for high criticality tasks, ensuring that their timing constraints cannot be jeopardized by the behavior or misbehavior of low criticality tasks.

CCS CONCEPTS
• Computer systems organization → Real-time systems;  
Real-time systems; • Software and its engineering → Real-time schedulability, Real-time schedulability.

KEYWORDS
real-time, multi-core, mixed criticality, fixed priority, schedulability analysis, cross-core contention, interference

ACM Reference Format:
https://doi.org/10.1145/3534879.3534883

1 INTRODUCTION
The most significant trend in real-time systems design in recent years has been the migration from using single-core to multi-core processors [1, 2] and the accompanying integration of functionality of different criticality levels onto the same hardware platform, i.e. the advent of mixed criticality systems [65].

In mixed criticality systems, the main challenge is to provide appropriate levels of assurance, such as timing guarantees, to software tasks that have different levels of criticality. Crucially, this needs to be done without having to treat all of the tasks as having the highest level of criticality, with the attendant increase in verification costs and reduction in usable system capacity that would entail. In multi-core systems, the main challenge is to bound and correctly account for the effects of cross-core contention over shared hardware resources, due to tasks running on different cores, and the impact that has on task response times and consequently on system schedulability.

In this paper, we consider mixed criticality multi-core systems with two criticality levels. More specifically, HI- and LO-criticality tasks running on a multi-core processor that are subject to cross-core contention and interference over shared hardware resources. In this context, HI-criticality tasks must be afforded robust timing guarantees, such that their timing constraints cannot be jeopardized by the behavior or misbehavior of LO-criticality tasks running on either the same or different cores. Following Vestal’s model [65], LO-criticality tasks have a single low assurance estimate of their standalone Worst-Case Execution Time (WCET), whereas HI-criticality tasks have two such estimates; one low assurance estimate and a larger high assurance estimate that may, for example, include provision for error handling code that is not expected to execute during normal operation [50].

Multi-core processors typically share hardware resources, such as the interconnect and the memory hierarchy, between cores. Unfortunately, a consequence of these hardware design decisions is that the execution time of a task running on one core can be impacted by co-running tasks on other cores contending with it for access to shared hardware resources. This increase in execution time is referred to as interference.

Work on micro-benchmarks [36, 44, 56, 60, 61] has sort to characterize the maximum amount of interference that a task can be subject to, assuming a given multi-core hardware configuration. Further research on the Multi-core Resource Stress and Sensitivity (MRSS) task model [30, 31] takes this idea a step further, aiming to bound the total amount of interference that can occur from two different perspectives by employing additional task parameters:

1. The Resource Sensitivity of a task characterizes the maximum increase in its execution time that can occur due to contention over a specific resource emanating from any possible co-runner.
2. The Resource Stress of a task characterizes the maximum increase in the execution time of any possible co-runner due to contention over a resource emanating from the task.
The resource sensitivity and resource stress parameters of a task characterize, in a simple but useful way, the impact on that task’s execution time of contention over the resource and the behavior of the arbitration policy in controlling access to it. See [30, 31] for a discussion of how the resource sensitivity and resource stress parameters can be obtained.

By combining measures of resource stress and resource sensitivity, analysis of the MRSS task model can more accurately bound the amount of interference that can actually occur. The MRSS task model retains the advantages of the two-step approach that is traditionally employed on single-core systems, providing a separation of concerns between timing analysis and schedulability analysis, and has been validated via a proof-of-concept case study on multi-core hardware [30, 31].

This paper builds on the MRSS task model and its schedulability analysis for partitioned fixed priority preemptive scheduling on multi-core systems. The main contribution of this work is the integration of mixed criticality and multi-core in the form of the MRSS task model, along with the derivation of schedulability analysis for four mixed criticality scheduling schemes.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 introduces the system model, terminology, and notation used. Section 4 presents schedulability analysis for the four mixed criticality schemes studied, with a systematic evaluation of their performance given in Section 5. Finally, Section 6 concludes with a summary.

2 RELATED WORK

In this section, we outline prior work on: (i) mixed criticality fixed priority scheduling schemes for single-core processors, since those schemes form the basis for partitioned multi-core systems; (ii) mixed criticality systems on multi-cores that seeks to enforce limits on the amount of cross-core contention and interference that can occur; and (iii) single criticality multi-core systems that integrate interference effects into schedulability analysis.

Since Vestal’s seminal paper [65] in 2007, mixed criticality systems have become a hot topic of real-time systems research, see [20, 21] for a survey and a more recent review. Many of these papers focus on scheduling schemes that are based on fixed priorities, most notably Static Mixed Criticality (SMC) [11] and Adaptive Mixed Criticality (AMC) [12]. AMC is considered the most effective fixed priority scheme [43] for single cores, and has been extended to account for many additional aspects including: preemption thresholds [68, 69], multiple criticality levels [37], criticality-specific task periods [13], changes in priority [10], communications [18], deferred preemption [19], a fast return to LO-criticality behavior [15, 16], weakly-hard timing constraints [38], probabilistic task models [54], design optimization [71], context switch costs [28], robustness and resilience [23], implementation overheads [51], and semi-clairvoyant timing behavior [22, 70]. An exact analysis for AMC has also been developed for periodic task sets with offsets [6, 58]. Finally, a modified AMCR runtime protocol [17] has been developed that delays the onset of degraded behaviour where LO-criticality jobs are dropped.

The first work to discuss mixed criticality within the context of multi-core systems was by Anderson et al. [4, 55], with later work in this area addressing overheads [26, 42], showing the advantages of using different partitioning and isolation techniques at different criticality levels [47], and reconciling issues of data sharing [25] and simultaneous multithreading [9].

In the context of multi-core systems, much of the prior work on mixed criticality has sort to limit the amount of interference that can occur. To achieve this, criticality-based partitioning is typically assumed, with HI-criticality tasks allocated on one core, and LO-criticality tasks to other cores. Here, one way of limiting interference is to monitor the execution time of each HI-criticality task and to abort co-running LO-criticality tasks when no more interference can be tolerated [48]. A more subtle approach is to throttle the resource access bandwidth available to the LO-criticality cores, temporarily suspending execution on those cores whenever the maximum permitted number of accesses in a given period has been reached [67].

Research into the timing analysis and schedulability analysis of multi-core systems has also become a hot topic of real-time systems research over the past 15 years, see [53] for a survey. Of specific interest here is the integration of interference effects into schedulability analysis. Early work in this area [63] used arrival curves to model the memory bus accesses of each task, and how delays due to contention impact task response times. Subsequently, more detailed analysis [39, 49, 59, 64] divided each task into a sequence of blocks and used information about the number of accesses within each block to provide more refined results. Further work [57] proposed using a WCET-matrix and WCET-sensitivity values to characterize the variation in task execution times for different numbers of contending cores. A later more detailed analysis [5] considered different execution times dependent on specific co-runners, but suffered from significant scalability issues.

