Extending Mixed Criticality Scheduling

Thomas D Fleming

MSc By Research

The University Of York

Computer Science

September 30th 2013

Abstract

The capability of hardware is constantly developing in capacity, speed and efficiency. This development has sparked industrial and academic interest in how best to utilise the increased capability. It is now possible to integrate many systems that in the past might have existed as different nodes, into the one consolidated architecture. This desire to centralise functionality leads to the potential of a system that contains software components of differing levels of importance or criticality. Such Mixed Criticality Systems pose a challenging problem with regard to analysis and certification. Much work has been undertaken investigating the use of Fixed Priority scheduling for Mixed Criticality Systems, a notable scheme, known as Adaptive Mixed Criticality (AMC), provides significant advances in schedulability over prior approaches. The focus of the work on AMC revolves around just two levels of criticality. In this work we develop extensions to consider greater than two levels of criticality, for both forms of AMC analysis (AMCrtb & AMCmax) and consider the implication of applying these extended approaches. Alongside this we adapt some of the schemes developed prior to AMC in order to assess their relative effectiveness. We also review and further develop Period Transformation for use with Mixed Criticality Systems. Finally we provide a set of evaluations to illustrate the results. We conclude that AMC maintains its effectiveness over many criticality levels and remains an effective scheme. Of the two forms of analysis, AMCrtb is the most practical as the schedulability improvement gained by using AMCmax is slight and the increase in computation required is extreme. When considering an arbitrary number of criticality levels AMCrtb is a dependable, comprehensive scheme.

Contents

1 Introduction **8**

2 Literature Review **10**

2.1 The System Model 11
2.2 Analysis 13
2.2.1 Priority Assignment 13
2.2.2 Static, Response Time Approaches 14
2.2.3 Period Transformation 16
2.2.4 Dynamic Scheduling 17
2.2.5 Other Approaches 18
2.3 Comparative Work 19
2.4 Variations On The Analysis 19
2.4.1 Multi-Core/Processor 19
2.4.2 Communication & Shared Resources 20
2.5 Implementations/Frameworks 22
2.6 Summary 23

3 The AMCrtb Approach **24**

3.1 Dual Criticality 24
3.2 Many Criticality Levels 26
3.2.1 Stage One 26
3.2.2 Stage Two 27
3.3 Adapting Additional Approaches 29
3.3.1 CrMPO 29
3.3.2 SMC-NO 30
3.3.3 SMC 30
3.4 Some Illustrative Results 31

3.5 Summary . . . 32

4 The AMCmax Approach **33**

4.1 Original Analysis . . . 33

4.2 Extending the Analysis . . . 36

4.2.1 Adding a Medium Level . . . 37

4.2.2 To n Criticality Levels . . . 42

4.3 Some Illustrative Results . . . 45

4.4 Summary . . . 47

5 Period Transformation **49**

5.1 Standard Period Transformation . . . 49

5.2 Vestal's Period Transformation . . . 51

5.3 Improving the Analysis . . . 53

5.4 Greater Than Two Criticality Levels . . . 54

5.5 Some Illustrative Results . . . 56

5.6 Summary . . . 58

6 Evaluation **59**

6.1 Assumptions . . . 59

6.2 Task Generation . . . 60

6.3 Criticality Dependent Utilisation . . . 61

6.4 Results . . . 61

6.4.1 AMCrtb . . . 61

6.4.2 AMCmax . . . 64

6.4.3 Period Transformation . . . 66

6.5 Overall comparison . . . 69

6.6 Discussion . . . 71

6.7 Summary . . . 72

7 Conclusions **73**

List of Figures

3.1 Bounded interference example 28
3.2 AMCrtb, SMC and CrMPO with 2 criticality levels. 31
3.3 AMCrtb, SMC and CrMPO with 5 criticality levels. 32

4.1 Example AMCmax criticality change. 34
4.2 AMCmax with three criticality levels. 37
4.3 A criticality change showing execution of Task 1 in two modes. . 40
4.4 The structure of execution for AMCmax over 3 criticality levels. 42
4.5 A system with modes A and B. 44
4.6 The system with modes A and B with an additional level, C added. 45
4.7 AMC, AMCmax and SMC . 2 Criticality Levels 46
4.8 AMC, AMCmax and SMC . 5 Criticality Levels 47

5.1 $C_j(HI)/n$ slices completing within $C_j(LO)$ 52
5.2 An example showing transformed executions constituting a complete $C_j(LO)$ 53
5.3 Period Transformation, AMCrtb and AMCmax: 2 Criticality Levels 57
5.4 Period Transformation, AMCrtb and AMCmax: 5 Criticality Levels 57

6.1 UUniFast [15] 60
6.2 AMCrtb, SMC, UB & CrMPO: 2 Criticality Levels 62
6.3 AMCrtb, SMC, UB & CrMPO: 3 Criticality Levels 62
6.4 AMCrtb, SMC, UB & CrMPO: 4 Criticality Levels 63
6.5 AMCrtb, SMC, UB & CrMPO: 5 Criticality Levels 63
6.6 AMCmax, AMCrtb, SMC, UB & CrMPO: 2 Criticality Levels . . 64
6.7 AMCmax, AMCrtb, SMC, UB & CrMPO: 3 Criticality Levels . . 65
6.8 AMCmax, AMCrtb, SMC, UB & CrMPO: 4 Criticality Levels . . 65
6.9 AMCmax, AMCrtb, SMC, UB & CrMPO: 5 Criticality Levels . . 66

6.10 Period Transformation, AMCrtb and CrMPO: 2 Criticality Levels 67
6.11 Period Transformation, AMCrtb and CrMPO: 3 Criticality Levels 67
6.12 Period Transformation, AMCrtb and CrMPO: 4 Criticality Levels 68
6.13 Period Transformation, AMCrtb and CrMPO: 5 Criticality Levels 68
6.14 AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 2 Criticality Levels 69
6.15 AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 3 Criticality Levels 70
6.16 AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 4 Criticality Levels 70
6.17 AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 5 Criticality Levels 71

List of Tables

2.1 An Example of the Sub-Optimality of RM & DM Assignment. . 13

5.1 Period Transformation Example 50
5.2 Untransformed Mixed Criticality Example 51
5.3 Transformed Mixed Criticality Example 52
5.4 3 Criticality Level PT Example, Untransformed 55
5.5 3 Criticality Level PT Example, Transformed 56

Acknowledgements

I would like to thank my supervisor, Alan Burns, for his help and guidance throughout. I would also like to thank Rob Davis for his useful discussions on Period Transformation. Finally I would like to thank my family, without whom, none of this would have been possible.