An alternative approach [27] used request functions to model the maximum number of resource accesses from each task in a given time interval, and integrated this request function into response time analysis. Further work [46, 66] along this line provided detailed analysis of the contention caused by memory accesses, accounting for variations in latencies due to different memory states.

Subsequently, the Multi-core Response Time Analysis (MRTA) framework [3, 29] was introduced, aimed at combining the demands that tasks place on difference types of shared resources with the resource supply provided by those resources, and integrating the resulting explicit interference directly into response time analysis. This framework was later built upon to analyze bus arbitration policies on a many-core processor [62]. Further, the symmetry between processing and resource access has been leveraged to derive a suspension-based schedulability analysis [24], with similar performance to MRTA.

3 SYSTEM MODEL

In this paper, we assume a mixed criticality multi-core system with \( m \) homogeneous cores that executes tasks under various scheduling schemes, based on partitioned fixed priority preemptive scheduling.
With partitioning, tasks are assigned to a specific core and do not migrate from one core to another.

The mixed criticality system is assumed to have two criticality levels: HI and LO. Each task $\tau_i$ is characterised by its criticality level $L_i$, which is either HI or LO. Each LO-criticality task $\tau_i$ has a single estimate $C_i(LO)$ of its WCET when executing stand-alone. By contrast, each HI-criticality task $\tau_k$ has two estimates $C_k(LO)$ and $C_k(HI)$ of its WCET when executing stand-alone, where $C_k(HI) \geq C_k(LO)$. (Note for ease of presentation of the analysis in Section 4, we assume that $C_k(LO) = C_k(HI)$ for LO-criticality tasks). Each task $\tau_i$ has a minimum inter-arrival time or period $T_i$ between releases of its job, and a constrained relative deadline $D_i$, where $D_i \leq T_i$.

Each task $\tau_i$ is assumed to have a priority that is unique across all cores, with $hp(i)$ used to denote the set of tasks with higher priority than task $\tau_i$. The priorities of tasks are unrelated to their criticality levels. The notation $\Gamma_i$ is used to denote the set of tasks that execute on the same core, with index $x$, as the task of interest $\tau_i$. Similarly, $\Gamma_x$ is used to denote the set of tasks that execute on a different core with index $y$.

The tasks are assumed to be independent, but may access a set of shared hardware resources $r \in R$, thus causing interference on the execution of tasks on other cores via cross-core contention.

Further aspects of the model are based on the concept of resource sensitive contenders and resource stressing contenders [30, 31].

A resource stressing contender maximizes the stress on a resource $r$ by repeatedly making accesses to it that cause the most contention. Running a resource stressing contender in parallel with a task creates the maximum increase in execution time for the task due to contention over resource $r$ emanating from any single co-runner.

A resource sensitive contender for a resource $r$ suffers the maximum possible interference by repeatedly making accesses to the resource that suffer the most contention. Running a resource sensitive contender in parallel with a task creates the maximum increase in execution time for any single co-running contender due to contention over resource $r$ emanating from the task.

Each task $\tau_k$ is characterised by its resource sensitivity $X_k^r$ and its resource stress $Y_k^r$ for each shared hardware resource $r \in R$. $X_k^r$ captures the maximum increase in execution time of task $\tau_k$ (from $C_1$ to $C_1 + X_k^r$) when it is executed in parallel with a resource stressing contender for resource $r$. Thus $X_k^r$ models how much task $\tau_k$ behaves like a resource sensitive contender. Similarly, $Y_k^r$ captures the increase in execution time of a resource sensitive contender for resource $r$, when it is executed in parallel with task $\tau_k$. Hence $Y_k^r$ models how much task $\tau_k$ behaves like a resource stressing contender. With this model, the execution time of a task $\tau_k$ running on one core, subject to interference via shared hardware resource $r$ from a single task $\tau_k$ running in parallel on one other core, is increased by at most $\min(X_k^r, Y_k^r)$ i.e. from $C_1$ to $C_1 + \min(X_k^r, Y_k^r)$. Assuming the worst-case stress on resource $r$ emanating from any arbitrary tasks on $m - 1$ other cores, the execution time of task $\tau_k$ is increased from $C_1$ to at most $C_1 + (m - 1)X_k^r$. Finally, the multi-core system is assumed to be symmetrical, and so the cross-core contention between two tasks over a resource does not depend on the two specific cores on which those tasks run.

We do not assume dual values for resource sensitivity $X_k^r$ and resource stress $Y_k^r$ based on criticality. For a LO-criticality task, these values reflect its LO-criticality execution behavior, but cannot impact the guarantees afforded to HI-criticality tasks under the analysis described in this paper. For a HI-criticality task the values reflect its worst-case i.e. HI-criticality execution behavior.

The Real-Time Operating System (RTOS) is required to provide standard per task execution time monitoring and budget enforcement facilities. The RTOS is assumed to abort any job of a task $\tau_i$ that does not complete within its execution time budget $B_i(L_i)$. This budget is set to $C_i(L_i) + \sum_{r \in R}(m - 1)X_k^r$, where $H$ is the set of shared hardware resources, $m$ is the number of cores, and $L_i$ is the criticality level of the task. The budget $B_i(L_i)$ thus accounts for the WCET of the task when faced with the worst-case stress on every resource $r$, from any arbitrary tasks on all of the other $m - 1$ cores. Assuming that the parameters $C_i(L_i)$ and $X_k^r$ represent sound upper bounds, then budget enforcement will only occur if the task itself executes erroneously. (Note, no enforcement is assumed on the number of accesses that can be made to shared hardware resources).

The schedulability tests introduced in this paper are named using the following convention: $CpSched-m-X-MCS$, where $C$ indicates a contention-based test for $p$ partitioned scheduling, using the basic scheduling policy Sched, which is FFPS. The test is for $m$ cores, makes use of information $X$, which is either D or R meaning the deadlines or the response times of the tasks on other cores, or $fc$ meaning fully composable, i.e. the test does not rely on any information about the tasks running on the other cores, or $no$ meaning no effects of contention are included. Finally, $MCS$ is the mixed criticality scheme employed, which is either NMC, SMC, AMC, or AMCR, as described in Section 4. This naming convention builds on that introduced in [30, 31] for schedulability tests compatible with the MRSS task model.

4 SCHEDULING SCHEMES AND ANALYSES

In this section, we derive schedulability analysis for partitioned fixed priority preemptive scheduling of mixed criticality systems on multi-cores, under four different mixed criticality scheduling schemes, in each case accounting for cross-core contention and interference, according to the MRSS task model.

Most scheduling schemes for mixed criticality systems identify two distinct modes of behavior. A normal or LO-criticality mode, which comprises the expected behavior of the system, and an abnormal or HI-criticality mode, which is expected to be rarely if ever entered as a consequence of the runtime behavior of HI-criticality tasks.

There are disparate views within the real-time systems community as to the timing requirements for mixed criticality systems [20], while most works assume that LO-criticality tasks do not have to meet their deadlines in abnormal mode, and can potentially be dropped, others [34, 35] argue that this represents a disconnect with respect to industry practice and standards. The argument against missing deadlines and job dropping is that...

1 In this first paper combining mixed criticality and the MRSS model, we choose not to use dual values so as to simplify the overall model and analyses. Models and analyses for mode specific resource sensitivity and resource stress are left for future work.)
criticality is not synonymous with importance, and thus the functionality of LO-criticality tasks cannot simply be discarded. In this section, we derive analyses for different schemes that reflect these different viewpoints. Four schemes are considered:

1. No Mixed Criticality (NMC): Assumes that jobs of all tasks are required to meet their deadlines in both normal and abnormal modes. Under NMC no runtime mode change operations are required. NMC provides a baseline for systems where missing deadlines or dropping jobs is not acceptable even for LO-criticality tasks.

2. Static Mixed Criticality (SMC) [11]: Assumes that jobs of LO-criticality tasks continue to execute and to be released in abnormal mode, but are not required to meet their deadlines in that mode. Under SMC no runtime mode change operations are required.

3. Adaptive Mixed Criticality (AMC) [12]: Assumes that no new jobs of LO-criticality tasks are released in abnormal mode, and that any previously released jobs of LO-criticality tasks are not required to meet their deadlines in that mode. With AMC, the RTOS is responsible for runtime mode change operations, and for ensuring that LO-criticality tasks do not release new jobs in abnormal mode.

4. Adaptive Mixed Criticality with modified runtime protocol (AMCR) [17]: AMCR is similar to AMC, but uses a modified runtime protocol that delays the time at which LO-criticality tasks stop releasing new jobs in abnormal mode, see Section 4.4 for details.

In this paper, we assume that each core is considered separately and independently in terms of the runtime mode change operations performed by the RTOS; however, in contrast the timing requirements placed upon the tasks are defined by the overall system behavior, with different levels of timing assurance required for HI- and LO-criticality tasks as follows:

**R1** LO-criticality tasks require assurance that they will meet their timing constraints (deadlines) under normal system behavior, i.e. under the condition that all tasks on all cores comply with their LO-criticality execution time $C_i(LO)$, resource sensitivity $X_i^r$, and resource stress $Y_i^r$ parameters.

**R2** HI-criticality tasks require more robust assurance that they will meet their timing constraints at all times (irrespective of the behavior or misbehavior of other tasks) i.e. subject only to the condition that they comply with their own HI-criticality execution time $C_i(HI)$ and resource sensitivity $X_i^r$ parameters.

### 4.1 No Mixed Criticality (NMC)

In this section, we build upon the schedulability analysis for the MRSS task model given in [30, 31], making use of the context-dependent schedulability tests for LO-criticality tasks, and the fully composable context-independent schedulability test for HI-criticality tasks, see Sections 3.1 and 3.3 of [30] respectively.

Adding cross-core interference considering each resource $r \in H$ to the standard response time analysis [8, 45] for fixed priority preemptive scheduling, we can compute the worst-case response time for mixed criticality tasks under the NMC scheme as follows:

$$
R_i(L_i) = C_i(L_i) + \sum_{j \in E_i \cap \text{hp}(i)} \left[ \frac{R_i(L_j)}{T_j} \right] C_j(L_j) + \sum_{r \in H} R_i^r(L_i) = C_i(L_i) + \sum_{j \in E_i \cap \text{hp}(i)} \left[ \frac{R_i(L_j)}{T_j} \right] C_j(L_j) + \sum_{r \in H} Y_i^r
$$

where $R_i^r(L_i)$ is an upper bound on the interference that may occur within the response time of task $t_i$, via shared hardware resource $r$, due to tasks executing on the other cores.

The interference term $R_i^r(L_i)$ depends on: (i) the total resource sensitivity for resource $r$, denoted by $S_i^r(R_i(L_i), x)$, for the tasks executing on the same core $x$ as task $t_i$ within its response time $R_i(L_i)$; and (ii) the total resource stress on resource $r$, denoted by $E_i^r(R_i(L_i), y)$, that can be produced by tasks executing on each of the other cores $y$ within an interval of length $R_i(L_i)$.

$$
I_i^r(L_i) = \sum_{y \not= x} \min(E_i^r(R_i(L_i), y), S_i^r(R_i(L_i), x))
$$

This is the case, since the maximum interference due to contention from each core $y$ cannot exceed the total resource stress $E_i^r(R_i(L_i), y)$ from that core within an interval of length $R_i(L_i)$.

The total resource sensitivity $S_i^r(R_i(L_i), x)$ is computed based on the jobs that may execute on the same core $x$ within the worst-case response time of task $t_i$, thus we have:

$$
S_i^r(R_i(L_i), x) = X_i^r + \sum_{j \in E_i \cap \text{hp}(i)} \left[ \frac{R_i(L_j)}{T_j} \right] X_j^r
$$

The total resource stress $E_i^r(R_i(L_i), y)$ due to tasks that execute on another core $y$ in the interval $R_i(L_i)$ can be upper bounded in three different ways.

When analysing a HI-criticality task $t_i$, the total resource stress $E_i^r(R_i(L_i), y)$ is assumed to be infinite, and hence the schedulability test for that task becomes context-independent and fully composable, since the computed response time is unaffected by any changes to the parameters of the tasks that execute on the other cores. In other words, when (4) is used, (1), (2), and (3) become dependent only on the set of tasks executing on the same core as $t_i$.

$$
E_i^r(R_i(L_i), y) = \infty
$$

When analysing a LO-criticality task $t_i$, the total resource stress $E_i^r(R_i(L_i), y)$ can be upper bounded in two ways, making use of either the deadlines or the response times of the contending tasks that execute on the other cores:

$$
E_i^r(R_i(L_i), y) = \sum_{j \in E_i} \left[ \frac{R_i(L_j) + D_j}{T_j} \right] Y_j^r
$$

Here, the upper bound on the worst case does not correspond to synchronous release of the contending tasks at the start of the interval $R_i(L_i)$, but rather to a scenario where the first job of a contending task executes as late as possible within its own period.

Note, for systems where memory accesses issued by a preempted lower priority task on the same core may be still pending after a context switch, then the analysis needs to also include such additional accesses.
(i.e. assumed in (5) to be just before its deadline, and assumed in (6) to be just before its response time) and then further jobs of that task execute as early as possible in their subsequent periods. Further, for the purposes of ensuring a correct upper bound, resource stress from each contending job is assumed to be able to occur instantaneously. This leads to a sound, but potentially somewhat pessimistic upper bound $E'_i(R_i(L_i),y)$. However, to provide a tighter bound would require highly detailed information about the timing of resource accesses within each task. Note that tasks of any priority can cause contention when executing on the other cores.