Declaration

I declare that the work presented in this document is my own, unless explicitly indicated. This research was undertaken by myself, under the supervision of Professor Alan Burns between February 2013 and September 2013 at the University Of York. External sources are acknowledged via the use of bibliographic references.

Chapter 1

Introduction

Alongside the development of faster and more efficient hardware there is an increasing demand to support systems of a progressively heterogeneous nature. In the past system functionality might have been spread over many nodes, however advances in single and multi-core architectures have paved the way for the consolidation of this functionality. These advances have begun to push industrial and academic interest towards developing systems to facilitate a wider range of functionality. Many key industrial sectors, from automotive to aerospace, have recognised the advantages and, perhaps, the necessity, of moving towards more centralised architectures.

Such systems are likely to include components of differing level of importance, or Criticality. Components might be safety critical or simply have a level of desired performance. We define a Mixed Criticality System (MCS) as, a system that incorporates two or more different levels of criticality. Safety Critical elements are typically subject to certification by a relevant Certification Authority (CA), this requires the specific (Safety Critical) components to adhere to the, often highly pessimistic, analysis mandated by the CA.

This highlights one of, if not the key challenge in the field of Mixed Criticality systems; balancing the need to satisfy the CA and provide suitable guarantees of safety, whilst ensuring as high as possible resource utilisation. One component might not require any certification, and thus the system designer's performance predictions provide an adequate basis for analysis, however another element might require certification, and thus pessimistic techniques are used to gauge its' performance.

There are clear advantages to the use of Mixed Criticality Systems, these

include (but are not limited to): increased energy efficiency, reduced cost and a smaller physical footprint. These requirements are most apparent in safety critical industries, such as the automotive or aerospace domains. Systems in these areas are required to deal with increasingly more complex mission critical or even general purpose applications such as image capture and recognition. Sitting these applications beside high integrity, safety critical functionality is challenging.

A key stepping stone in supporting such systems is to consider the scheduling of its tasks. In this work we consider several new and old, uni-processor, fixed priority scheduling policies and assess their effectiveness. Much of the previous work has limited its analysis to consider only two levels of criticality (importance) HI and LO, work such as [41] and [10]. This work seeks to extend these schemes to allow them to deal with 2 to n possible criticality levels and to investigate what performance impact this might have.

The document is structured as follows; Chapter 2 contains a review of the current MCS literature, Chapter 3 considers AMCrtb [10] and its extension, Chapter 4 considers AMCmax [10] and its extension, Chapter 5 Considers Period Transformation for Mixed Criticality Systems, Chapter 6 provides a detailed Evaluation and Chapter 7 ends the document with some concluding remarks.

Chapter 2

Literature Review

In 2007 Vestal [41] published what is widely considered to be the initial work on the verification of Mixed Criticality Systems. In this paper Vestal identifies the key MCS problem, verification vs utilisation and the need for criticality-aware, graceful-degradation. Graceful degredation implies that a system should ensure it provides sufficient execution budget for each task subject to the bounds set by their criticality level. If a task overruns these bounds, it should be dealt with in some way that allows higher criticality tasks to continue to work within their timing requirements. Two papers in 2008 built upon Vestal's work; Baruah and Vestal [14] refined the initial model and noted that EDF is not optimal for Mixed Criticality Systems and Huber et al. [24] considered MC systems from a multi-processor perspective.

The work that followed Vestal's seminal paper focused primarily upon uni-processor Mixed Criticality systems and their analysis, with a view to both dynamic and static scheduling approaches. More recently a larger body of work has formed investigating MC systems on multi-processor/core platforms. This move to more advance platforms has been fuelled by industrial pressure to utilise new and powerful hardware available in multi-processor form. A rich body of work continues to develop for both uni-processor and multi-processor platforms.

The following Chapter provides a review of the work on Mixed Criticality Systems. Section 2.1 describes the system model used in the review. Section 2.2 considers current analytical work including priority assignment, static and dynamic analysis. Section 2.3 briefly considers some comparative works. Section 2.4 considers variations on the analysis presented in section 2.2. Section 2.5 looks at some more practical approaches and Section 2.6 provides a summary.

2.1 The System Model

Although there are variations of the Mixed Criticality Model defined by Vestal [41], the model described below provides a good basis for this review and is commonly used in much of the literature.

A system constitutes a finite set of components K. Each of these components is assigned a criticality level, L (designated by the system designer) and consists of a finite set of sporadic tasks. Each task, τ_i, is defined as $\tau_i = \{C_i, T_i, D_i, L_i\}$ where C_i is the Worst Case Execution Time (WCET) time, T_i is the period (minimum inter-arrival time), D_i is the deadline and L_i is the criticality level. Each task gives rise to an unbounded series of jobs.

Vestal [41] makes an important observation regarding the relationship between the criticality level of a task and its computation time. As the criticality level increases, so does the computation time. This is due to the increased level of pessimism in the analysis of higher criticality tasks. A safety critical task might have a criticality level of $L1$ (Where $L1 > L2$), the task might also be verified to criticality level $L2$, its $L2$ WCET would be less than or equal to its $L1$ WCET. Variation in the frequency of the minimum inter-arrival time or period of each task has also been considered. Burns and Baruah [16] note that this is less likely, but it could be due to the certification of a task requiring a more pessimistic (therefore more frequent) inter-arrival time at a higher criticality level. Several other papers [6, 9, 11] also consider this potential variation.

The observations described above allow us to modify our definition of a task, $\tau_i = \{\overrightarrow{C}, \overrightarrow{T}, D, L\}$, where $\overrightarrow{C}$ and $\overrightarrow{T}$ are vectors, one value for each criticality level. These vectors conform to the following, for any two criticality levels $L1$ & $L2$:

$$L1 > L2 \implies C(L1) \geq C(L2)$$

$$L1 > L2 \implies T(L1) \leq T(L2)$$

It is also possible to state a similar constraint for a criticality dependent deadline (although this has been given some focus [9], this is still a subject for future study).

$$L1 > L2 \implies D(L1) \geq D(L2)$$

A shorter, $L2$ criticality deadline might be one desired for high quality of service

whiles its higher criticality $L1$ deadline is safety critical.