Bounding the total resource stress $E'_i(R_i(L_i),y)$ in (5) or (6) results in a context-dependent schedulability test for LO-criticality task $i$, since schedulability of the task is dependent on the parameters of the contending tasks that execute on the other cores. Using (6), the response times of the LO-criticality tasks on the same and different cores become interdependent; however, schedulability can still be determined via fixed point iteration. In this case, an outer iteration starts with $R_j(L_j) = C_j(L_j)$ for every task $j$ in the system, and repeatedly computes the response times for all tasks on all cores. This is done using the $R_i(L_i)$ values in the right hand side of (6) from the previous round, until all response times either converge, in other words are unchanged from the previous round, or one of them exceeds the associated deadline.

The correctness of the context-dependent schedulability test [30, 31], embodied in (1), (2), (3) and either (4) or (5), is sufficient to ensure compliance with the timing assurance requirement R1 for LO-criticality tasks. In fact the test provides a stronger guarantee, ensuring that LO-criticality tasks are schedulable provided that all HI-criticality tasks comply with their HI-criticality stand-alone execution times $C_i(HI)$, rather than their LO-criticality stand-alone execution times $C_i(LO)$ as required by R1. However, all tasks must still comply with their resource sensitivity $X'_i$, and resource stress $Y'_i$ parameters for the guarantee to hold.

We now show that the requirement R2 for robust timing assurance of HI-criticality tasks is also met. Each HI-criticality task $i^c$ is analysed using the fully composable context-independent schedulability test, comprising (1), (2), (3), and (4). This test effectively assumes that the contribution to the response time of task $i^c$ from each job of another task $i^c$ that executes on the same core is bounded by $B_k(L_k) = C_k(L_k) + \sum_{r \in H} (m - 1)X'_r$ (see Section 3 for details of how $B_k(L_k)$ is defined and why this is a valid bound). If a job of task $i^c$ has not completed after executing for a time $B_k(L_k)$ due to internal overrun of its own code, or extra interference resulting from a higher than expected level of resource sensitivity, then the RTOS will prevent the job of task $i^c$ from continuing to execute. Hence the RTOS prevents other tasks that execute on the same core within the response time of HI-criticality task $i^c$ from compromising its timing constraints. Since the fully composable schedulability test considers the maximum possible interference of $\sum_{r \in H} (m - 1)X'_r$ occurring during the execution of task $i^c$, then provided that the stand-alone execution time $C_i(HI)$ and the resource sensitivity parameters ($X'_i$) of task $i^c$ have been correctly upper bounded, then the task will complete its execution within its budget, irrespective of the level of resource stress emanating from potentially misbehaving tasks on other cores. Hence, task $i^c$ has robust assurance that it will meet its deadline, assuming of course that it has been deemed schedulable by the test.

Three NMC schedulability tests are evaluated in Section 5:

- **CpFPSS-m-fc-NMC**: Uses the context-independent test, comprising (1), (2), (3), and (4), for all tasks.
- **CpFPSS-m-D-NMC**: Uses the context-independent test, comprising (1), (2), (3), and (4), for HI-criticality tasks, and the deadline based context-dependent test, comprising (1), (2), (3), and (5), for LO-criticality tasks.
- **CpFPSS-m-R-NMC**: Uses the context-independent test, comprising (1), (2), (3), and (4), for HI-criticality tasks, and the response time based context-dependent test, comprising (1), (2), (3), and (6), for LO-criticality tasks, and also to compute $R_j(L_j)$ for HI-criticality tasks, used as an intermediate value in (6).

### 4.2 Static Mixed Criticality (SMC)

In this section, we extend the analysis presented in Section 4.1 to cater for the Static Mixed Criticality (SMC) scheme [11]. The only difference in the schedulability analysis for SMC compared to NMC is that with SMC, LO-criticality tasks are only required (as per R1) to be schedulable when all tasks comply with their LO-criticality stand-alone execution time parameters. As a consequence, the response time analysis is modified as follows. Equation (1) is replaced by (7), the only change being the replacement of $C_j(min(L_i, L_j))$ by $C_j(min(L_i, L_j))$. Similarly, equation (6) is replaced by (8), the only change being the replacement of $R_j(L_j)$ by $R_j(min(L_i, L_j))$.

$$R_i(L_i) = C_i(L_i) + \sum_{j \in \Gamma_x} \left[ \frac{R_i(L_i)}{T_j} \right] C_i(min(L_i, L_j)) + \sum_{r \in H} Y'_r(R_i(L_i)) \quad (7)$$

$$E'_i(R_i(L_i), y) = \sum_{j \in \Gamma_y} \left[ \frac{R_i(L_i) + R_i(min(L_i, L_j))}{T_j} \right] y'_j \quad (8)$$

The analysis for SMC thus comprises: (i) a fully composable context-independent test for HI-criticality tasks, defined by (7), (2), (3), and (4), which is effectively the same as that for NMC; (ii) a deadline based context-dependent test for LO-criticality tasks, defined by (7), (2), (3), and (5); and (iii) a response time based context-dependent test for LO-criticality tasks, defined by (7), (2), (3), and (6). Note, the latter test requires that the value of $R_j(LO)$ is similarly computed for each HI-criticality task, for use as an intermediate value in (8).

Three SMC schedulability tests are evaluated in Section 5:

- **CpFPSS-m-fc-SMC**: Uses the context-independent test, comprising (7), (2), (3), and (4), for all tasks.
- **CpFPSS-m-D-SMC**: Uses the context-independent test, comprising (7), (2), (3), and (4), for HI-criticality tasks, and the deadline based context-dependent test, comprising (7), (2), (3), and (5), for LO-criticality tasks.
- **CpFPSS-m-R-SMC**: Uses the context-independent test, comprising (7), (2), (3), and (4), for HI-criticality tasks, and
the response time based context-dependent test, comprising (7), (2), (3), and (8), for LO-criticality tasks, and to compute the LO-criticality response times for HI-criticality tasks, used as an intermediate value in (8).

### 4.3 Original Adaptive Mixed Criticality (AMC)

In this section, we extend the analysis presented in section 4.2 to cater for the original Adaptive Mixed Criticality (AMC) scheme [12]. The only difference in the schedulability analysis for AMC compared to SMC is that with AMC, LO-criticality tasks no longer release new jobs in abnormal mode. The analysis of LO-criticality response times, \( R_L(LO) \), for both HI- and LO-criticality tasks is therefore the same as for SMC. The response time \( R_L(HI) \) for a HI-criticality task \( \tau_i \) is derived as follows, using context-dependent analysis:

\[
R_L(HI) = C_L(HI) + \sum_{r \in H} (m-1)X^r_f + \sum_{j \in \Gamma_r \land j \notin \text{hpHi}(i)} \left( \frac{R_L(HI)}{T_j} \right) \left( C_j(HI) + \sum_{r \in H} (m-1)X^r_f \right) + \sum_{k \in \Gamma_r \land k \notin \text{hpL}(i)} \left( \frac{R_L(LO)}{T_j} \right) \left( C_k(LO) + \sum_{r \in H} (m-1)X^r_f \right)
\]  

(9)

where \( \text{hpHi}(i) \) is the set of HI-criticality tasks with priorities higher than that of task \( \tau_i \), and similarly \( \text{hpL}(i) \) is the set of LO-criticality tasks with priorities higher than that of task \( \tau_i \). Further, \( R_L^1(LO) \) is the context independent LO-criticality response time of task \( \tau_i \) given by:

\[
R_L^1(LO) = C_L(LO) + \sum_{r \in H} (m-1)X^r_f + \sum_{k \in \Gamma_r \land k \notin \text{hpL}(i)} \left( \frac{R_L(LO)}{T_j} \right) \left( C_k(LO) + \sum_{r \in H} (m-1)X^r_f \right)
\]  

(10)

Here, (9) and (10) represent the standard analysis equations for the AMC-rtb schedulability test [12] adapted to use inflated execution time budgets, e.g., \( C_L(HI) + \sum_{r \in H} (m-1)X^r_f \) and \( C_L(LO) + \sum_{r \in H} (m-1)X^r_f \) in place of the original execution time budgets \( C_j(HI) \) and \( C_k(LO) \).

Previous work on AMC [12] assumes that abnormal mode is entered when some job of a HI-criticality task \( \tau_k \) executes for \( C_k(LO) \) without completing. However, this criterion is not enough when cross-core contention and interference is considered, rather an inflated execution time budget of \( C_k(LO) + \sum_{r \in H} (m-1)X^r_f \) must be used instead. Given that both LO- and HI-criticality tasks may be subject to cross-core contention and interference, \( R_L^1(LO) \) represents the longest possible time interval from the release of a job of task \( \tau_i \) until either: (i) the job has completed, or (ii) abnormal mode has been entered. Hence, the interval in (9) during which LO-criticality jobs need to be considered is limited to \( R_L^1(LO) \), rather than \( R_L(LO) \). Use of the intermediate value, \( R_L^1(LO) \), is necessary to ensure compliance with requirement R2 for robust timing assurance of HI-criticality tasks, including when the behavior of other tasks is such that they do not comply with their resource sensitivity and resource stress parameters.

Three AMC schedulability tests are evaluated in Section 5:

- \( \text{CpFPPS-m-fc-AMC} \): Uses the context-independent test, comprising (9) and (10), for HI-criticality tasks, and the context-independent test, comprising (7), (2), (3), and (4), for LO-criticality tasks.
- \( \text{CpFPPS-m-D-AMC} \): Uses the context-independent test, comprising (9) and (10), for HI-criticality tasks, and the deadline based context-dependent test, comprising (7), (2), (3), and (5), for LO-criticality tasks.
- \( \text{CpFPPS-m-R-AMC} \): Uses the context-independent test, comprising (9) and (10), for HI-criticality tasks, and the response time based context-dependent test, comprising (7), (2), (3), and (8), for LO-criticality tasks, and to compute the LO-criticality response times for HI-criticality tasks, used as an intermediate value in (8).

Since the AMC scheme for partitioned multi-core systems implements independent transitions from normal to abnormal mode on each core, it is interesting to consider how the resource sensitivity and resource stress parameters of tasks impact the mode change behavior. A mode change takes place when a job of a HI-criticality task exceeds its LO-criticality budget \( B_k(LO) = C_k(LO) + \sum_{r \in H} (m-1)X^r_f \). This can only happen if the task’s stand-alone execution exceeds \( C_k(LO) \), since the additional budget terms account for the impact of the worst-case resource stress on all resources from any arbitrary tasks on the other \( m-1 \) cores. In practice, if at runtime the resource stress is below the worst case assumed, then the HI-criticality task’s stand-alone execution could exceed \( C_k(LO) \), effectively taking up the slack, without triggering a mode change. This would not however impact the schedulability of any other tasks. The resource sensitivity values, \( X^r_f \), for a HI-criticality task affect its own budget and hence indirectly affect when it may cause a mode change. By contrast, using resource stress values, \( Y^r_f \), enables less pessimistic schedulability analysis for LO-criticality tasks, however, these values do not impact the timing guarantees afforded to HI-criticality tasks.

### 4.4 Modified Adaptive Mixed Criticality (AMCR)

In this section, we adapt the analysis presented in section 4.3 to cater for the modified AMC scheme introduced by Bate et al. in [17].

The AMCR family of schemes differ from the original AMC scheme in terms of the criterion used to trigger a change to degraded mode during which jobs of LO-criticality tasks are no longer released. Two different AMCR schemes were presented in [17], here we consider the simpler scheme that returns to normal mode on an idle instant.

In the context of this work, i.e. partitioned scheduling on a multi-core system with cross-core interference modelled via resource sensitivity and resource stress, the AMCR scheme operates as follows. AMCR requires that the RTOS transitions a core to degraded mode whenever a job of a HI-criticality task \( \tau_i \) running on that core reaches, without completing its execution, an elapsed time equal to its LO-criticality response time \( R_L(LO) \), as measured from the start of the priority level-i busy period during which it was released. The RTOS transitions the core back to normal mode on an idle instant for that core. (The efficient implementation of this scheme is discussed in [17]).
Given that the LO-criticality response time $R_i(LO)$ of each HI-criticality task $\tau_i$ is derived and calculated, it follows that under AMCR, while all tasks on all cores exhibit normal behavior (i.e., comply with their LO-criticality execution time $C_j(LO)$, resource sensitivity $X'_j$, and resource stress $Y'_j$ parameters), no job of a HI-criticality task can cause a transition to degraded mode. Hence AMCR can ensure that LO-criticality tasks meet their timing assurance requirement $R1$ (see Section 1) using the same analysis as the standard AMC scheme.

The following analysis for AMCR meets the more robust timing assurance required for HI-criticality tasks.

$$R_i(HI) = C_i(HI) + \sum_{r \in H} (m - 1)X'_r + \sum_{j \in T_j \land j \notin \text{hpl}(i)} \left( \frac{R_j(HI)}{T_j} \right) \left( C_j(HI) + \sum_{r \in H} (m - 1)X'_r \right) + \sum_{k \in T_k \land k \notin \text{hpl}(i)} \left( \frac{R_k(LO)}{T_j} \right) C_k(LO) + \sum_{r \in H} (m - 1)X'_r \right) \quad (11)$$

Observe that the only difference between the analysis for HI-criticality tasks under the original AMC scheme, given by (9), and that for AMCR, given by (11), is that $R_i(LO)$, given by (10), is replaced by $R_i(HI)$, given by (7). Further, $R_i(LO)$ may be computed using context-dependent analysis, improving the precision of the schedulability test.