The last point to address is the concept of criticality modes, a system might be defined to execute in a number of different modes depending upon the number of specified criticality levels. Such systems always begin their execution in the lowest criticality level ($L2$), a mode change occurs, $L2 \longrightarrow L1$, when a task at level $L2$ executes to its $L2$ WCET, $C_i(L2)$, without signalling completion.

Both the observations about period and computation time, and the idea of a criticality based mode change stem from the desire to satisfy two conflicting properties: Static verification (certification) and efficient resource utilisation. The Certification Authority will consider only the verification of tasks that are safety critical, as long as a suitable level of isolation is maintained they are not concerned with the rest of the system. However using just the pessimistic response time predictions provided by the CA would lead to a very inefficient system. Instead we use criticality levels, this allows the high criticality tasks (those verified by the CA) to, if necessary, execute for their pessimistic execution times. In this case the lower criticality tasks would be managed in some way as to prevent them from interfering with the execution to the high criticality tasks.

The model assumes that, in reality, the system designer's predictions for task response times are likely to be accurate, thus the system will run comfortably in the lowest criticality level. The ability to perform a mode change provides the reassurance required for safety critical aspects of the system. It is worth noting that due to this assumption about the correctness of the systems designer's predictions, the majority of the MC work considers only an increase in the criticality level of a system. The potential return to lower criticality levels has not yet been addressed in any detail.

It is worth illustrating this functionality by means of a naive, but commonly used example. Consider the case of an Unmanned Aerial Vehicle (UAV). In order to fly in civilian airspace the flight control software must be certified, as it is safety critical. The reconnaissance software, required for the successful operation of the UAV is considered mission critical and is not subject to certification. The system designer estimates (reliably) that the mission critical elements require 0.45 of a processor. The Certification Authority analyses (pessimistically) the safety critical element and determines that 0.9 of a processor is required. On the face of it, with a utilisation of 1.35 it seems that we need two processors. However the system designer estimates the utilisation for the safety critical element to be only 0.5. Therefore in the low criticality mode the

utilisation is 0.95[1], only one processor is required. If a criticality change occurs 0.9 of the processor is given to the safety critical software.

2.2 Analysis

2.2.1 Priority Assignment

Mixed Criticality priority assignment was initially considered by Vestal in his 2007 paper [41]. He observed that Rate Monotonic and Deadline Monotonic priority assignments were not optimal for use in Mixed Criticality Systems. Much of the mixed criticality literature uses the notation LO and HI to denote the criticality levels in a dual criticality system. This notation is used, where appropriate, throughout this review. Consider the task set in Table 1:

τ	$C_i(LO)$	$C_i(HI)$	T_i	D_i	L
1	1.5	-	2.5	2.5	LO
2	1	3	4	4	HI

Table 2.1: An Example of the Sub-Optimality of RM & DM Assignment.

Under Deadline Monotonic (and Rate Monotonic) assignment, τ_1 would be given the highest priority. Execution in the LO mode is acceptable as $1.5 + 1 \leq D_2$. However during a criticality change there is a problem. If both τ_2 & τ_2 are released at the same instant, τ_1 executes to 1.5 then τ_2 executes to 1. However if τ_2 does not signal completion a change of criticality level from $LO \longrightarrow HI$ occurs. In this situation the response time of τ_2 would be 4.5[2] which is greater than D_2. Therefore the task-set is not schedulable with τ_1 at the highest priority. If τ_2 were given the highest priority, the LO mode would execute the same as before $1 + 1.5$ which meets both deadlines. If τ_2 executes to its LO budget without signalling completion, a change from $LO \longrightarrow HI$ occurs and τ_2 is able to execute to its HI WCET value, 3, and τ_1 is suspended. In this way Deadline and Rate Monotonic algorithms are not optimal for use in MC systems.

As shown above, this is due to their inability to deal with multiple execution time values for each task. Vestal did note that Audsley's [2] optimal priority assignment algorithm is applicable. Vestal suggests that this algorithm can be adjusted to utilise Mixed Criticality Scheduling analysis in order to find an optimal priority assignment. Audsley's algorithm seeks to assign a task to the

[1] 0.45+0.5
[2] $1.5 + 3$

lowest priority, when a task is assigned this priority it is removed from the system and the test is run again until all tasks are granted a priority or the search fails and is unable to assign a complete set of priorities. In this case we can consider the system unschedulable. Vestal utilises a metric known as the Criticality Scaling Factor [29, 41] the largest value by which all execution times can be simultaneously multiplied while preserving feasibility. If when looking for a task to assign to a priority, two are feasible, the task with the greatest Criticality Scaling Factor will be assigned that priority. Audsley's algorithm has the advantage of being able to determine if an optimal assignment exists within n(n+1)/2 steps instead of an exhaustive search of all possible priority assignments.

In 2008 Baruah and Vestal [14] generalised the priority assignment algorithm by assessing both EDF and FP (Fixed Priority) assignment. This assessment coupled with the use of sporadic task systems aided the development of the Augmented Audsley algorithm. Dorin et al. [18] provided a proof of the optimality of Audsley's approach for Mixed Criticality Systems.

2.2.2 Static, Response Time Approaches

After Vestal's 2007 paper [41], much work went into the static analysis of Mixed Criticality systems. When considering this analysis, we must again reflect upon the conflicting aims of isolation and efficiency. Until recently general practice has focused upon isolation of tasks for safety, system resources are often inefficiently used. Techniques such as space partitioning, exclusive resource access and time partitioning provide poor resource utilisation. A further approach is known as partitioned criticality scheduling [10] (criticality monotonic), this assigns priorities according to criticality level, all tasks of a higher criticality will have higher priorities than those of lower criticality. The latter approach removes the risk of criticality inversion, where a task of higher priority but lower criticality interferes with the execution of a higher criticality task. However this approach is extremely inefficient.

The majority of the static MC analysis below is based around standard response time techniques [3].

$$R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.1}$$

Where the response time of τ_i, R_i is solved recursively based upon interference

suffered from the set of tasks with a higher priority than τ_i.