Under AMC, once an elapsed time of $R_i(LO)$ has passed since the start of the priority level-1 busy period in which a job of HI-criticality task $\tau_i$ was released and the job has not completed, then the RTOS ensures that degraded mode is entered. This prevents any further releases of higher priority LO-criticality tasks on that core, until after $\tau_i$ completes. Whatever caused $R_i(LO)$ to be exceeded, for example a job of a higher priority HI-criticality task $\tau_j$ on the same core exceeding its LO-criticality budget $B_j(LO) = C_j(LO) + \sum_{r \in H} (m - 1)X'_r$, or a LO-criticality task on another core misbehaving and causing more resource stress than expected, does not matter as far as the analysis is concerned. This is the case because (11) accounts for the maximum number of job releases of each LO-criticality task $\tau_k$ up to $R_i(LO)$ at their LO-criticality budget $B_k(LO) = C_k(LO) + \sum_{r \in H} (m - 1)X'_r$, and the maximum number of job releases of each HI-criticality task $\tau_j$ up to $R_i(HI)$ at their HI-criticality budget $B_j(HI) = C_j(HI) + \sum_{r \in H} (m - 1)X'_r$, hence the robust timing guarantee $R2$ (see Section 1) required by HI-criticality task $\tau_i$ holds.

Three AMCR schedulability tests are evaluated in Section 5:

- **CpFPSS-m-R-AMCR**: Uses the context-independent test, comprising (11) for HI-criticality tasks, and the context-independent test, comprising (7), (2), (3), and (4), for LO-criticality tasks and to provide the LO-criticality response times used in (11). Note, this is effectively the same schedulability test as the fully-composable test for the original AMC scheme.

- **CpFPSS-m-D-AMCR**: Uses the context-dependent test, comprising (11), for HI-criticality tasks, and the deadline based context-dependent test, comprising (7), (2), (3), and (5), for LO-criticality tasks and to provide the LO-criticality response times used in (11).

- **CpFPSS-m-R-AMCR**: Uses the context-dependent test, comprising (11), for HI-criticality tasks, and the response time based context-dependent test, comprising (7), (2), (3), and (8), for LO-criticality tasks, and to compute the LO-criticality response times used in (8) and in (11).

We note that although the value of $R_i(LO)$ used in (11) can be computed via context-dependent analysis (as in the -D and -R tests above), this does not mean that the schedulability guarantees afforded to HI-criticality tasks by (11) are dependent on the behavior of other tasks. The subtlety is that under AMCR, the RTOS enforces the transition to degraded mode at $R_i(LO)$ irrespective of the behavior or misbehavior of the other tasks, hence ensuring that the robust timing requirement $R2$ (see Section 1) required by HI-criticality tasks holds.

### 4.5 Dominance Relations

A schedulability test $S$ is said to dominate another test $Z$, for a given task model and scheduling algorithm, if every task set that is deemed schedulable according to test $S$ is also deemed schedulable by test $Z$, and there exists some task sets that are schedulable according to test $S$, but not according to test $Z$. Comparing the definitions of $E'_{k} (R_{j}(L_{j}), y)$ given by (5), (6), and (8), it is evident that each of the CpFPSS-m-R-MCS tests deems schedulable all task sets that are schedulable according to the corresponding CpFPSS-m-D-MCS test. This is the case, since in any schedulable system, the response time of a task is no greater than its deadline ($R_j(L_j) \leq D_j$), and hence the $E'_{k} (R_{j}(L_{j}), y)$ term for the former tests, given by (6) or (8), is less than or equal to the equivalent term, given by (5), for the latter tests. Further, it is easy to see that there exists task sets that are schedulable according to the former tests, but not according to the latter tests due to a larger contention contribution emanating from the larger $E'_{k} (R_{j}(L_{j}), y)$ term. Hence, each CpFPSS-m-R-MCS test dominates the corresponding CpFPSS-m-D-MCS test. Similarly, comparing the definitions of $E'_{k} (R_{j}(L_{j}), y)$ given by (5) and (4) it is evident that each of the CpFPSS-m-D-MCS tests dominates the corresponding CpFPSS-m-fc-MCS test.

Since dominance is transitive, we have: CpFPSS-m-R-MCS $\rightarrow$ CpFPSS-m-D-MCS $\rightarrow$ CpFPSS-m-fc-MCS, where $S \rightarrow Z$ indicates that test $S$ dominates test $Z$, and MCS is NMC, SMC, AMC, or AMCR.

Comparing the response time equations (1), (7), (9) and (11), it is also evident that: CpFPSS-m-X-AMCR $\rightarrow$ CpFPSS-m-X-AMC $\rightarrow$ CpFPSS-m-X-SMC $\rightarrow$ CpFPSS-m-X-NMC, where X is fc, D, or R.

### 4.6 Complexity

The standard response time analysis [8, 45] for partitioned fixed priority preemptive scheduling, not considering cross-core contention, has pseudo-polynomial complexity: $O(mn^2D_{max})$ [31], where $m$ is the number of cores, $n$ is the number of tasks on each core, and $D_{max}$ is the longest deadline of any task. The schedulability tests presented in this paper for mixed criticality systems under the MRSS task model inherit their complexity from
the schedulability tests for single criticality systems under the same model [30, 31]. Hence, the -fe, -D, and -R tests have complexity of \( O(m|H|n^2D^{\max}) \), \( O(m^2|H|n^2D^{\max}) \), and \( O(m^3|H|n^3D^{\max}) \) respectively, where \( |H| \) is the number of resources. This represents an increase in complexity of \( |H|, m|H|, \) and \( m^2|H|n \) over the equivalent tests that do not consider cross-core contention.

Given the high performance of the standard response time tests for fixed priority preemptive scheduling [32], in practice, all of the tests presented in this paper scale well to realistic system sizes. As a consequence, utilizing the highest performing -R tests is often preferable, unless a fully composable -fe test is deemed necessary due to design and development requirements. However, as shown in [30, 31], the -D tests are compatible with Audsley’s Optimal Priority Assignment algorithm [7], whereas the -R tests are not. Thus, in some cases it may be advantageous to trade off using the technically inferior -D tests in order reap the performance benefits of optimal priority assignment.

5 EVALUATION

In this section, we present an empirical evaluation of the schedulability tests introduced in Section 4 for mixed-criticality task sets executing on a multi-core system, assuming a single hardware resource shared between all cores. (Note, multiple shared hardware resources resulting in the same total interference would have the same impact on schedulability, due to the summation over resources in (1)). Experiments were performed for 2 and 4 cores.