In 2007 Vestal [41] proposed an approach to mixed criticality scheduling. Vestal's approach made an important step forward by allowing for interleaved task priorities between criticality levels. However the analysis is based on the assumption that all tasks are verified to the highest criticality level in the system. This is prohibitively expensive and provides far from optimal resource utilisation.

Own Criticality Based Priority is a scheme first suggested by Baruah et al. [13]. Essentially it is an extension of Audsley's algorithm [2] to allow for mixed criticality systems. It provides both a priority ordering and a sufficient schedulability test. The algorithm seeks to find the job j_i that might be assigned the lowest priority if all other jobs execute for their $C_i(L_i)$. In this way the priority ordering is based upon each job's criticality level. While this does provide an improvement over Vestal's [41] original analysis, as we consider jobs at their criticality level not at the highest, it still provides far from efficient utilisation.

Static Mixed Criticality is a continuation of the Own Criticality Based Priority scheme, extended to utilise run-time monitoring. If the system detects that a low criticality job, j_i, is overrunning its allocated C_i it is prevented from executing further and is suspended. If an overrun is detected for a high criticality job, j_k, the system undergoes a criticality level change to the high mode. Jobs of τ_k (and all other high criticality tasks) are given their high criticality execution budgets, $C_k(HI)$. This use of run-time monitoring provides schedulability analysis far superior to that initially developed by Vestal [41] which we can re-name SMC-NO [10], SMC with no run-time monitoring.

Baruah et al. [10] utilise run-time monitoring to detect jobs that reach their maximum execution time ($C_i(LO)$) but do not signal completion. This monitoring allows for a criticality change to occur, $LO \longrightarrow HI$(for the purpose of this explanation Baruah et al. restrict themselves to two criticality levels). This functionality was used to derive a new algorithm, Adaptive Mixed Criticality (AMC). The runtime behaviour of AMC is as follows. All jobs in the system begin execution in their LO criticality mode, if j_i executes for more than its allocated $C_i(LO)$ a criticality change occurs. All jobs of criticality LO are suspended indefinitely. Jobs with a HI criticality level continue to execute, but this time to their $C_i(HI)$ budgets. They present two analytical methods, method 1 or AMCrtb is a simpler response time based approach that assesses both the schedulability of the tasks in each mode of the system and the schedulability of

any Criticality changes. The more precise Method 2, AMCmax, only considers the finite set of possible points (s) at which a criticality change could occur, from these points it is possible to perform analysis to discover the worst case point of s and thus determine if the task set in question is schedulable. We cover the analysis of both AMCrtb and AMCmax in Chapters 4 and 5.

AMCrtb has been extended in [43] to utilise preemption thresholds and Baruah and Chattopadhyay [11] consider SMC and AMC when task periods alter according to their criticality level rather than WCETs.

Zero Slack Scheduling is a further technique for scheduling Mixed Criticality Systems initially suggested by Niz et al. [34]. They work on the basis that criticality inversion (lower criticality tasks with higher priorities interfering with higher criticality tasks at lower priorities) only matters during overload conditions, similar to the idea of a criticality change [10]. They define two modes, N mode (normal) and C mode (critical). They calculate the last possible time at which a task (of high criticality) must begin execution in order to meet its deadline, if this is not met, the system moves to C mode. In C mode lower criticality tasks are prevented from interfering to ensure the high criticality task completes by its deadline. During normal execution (N mode) the system allocates the slack, before the zero slack instant, to lower criticality tasks. Huang et al. [23] expand further by identifying a situation where a low criticality task might miss its deadline and affect the higher criticality tasks, to solve this they present a priority demotion technique. Alongside this addition an updated analysis is presented.

2.2.3 Period Transformation

Period Transformation [39] is also applicable to MC systems. The idea is that a task is split into smaller component parts by some factor n. So τ_i's T_i would be come T_i/n and C_i would become C_i/n. The transformed task set might then be assigned some more optimal priority assignment (Rate Monotonic etc.). If all tasks are transformed, in a non MC system, any task set with a utilisation less than or equal to 1 will be schedulable as the task set will be harmonic[3]. However period transformation suffers from excessive overheads involved with splitting tasks and managing their executions alongside an increased number of context switches. These excessive overheads might explain why industrial uptake of the scheme has not occurred, as such Period Transformation presents

[3]Where all task periods are integer multiples

very attractive theoretical properties but suffers from many practical issues.

Vestal proposes a PT technique applicable to MC systems, tasks are transformed if their period (for tasks where $T = D$) is less than the shortest period of a LO criticality task. The purpose of this technique is to provide a criticality monotonic ordering. There are two groups of tasks which might not be transformed.

- LO criticality tasks will not be transformed as they are to be given the lowest priorities.
- HI criticality tasks with a period less than that of the lowest LO task. These tasks are not transformed as they might already be assigned a higher priority under criticality monotonic assignment.

Due to the fact that not all tasks are transformed, the bound stating that a task set is schedulable if the utilisation is less than 1 is no longer applicable as a harmonic task set is not created. Extending MC Period Transformation to more than 2 criticality levels has yet to be addressed and may present additional challenges. We cover the analysis for Period Transformation and its extensions, in detail, in Chapter 6.

2.2.4 Dynamic Scheduling

The use of EDF scheduling in Mixed Criticality systems was initially considered by Baruah and Vestal [14]. They note that due to the nature of EDF, any task might be prioritised over another, therefore all tasks must be verified to the highest level of criticality. This leads to the key point that standard EDF is not optimal when considering systems with multiple worst case execution times/levels of criticality. Baruah et al. [8] introduce EDF-VD, Earliest Deadline First with Virtual Deadlines, an EDF scheduling approach which uses modified, artificial deadlines to ensure schedulability of multiple levels of criticality. Further work on EDF-VD [7] proves optimality via the use of the speed-up factor [25].

Baruah et al. [13] describe the speed-up factor metric as: “the minimum multiplicative factor by which processors must be made faster in order to compensate for the inexactness of the test”. In other words it is a metric used to gauge how much faster a processor would need to be to make a task set schedulable under a particular algorithm. This is particularly useful when considering

the effectiveness of both Fixed Priority and Dynamic scheduling algorithms as they are not directly comparable.

Ekberg and Yi [19] expand upon previous work EDF-VD [8, 7] by allowing EDF to use different artificial deadlines for tasks depending on the current criticality mode. They employ demand bound functions, DBF_{LO} & DBF_{HI} to assess the maximum execution demand in any given time interval. The same principles of EDF-VD hold here, the demand bound functions are used to tune the deadlines of tasks to achieve better utilisation and in turn schedulability.

Park and Kim [35] derive an algorithm known as $CBEDF$ (Criticality Based Earliest Deadline First), this algorithm uses slack reclamation to provide efficient scheduling. Two types of slack are defined;

- Remaining Slack: *Spare time between $C_i(LO)$ & $C_i(HI)$ if a high criticality job completes early.*
- Empty Slack: *If all $C_i(HI)$ tasks use their allocated time, any additional slack is empty slack.*

CBEDF allows both forms of slack to be allocated to C_iLO tasks in such a way that it does not interfere with those tasks of higher criticality. Finally their experimentation shows CBEDF's dominance over OCBP scheduling.

PLRS [21] is a dynamic algorithm which draws its inspiration from both static and dynamic scheduling. Pre-runtime PLRS calculates job priorities eventually creating a priority plan taking into account multiple criticality levels. This plan is then used at runtime to assign priorities. This algorithm is a hybrid using static, offline analysis to produce a plan, but dynamic assignment at run-time.

2.2.5 Other Approaches

Baruah and Fohler [12] explore the use of Time Triggered (TT) scheduling in a Mixed Criticality context. Due to the nature of the complete determinism provided by a time triggered system it is widely used and favoured by Certification Authorities. They show that achieving high utilisation and meeting certification requirements is difficult with strict TT scheduling. However they show that such systems can be extended to include mode changes, this requires multiple dispatch tables, one for each level of criticality in the system. Steiner [40] also touched on the TT approach, this time looking at the incorporation of TT network traffic with unsynchronised traffic.

Lackorzynski et al. [27] explore the potential link between Mixed Criticality and Hierarchical scheduling. Hierarchical scheduling, usually associated with virtualisation, could help provide strict isolation between tasks of differing criticality levels. Each level might run on a different guest OS, or High criticality tasks on one, all other criticalities on another. They show that current hierarchical scheduling techniques are not flexible enough to deal with the challenge of MC systems. However Lackorzynski et al. [27] propose alterations to deal with MC systems, whereby each guest OS is assigned a budget for each criticality level.

2.3 Comparative Work

Comparing different algorithms is not always straightforward. Baruah et al. [10] provide an effective comparison of several variations on FP Mixed Criticality scheduling. They perform experiments using large sets of randomly generated tasks, the key result is the relationship between task set utilisation and the percentage of schedulable task sets. Kelly et al. [26] provide an experimental analysis which compares Audsley's optimal priority assignment with Rate Monotonic priority assignment.

Haung et al. [23] present an evaluation of Response Time, Period Transformation and Zero-Slack scheduling based approaches upon harmonic and non-harmonic task sets of varying sizes. The evaluation also includes overheads, allowing for a better comparison with the theoretically superior Period Transformation. Their work does not explicitly state the number of criticality levels considered. In one example it appears that 3 criticality levels are used. However, for the most part, only dual criticality systems are considered, or, the number of criticality levels is left unspecified. This work represents one of the most thorough comparative evaluations of common Mixed Criticality approaches.

Further experimental analysis can be found [4, 22, 27, 28].

2.4 Variations On The Analysis

2.4.1 Multi-Core/Processor

Multi-core support of Mixed Criticality was initially considered by Anderson et al. [1]. They note two techniques for multi-core scheduling;

- Partitioned Scheduling: *One dispatching table for each processor*
- Global Scheduling: *One global dispatching table for many processors*

Current practice employs the use of partitioned scheduling for Hard Real-Time systems and Global scheduling for Soft Real-Time systems. They propose an innovative scheme utilising Containers (Servers) to provide suitable isolation between criticality levels whilst Global scheduling provides good processor utilisation. However their work is limited to Harmonic Task Sets. Mollison et al. [31] extend the work by Anderson et al. [1] by considering the use of Hierarchical scheduling. They develop a scheme which defines 5 criticality levels A to E, each level is scheduled within its own container. Level A Tasks are scheduled via a cyclic executive, level B via EDF, levels C and D via G-EDF (Global EDF) and Level E via a best effort scheme. This complex scheme does help provide isolation, however the runtime overheads are unclear. Herman et al. [22] provide an examination of the issues surrounding fully implementing the scheme described above. They show that implementation is possible and that overheads can be kept to within reasonable bounds.

Li and Baruah [30] also explore the issue of scheduling MC systems on multi-processors/cores. Their work is a generalisation of the algorithm $fpEDF$ (Fixed Priority Earliest Deadline First) [5], $fpEDF$ is an algorithm for scheduling normal (non-mixed criticality) tasks on a multi-processor system. They continue to use a previously developed algorithm $EDF-VD$ [7] to develop a scheduling technique applicable for multi-processor MC systems.

Pathan [36] presented a Fixed Priority, multi-processor scheme. He describes an algorithm MSM (Mixed Criticality Scheduling algorithms on Multiprocessors) the fundamentals of which are based on previous FP work such as AMC [10]. Alongside this uni-processor algorithm sits multiprocessor scheduling analysis which utilises Audsley's Optimal Priority Ordering [2]. The effectiveness of the technique is evaluated against Deadline-Monotonic & Criticality-Monotonic Priority Orderings and is shown as more effective.

2.4.2 Communication & Shared Resources

Access to shared resources and inter-task/processor communication are particularly challenging topics in Mixed Criticality systems. There is a clear issue when considering the potential communication between low and high criticality

tasks. For example, low $\longrightarrow$ high, the low criticality task may overrun its deadline, sending a message late or not at all. Unless the high criticality task is able to deal with potentially unreliable communication this could cause the system to be unschedulable. Communicating from high $\longrightarrow$ low still poses a problem; consider a high criticality task attempting to send a message to a low criticality task which is not ready to receive. This might be due to the task locking a resource or high levels of interference. The high criticality task might suffer, or even miss a deadline. It is clear that more stringent controls and protocols are required to maintain suitable isolation, but allow controlled communication where appropriate.

Burns and Davis [17] examine Mixed Criticality communication over a Controller Area Network (CAN). They identify similar conflicting requirements to MC scheduling: how to partition use of the network whilst sharing the capacity. A Trusted Network Component (TNC) is key to their solution, a TNC allows for message send requests to be monitored. If these requests are too frequent a criticality mode change occurs. A triggering message [17] is an irregular message that breaks the send request frequency for the current criticality level.