5.1 Task Set Parameter Generation

The task set parameters used in the experiments follow the approach taken for the MRSS task model [30, 31] and for mixed criticality systems [41], with the Dirichlet-Rescale (DRS) algorithm [41] (open source Python software [40]) used to provide an unbiased distribution of utilization values that sum to the target utilization required subject to a set of individual constraints. The values selected for task resource sensitivity and task resource stress are grounded in the results obtained from the proof-of-concept case study detailed in [30, 31].

- The number of tasks per core was fixed, default \( n = 10 \). The number of HI-criticality tasks \( n(HI) \) was set to \( n \cdot CP \) where \( CP \) is the Criticality Proportion (default \( CP = 0.2 \)), with the remaining tasks of LO-criticality.
- Task utilizations were generated using the DRS algorithm. First, HI-criticality utilization values \( U_i(HI) \) were generated for the \( n(HI) \) HI-criticality tasks, such that the total HI-criticality utilization of those tasks summed to \( U(HI) = CP \cdot CF \cdot U \), where \( CF \) is the Criticality Factor (default \( CF = 2.0 \)) characterizing the multiplier between LO-criticality and HI-criticality utilization, and \( U \) is the overall target utilization required. Second, LO-criticality utilization values \( U_i(LO) \) were generated for all of the tasks, such that the total LO-criticality utilization of all tasks summed to \( U(LO) = U \). For LO-criticality tasks, each \( U_i(LO) \) value was constrained to be in the range \([0.0, 1.0]\), while for HI-criticality tasks, each \( U_i(LO) \) value was constrained to be in the range \([0.0, U_i(HI)]\).
- Task periods \( T_i \) were generated according to a log-uniform distribution [33] with a factor of 100 difference between the minimum and maximum possible period. This represents a spread of task periods from 10ms to 1 second, as found in many real-time applications.
- Task deadlines \( D_i \) were set equal to their periods \( T_i \).
- The stand-alone LO-criticality execution times for all tasks were given by \( C_i(LO) = U_i(LO) \cdot T_i \), and the stand-alone HI-criticality execution times of HI-criticality tasks by \( C_i(HI) = U_i(HI) \cdot T_i \).
- Task resource sensitivity values \( X'_i \) were determined as follows. The DRS algorithm was used to generate task resource sensitivity utilization values \( V'_j \), such that the total resource sensitivity utilization was given by the Sensitivity Factor \( SF \) (default \( SF = 0.25 \)) times the target utilization (i.e. \( \sum V'_j \cdot T_j = U \cdot SF \)), and each individual task resource sensitivity utilization was upper bounded by the corresponding task LO-criticality utilization, i.e. \( V'_j = U_j \cdot LO \). Each task resource sensitivity value was then given by \( X'_i = V'_j \cdot T_j \).
- Task resource stress values \( X''_i \) were set to a fixed proportion of the corresponding resource sensitivity value \( X'_i = X''_i \cdot RF \), where \( RF \) is the Stress Factor (default \( RF = 0.5 \)).

5.2 Experiments

The experiments considered systems with 2 or 4 cores, with a different task set, generated according to the same parameters, assigned to each core. The per core target utilization \( U \), shown on the x-axis of the graphs, was varied from 0.025 to 0.975. For each utilization value examined, 1000 task sets were generated for each core considered (100 in the case of experiments using the weighted schedulability measure [14]). In the experiments, a system was deemed schedulable if and only if the different task sets assigned to each of its cores were schedulable, i.e. if all of the tasks in the system were schedulable. The experiments investigated the performance of schedulability tests for the following schemes:

- Upper Bound High and Low (UBHL) [12]: This test checks if all of the tasks are schedulable in normal mode and if all of the HI-criticality tasks are schedulable in abnormal mode ignoring the LO-criticality tasks. This equates to the test for a hypothetical clairvoyant scheme discussed in [22]. (Black lines on the graphs).
- Modified Adaptive Mixed Criticality (AMCR) [17]: See section 4.4. (Red lines on the graphs).
- Original Adaptive Mixed Criticality (AMC) [12]: See section 4.3. (Blue lines on the graphs).
- Static Mixed Criticality (SMC) [11]: See section 4.2. (Green lines on the graphs).
- No Mixed Criticality (NMC): See section 4.1. (Orange lines on the graphs).

In each case, four variants of the tests were considered, the first three corresponding to the context-independent -fe (dotted lines) and context-dependent -D (dashed lines) and -R (solid lines)
methods of accounting for cross-core contention and interference,
and the fourth, for comparison purposes only, assuming no such
interference -no (thin dot-dash lines).

Deadline Monotonic Priority Ordering [52] was used to assign
priorities, since the context-dependent -R tests are not compatible
with Audsley’s Optimal Priority Assignment algorithm [7], as
shown in [30, 31].

5.3 Results

The figures illustrating the results are best viewed in color.

In the first experiment, we compared the performance of the
various schedulability tests using the default parameters given in
Section 5.1. The Success Ratio, i.e. the percentage of systems
generated that were deemed schedulable, is shown for each of the
tests in Figure 1 for 2 cores, and in Figure 4 for 4 cores. The
relative performance of the various tests follows the dominance
relations discussed in Section 4.5. Observe, that for equivalent
tests, overall schedulability is reduced in the case of 4 cores
compared to 2 cores. This is due to the increased cross-core
contention and interference with more cores. Note, even when no
cross-core contention is considered (i.e. the thin dot-dash lines)
then schedulability is still reduced with 4 cores. This is because the
task sets on two extra cores must also be schedulable for the
overall system to be deemed schedulable.

Considering the four mixed criticality schemes, AMCR and AMC
substantially outperform both SMC and NMC, with SMC providing
only a small improvement over NMC. The reason for this is that
the robust timing guarantee R2 required by HI-criticality tasks
means that the schedulability of those tasks in abnormal mode is
the predominant factor in overall system schedulability. AMCR and
AMC enhance the schedulability of HI-criticality tasks in abnormal
mode by suspending releases of LO-criticality jobs, hence providing
a performance gain compared to both SMC and NMC, which both
continue to release jobs of LO-criticality tasks, impinging on HI-
criticality task schedulability. The small improvement that SMC
brings over NMC derives from the fact that LO-criticality tasks do
not have to be schedulable in abnormal mode.

In the second set of experiments, we used the weighted
schedulability measure [14] to assess schedulability test
performance while varying an additional parameter. In these
experiments, the other parameters were set to the default values
given in Section 5.1. In all of the weighted schedulability
experiments the relative performance of the different tests follows
the pattern illustrated in the first experiment, as dictated by the
dominance relationships.

The results of varying the Sensitivity Factor SF, from 0.05 to
0.95 in steps of 0.05, are shown in Figure 2. Recall that the
Sensitivity Factor determines the ratio of the total resource
sensitivity utilization to the total LO-criticality task utilization. As
expected, increasing the Sensitivity Factor, and hence the amount
of interference that tasks can be subject to due to cross-core
contention, results in a rapid decline in the weighted
schedulability measure for all of the tests that take cross-core
contention into account.