Similar issues exist around the access of shared resources. Yun et al. [42] examine the problems around memory access, providing suitable isolation while preventing intolerable interference. This problem is amplified once again by the introduction of multi/many-core systems. Yun et al. [42] observe that, using a standard controller, a task on an 8 core platform might have its WCET extended by up to 300% while it accesses memory for only 10% of its execution time. Clearly interference like this is prohibitive. To counter this they propose a memory throttling technique, based upon the idea of monitoring the traffic from each core. Budgets for memory access are dealt with in two ways:

- Static Budget Distribution: *Each core has its own budget which is statically distributed from a global budget. All cores share the same period. [42].*

- Dynamic Budget Distribution:*All throttled cores share a single global budget & period. When each core accesses memory it consumes a portion of the global budget. [42].*

In this way Yun et al. [42] present a solution to control memory access by either a static or dynamic scheme.

Hierarchical scheduling is one approach to scheduling on multi-core systems. Lackorzynski et al. [27] observed that performance suffers when criticality levels

are introduced. To remedy this they propose a method of Flattening Hierarchical scheduling, as covered Section 2.2.5.

2.5 Implementations/Frameworks

Among the current body of Mixed Criticality work, there are papers which look at more practical issues surrounding the implementation of MC systems. Huber et al. [24] suggests a resource management structure based on a Trusted Network Authority (TNA) and a Resource Management Authority (RMA). The RMA controls the resources available to any non-safety critical systems; the TNA monitors these systems to ensure they do not interfere with the safety critical applications.

Pellizzoni et al. [37] present a design methodology for SoC based Mixed Criticality systems. This methodology is based upon the idea of Platform-Based Design (PBD) [38] and the use of the Architectural Analysis and Design Language (AADL) [20]. Their work focuses on fault tolerance and the isolation of system components.

The issue of fault tolerance and error handling is also addressed by Axer et al. [4]. They consider SoC based fault tolerance and suggest a check point based system to deal with errors. These check points are created at regular intervals during runtime, the system can be rolled back if an error occurs. However this is only really applicable for soft real-time systems.

Baruah and Burns [9] present an implementation of a fixed priority scheme in Ada. They consider the necessary runtime monitoring, mode change functionality and how it might be implemented in Ada. They demonstrate this behaviour by providing code patterns.

Neukirchner et al. [33] present a contract-based dynamic task management system. The scheme covers a wide range of problems such as task management, memory access, fault tolerance and appropriate functional isolation.

Integrated Dependable Architecture for Many Cores (IDAMC) [32] is another, more complete scheme which aims to satisfy the Mixed Criticality goals of isolation and high utilisation. This is achieved through the use of runtime monitoring, and control/isolation of shared resources. IDAMC also considers fault tolerance and recovery.

2.6 Summary

The review above has covered the key works in what is a rapidly growing field. Vestal's [41] seminal work instigated this fresh study into Mixed Criticality systems. This work is driven by industrial pressures and the increasing cost effectiveness of more powerful and advanced hardware. This has given rise to the desire to consolidate functionality that might have traditionally been spread across many systems. A Mixed Criticality system must manage the balance between the efficient use of these resources while providing suitable levels of isolation and assurance where required. Static and Dynamic scheduling approaches have been considered alongside a raft of other, often more practically minded schemes. As Mixed Criticality study has progressed it is becoming increasingly clear that there is a need to support more complex system architectures with multi-core or many-core support. It is also clear that current scheduling models are, largely, too simplistic. Such approaches are often limited to a single processor and two criticality levels. There is a need to factor in issues around communication, access to shared resources and error handling. Mixed Criticality systems represent a fast moving and challenging area of research. Current work provides a good foundation to allow future study to address additional problems with a view to providing more comprehensive solutions.

In the rest of this work we address the issue of multiple criticality levels. As indicated above much of the published work has restricted itself by considering just two criticality levels. However standards such as ISO 26262, IEC 61508 and DO-178B typically have 4 or 5 levels. As such is is necessary to ensure that the analysis developed for two criticality scales appropriately to incorporate two or more criticality levels.

Chapter 3

The AMCrtb Approach

AMCrtb is a technique proposed by Baruah et al. [10] to provide schedulability analysis for the AMC scheduling policy. The scheme expands upon standard response time techniques in order to facilitate the properties of AMC. Baruah et al. [10] show that AMCrtb strictly dominates SMC for 2 criticality levels, we aim to extend the analysis beyond 2 levels to investigate whether this remains valid. The following chapter will consider the initial analysis proposed in [10] and an extension to this analysis to cope with more than 2 criticality levels. The chapter is structured as follows; Section 3.1 considers the original dual criticality approach, Section 3.2 presents the extensions to AMCrtb for 2 to n criticality levels, Section 3.3 briefly adapts some additional approaches, SMC, SMC-NO and CrMPO (Criticality Monotonic Priority Ordering) for n criticality levels, Section 3.4 presents some illustrative results and Section 3.5 summarises the Chapter.

3.1 Dual Criticality

The original analysis presented by Baruah et al. [10] is shown in Equations (3.1), (3.2) and (3.3). There are two stages to the approach, the first is to consider the LO and HI criticality levels individually and ensure they are schedulable. The second stage is to consider the criticality change from LO to HI and ascertain whether it is feasible. The first step is to assess the schedulability of each criticality mode in the system.

Stage 1A: *Check the schedulability of the LO mode for all tasks.*

$$R_i(LO) = C_i(LO) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{3.1}$$

Stage 1A considers all tasks in the LO criticality mode in order to ensure that the system is schedulable in its LO mode. This equation is solved using standard response time techniques for solving a recursive relation.

Stage 1B: *Check the schedulability of the HI mode for HI tasks.*

$$R_i(HI) = C_i(HI) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \tag{3.2}$$

Where hpH is the set of all higher priority HI criticality tasks. Stage 1B considers only the HI criticality tasks executing to their HI criticality budgets. This ensures that, once a criticality change has occurred, the system is schedulable.

The next step is to assess the schedulability of any HI criticality tasks executing during a criticality level change.

Stage 2A: *Calculate the schedulability of the criticality change for HI tasks.*

$$\begin{aligned} R_i^*(HI) = C_i(HI) + & \sum_{j \in hpH(i)} \left\lceil \frac{R_i^*(HI)}{T_j} \right\rceil C_j(HI) + \\ & \sum_{k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \end{aligned} \tag{3.3}$$

Where hpH is the set of all higher priority HI criticality tasks and hpL is the set of all higher priority LO criticality tasks. Stage 2A assesses the schedulability of the criticality level change. The use of the static value for higher priority but lower criticality tasks allows AMC to place an upper bound upon any potential interference from low criticality tasks during a criticality change. This is possible due to the way in which AMC handles a criticality level change. Under AMC, all LO criticality tasks are suspended when a criticality change occurs, as such during this time their ability to interfere with the high criticality tasks is limited. This limit is the LO response time of the high task as after that time the system will be running in the HI criticality mode, or the task will have completed and no criticality change need occur.

3.2 Many Criticality Levels

When considering n possible criticality levels we examine the two stages used for dual criticality systems in 3.1. In stage one, rather than considering only the LO and HI levels, here we examine each level, up to n, and determine whether they are schedulable. In stage two we consider $n-1$ criticality level changes, for each change we seek to determine whether all tasks in the set will meet their deadlines.

3.2.1 Stage One

Consider a system containing 5 distinct criticality levels, $L1 \ldots L5$ where $L1 > L5$. The analysis for $L5$ must consider the potential interference of all higher priority tasks, regardless of criticality level (as $L5$ is the lowest level). To calculate the interference suffered from higher priority $L4$ tasks we use the following term:

$$\sum_{j \in hp(i) | L_j = L4} \left\lceil \frac{R_i(L5)}{T_j} \right\rceil C_j(L5)$$

The algorithm looks for those higher priority tasks, τ_j, where the criticality level (L_j) is equal to $L4$. This considers any interference suffered from a task at $L4$, but uses their $L5$ values. The calculation can be completed to account for levels $L3 \ldots L1$ as shown in Equation (3.4).

$$\begin{aligned} R_i(L5) = C_i(L5) + & \sum_{j \in hp(i) | L_j = L4} \left\lceil \frac{R_i(L5)}{T_j} \right\rceil C_j(L5) \quad + \\ & \sum_{k \in hp(i) | L_k = L3} \left\lceil \frac{R_i(L5)}{T_k} \right\rceil C_k(L5) \quad + \\ & \sum_{l \in hp(i) | L_l = L2} \left\lceil \frac{R_i(L5)}{T_l} \right\rceil C_l(L5) \quad + \\ & \sum_{m \in hp(i) | L_m = L1} \left\lceil \frac{R_i(L5)}{T_m} \right\rceil C_m(L5) \end{aligned} \tag{3.4}$$

This process is repeated for each of the remaining criticality levels to check their schedulability. The tasks within each criticality level must be analysed to determine whether they are schedulable. It is possible to generalise these equations to one that can deal with $2 \longrightarrow n$ criticality levels. We must consider the schedulability of n criticality levels individually.

For each criticality level.

$$\forall L \in 1 \dots n$$

For all tasks where the criticality level is greater than or equal to L.

$$\forall \tau_i | L_i \geq L$$

Calculate the response times for that level.

$$R_i(L) = C_i(L) + \sum_{j \in hp(i) | L_j \geq L} \left\lceil \frac{R_i(L)}{T_j} \right\rceil C_j(L) \tag{3.5}$$

Equation (3.5) considers the response time of task τ_i at criticality level L by accounting for any interference from higher priority tasks with a criticality level greater than or equal to L. This test is repeated for each of the n criticality modes. In this way response times are calculated for all of the modes a task might execute in, up to their criticality level. Equations (3.1) and (3.2) are the dual criticality application of Equation (3.5).

3.2.2 Stage Two

In addition to assessing the schedulability of each criticality level, it is necessary to consider the behaviour of the system during a criticality change. Criticality changes are assumed to be sequential, if $L5$ is the lowest and $L1$ the highest then the system must go from $L5 \longrightarrow L4 \longrightarrow L3 \longrightarrow L2 \longrightarrow L1$. Therefore in the worst case a task at $L1$ could suffer interference from each criticality level during the final change from $L2 \longrightarrow L1$.

When assessing the interference suffered during a criticality change we must consider two groups of tasks. The first group are those tasks of a higher priority and with a criticality level greater than or equal to the task in question. The interference from these tasks has already been considered in the analysis for each criticality level shown above. As AMC suspends tasks with a criticality lower than the level the system is currently in, the analysis considers only higher priority tasks with a criticality greater than or equal to the current level.

The second group are those tasks with a higher priority but a lower criticality level. It is clear that under AMC, those tasks with a higher priority but lower criticality will have a bounded effect on a higher criticality task if a criticality change occurs (due to AMC suspending lower criticality tasks).

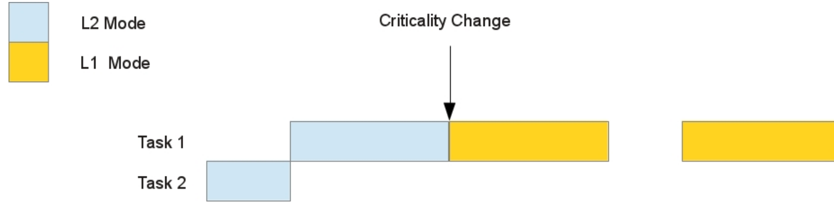

Figure 3.1: Bounded interference example

Consider the case in Figure 3.1, two tasks are executing $\tau_1(L1)$ and $\tau_2(L2)$ where $L1 > L2$ and τ_2 has the highest priority. A criticality change occurs at the instant shown, prior to that both τ_1 & τ_2 have been running at criticality level $L2$. After the criticality change τ_1 continues to execute and τ_2 is suspended. From this it is possible to see that any interference that τ_1 might suffer from higher priority tasks at criticality level $L2$ is bounded by its own $L2$ response time, $R_1(L2)$. As such the interference suffered by a task from a lower criticality level is bounded by its response time at that level.