The results of varying the Stress Factor RS, from 0 to 1.8 in
steps of 0.1, are shown in Figure 5. Recall that the Stress Factor
determines the ratio of the resource stress for each task to its
resource sensitivity. Here, interference effective saturates once the
Stress Factor reaches 1.0. By then, the total resource stress \( E_i(t, y) \),
given by (5) or (6), emanating from each additional core tends to
exceed the total resource sensitivity \( S_i(t, x) \), given by (3). Hence,
the context-dependent -R and -D tests reduce to exactly the same
performance as the context-independent -fc test.

Observe that in Figure 2, the -R, -D, and -fc tests have very
similar performance when combined with SMC or NMC. The
reason for this is that since LO-criticality jobs continue to be
released in abnormal mode, overall schedulability depends
predominantly on the schedulability of the HI-criticality tasks in
that mode, hence the form of analysis used for LO-criticality tasks
leaves little bearing on the overall results. This is not the case with
AMC, AMCR, or the UBHL bound, where modest gains are
apparent when using the -R or -D tests for all tasks in normal
mode. The same behavior is evident in Figure 5, however, in that
case as the resource Stress Factor is reduced, the impact of
contention on LO-criticality tasks decreases, and the performance
advantage obtained using the context-dependent -R and -D tests
increases.

In Figure 5, when the resource Stress Factor is zero, the UBHL
bound combined with the context-dependent -R and -D tests
provides almost the same performance as the no contention case
(-no). This is because the HI-criticality tasks considered alone are
easily schedulable in abnormal mode, and hence system
schedulability according to the UBHL bound is predominantly
influenced by schedulability in normal mode. This is not the case
with AMC, since although LO-criticality tasks are prevented from
releasing further jobs in abnormal mode, job releases prior to that
point still impinge upon HI-criticality task schedulability in
abnormal mode. AMCR shows a significant advantage over AMC
when the resource Stress Factor is small. This is because the
difference between \( R_i(LO) \) used in (11) and \( R_i'(LO) \) used in (10)
is amplified in this case, resulting in fewer jobs of LO-criticality tasks
impinging upon HI-criticality task schedulability under AMC.

The results of varying the Criticality Proportion CP, from 0.0 to
1.0 in steps of 0.1, are shown in Figure 3. With no HI-criticality tasks,
UBHL, AMCR, AMC, SMC, and NMC all reduce to the same
(-no, -R, -D, -fc) schedulability test and hence the same performance.
At the other extreme, when there are only HI-criticality tasks and
since these tasks require the robust timing guarantees afforded by
a context-independent test, the set of -R, -D, and -fc tests for each
scheme all reduce to the same performance. Additionally, since
there are only HI-criticality tasks, all of the schemes reduce to exactly
the same schedulability test, and so all of the lines for tests
where cross-core contention is considered meet at a single point.

In Figure 3, the performance of the SMC and NMC tests
improves as a final HI-criticality task is added and there are no
longer any LO-criticality tasks present. This stems from the way in
which HI- and LO-criticality utilization values are generated. The
total HI-criticality utilization of the HI-criticality tasks is precisely
controlled by the task set generation process, as is the total
LO-criticality utilization over all of the tasks. However, the
LO-criticality utilization of a single LO-criticality task is not. With
SMC and NMC, schedulability effectively depends on the total
utilization in abnormal mode, i.e. the sum of the LO-criticality
Figure 1: Success Ratio: Varying task set utilization, 2 cores.

Figure 2: Weighted Schedulability: Varying Resource Sensitivity, 2 cores.

Figure 3: Weighted Schedulability: Varying the Criticality Proportion, 2 cores.

Figure 4: Success Ratio: Varying task set utilization, 4 cores.

Figure 5: Weighted Schedulability: Varying Resource Stress, 2 cores.

Figure 6: Weighted Schedulability: Varying the Criticality Factor, 2 cores.
utilization of the LO-criticality tasks and the HI-criticality utilization of the HI-criticality tasks, and this can be worse when almost but not all of the tasks are of HI-criticality.

In Figure 3 with the UBHL bound, weighted schedulability remains roughly constant until the proportion of HI-criticality tasks exceeds 50%. This is because the default Criticality Factor is 2.0, hence when more than 50% of the tasks are HI-criticality, the increased utilization of HI-criticality tasks in abnormal mode becomes the dominant factor influencing schedulability. Before then, the total LO-criticality utilization is the dominant factor and that does not vary with the Criticality Proportion.

The results of varying the Criticality Factor CF, from 1.0 to 4.0 in steps of 0.2, are shown in Figure 6. Observe that schedulability according to AMC, AMC, SMC, and NMC, progressively decreases as the Criticality Factor increases, so increasing the utilization of HI-criticality tasks in abnormal mode. This trend is not evident with the UBHL bound, as the default Criticality Proportion of HI-criticality tasks is 0.2, and hence even with \( CF = 4.0 \) the increased utilization of HI-criticality tasks in abnormal mode is still not the dominant factor influencing system schedulability; rather, the total LO-criticality utilization is the dominant factor and that does not vary with the Criticality Factor.

Overall, the results for the modified AMCR scheme provide a useful improvement over their counterparts for the original AMC scheme, shifting the schedulability guarantees closer to the hypothetical UBHL upper bound that ignores the effects of the mode change transition. As expected, both AMCR and AMC significantly outperform SMC and NMC.

6 CONCLUSIONS

The main contributions of this paper are as follows: (i) The integration of mixed criticality concepts into the MRSS [30, 31] multi-core system model that characterizes cross-core contention and interference via task resource stress and sensitivity. (ii) Consideration of the different levels of assurance needed in mixed criticality systems, specifically the need to provide HI-criticality tasks with robust timing guarantees. (iii) Derivation of schedulability analysis for four mixed criticality scheduling schemes (NMC, SMC, AMC, and AMCR), accounting for resource contention and interference on a partitioned multi-core processor, providing appropriate timing guarantees for both HI- and LO-criticality tasks.

The key observations are as follows. Firstly, as expected, the significant performance advantages that the AMCR and AMC schemes hold over the simple SMC and NMC schemes are retained when cross-core contention and interference is included via a mixed criticality multi-core resource stress and sensitivity model.

Secondly, utilizing more precise context-dependent schedulability tests to bound the interference on LO-criticality tasks results in useful performance improvements, while still ensuring that HI-criticality tasks are provided with robust timing guarantees.

Finally, it is interesting to note that while the AMCR and AMC schemes have identical performance in terms of schedulability when cross-core contention is not considered, once such interference is included, then the AMCR scheme dominates AMC.

ACKNOWLEDGMENTS

This research was funded in part by Innova UK HICLASS project (113213) EPSRC Research Data Management: No new primary data was created during this study.

REFERENCES