The interference caused by task τ_2 can be calculated.

$$\left\lceil \frac{R_1(L2)}{T_2} \right\rceil C_2(L2)$$

The value $R_1(L2)$ represents τ_1's response time at criticality level $L2$. This can be generalised to include all higher priority, lower criticality tasks.

$$\sum_{k \in hp(i) | L_k < L_i} \left\lceil \frac{R_i(L_k)}{T_k} \right\rceil C_k(L_k)$$

Here we consider all tasks with a higher priority than τ_i where the criticality level is lower. $R_i(L_k)$ is the response time of τ_i at the criticality level of τ_k. It is worth noting that these values are static and do not change upon each iteration.

The use of a response time value for a task's criticality level and those below implies that the criticality change analysis must begin at the lowest criticality level and ascend. Thus producing the response time values required to bound the interference suffered from lower criticality tasks.

If we combine the analysis for the higher priority tasks with a criticality level greater than or equal to L_i and the analysis for the higher priority tasks with a criticality level less than L_i we can produce an algorithm to assess the feasibility of the criticality level changes in a system. In a system with n criticality levels we must consider $n - 1$ criticality level changes.

For each criticality level.

$$\forall L \in 1 \ldots n$$

For all tasks where the criticality level is greater than or equal to L

$$\forall \tau_i | L_i \geq L$$

Beginning at the lowest criticality level, calculate the schedulability of each criticality change.

$$\begin{aligned} R_i^*(L) = C_i(L) + \sum_{j \in hp(i) | L_j \geq L} \left\lceil \frac{R_i^*(L)}{T_j} \right\rceil C_j(L) \quad + \\ \sum_{k \in hp(i) | L_k < L} \left\lceil \frac{R_i(L_k)}{T_k} \right\rceil C_k(L_k) \end{aligned} \tag{3.6}$$

The algorithm shown in Equation (3.6) will assess the schedulability of the criticality changes within a task set containing $2 \longrightarrow n$ criticality levels. This combined with the algorithm in Equation (3.5) provides an AMCrtb schedulability test generalised to greater than 2 levels of criticality.

3.3 Adapting Additional Approaches

The main focus of this work is on extending AMCrtb and AMCmax. In order to compare their performance with their competitors we must consider the extension of these algorithms as well. We will briefly discuss the extension of Criticality Monotonic Priority Ordering (CrMPO), SMC-NO or Vestal's Original algorithm [41] and SMC (Static Mixed Criticality).

3.3.1 CrMPO

Criticality Monotonic Priority Ordering is the most simplistic of the techniques and is easily the least efficient. Tasks are given priorities based upon their criticality level, the higher the criticality level the higher the priority. If there are multiple tasks of the same criticality level these tasks are assigned priorities in deadline monotonic order. The schedulability of the resulting ordering is then determined via standard response time analysis.

Standard Analysis:

$$R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{3.7}$$

CrMPO does not take into account multiple WCETs depending upon the criticality level, it simply schedules each task, at its criticality level. However, the criticality monotonic ordering does avoid the problem of Criticality Inversion, (Lower criticality tasks with higher priorities interfering with high criticality tasks).

3.3.2 SMC-NO

Vestal's algorithm [41], or SMC-NO Static Mixed Criticality with No runtime monitoring, is a scheme based around standard response time techniques and utilises Audsley's Optimal Priority Assignment algorithm [2] to produce a priority ordering.

SMC-NO:

$$R_i(L_i) = C_i(L_i) + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i(L_i)}{T_j} \right\rceil C_j(L_i) \tag{3.8}$$

The use of $C_j(L_i)$ implies that the WCET value is required for the criticality level L_i. As SMC-NO does not support run-time monitoring and criticality inversion is not avoided, the values used in the analysis must be verified up to the criticality level of the task in question. Therefore lower criticality tasks would have to be verified up to the same level as those higher criticality tasks, this is prohibitively expensive.

3.3.3 SMC

Although SMC supports criticality change functionality no extension is required to allow for $2 \longrightarrow n$ criticality levels. The analysis for SMC is straight forward, the analysis shown below is used in conjunction with Audsley's Optimal Priority Assignment algorithm [2]. *SMC:*

$$R_i(L_i) = C_i(L_i) + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i(L_i)}{T_j} \right\rceil C_j(min(L_i, L_j)) \tag{3.9}$$

The use of $C_j(min(L_i, L_j))$ is to indicate that the WCET value used should be the lower of the two values, for $C_j(L_i)$ or $C_j(L_j)$.

3.4 Some Illustrative Results

In order to determine whether AMCrtb remains dominant over SMC we applied both algorithms, as well as Criticality Monotonic Priority Ordering (CrMPO) to randomly generated task sets. Further details of the experimental set-up can be found in Chapter 6. The graphs below briefly illustrate the performance of AMCrtb against other approaches.

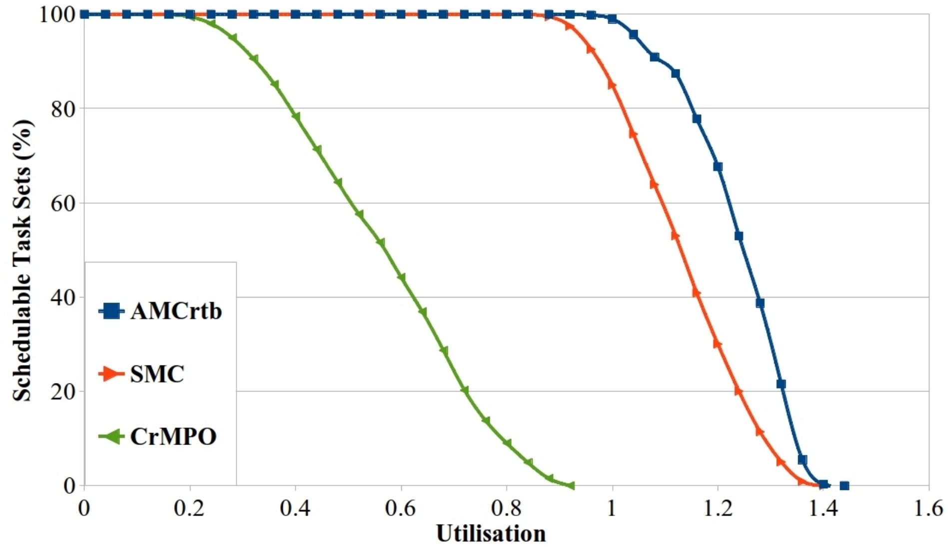

Figure 3.2: AMCrtb, SMC and CrMPO with 2 criticality levels.

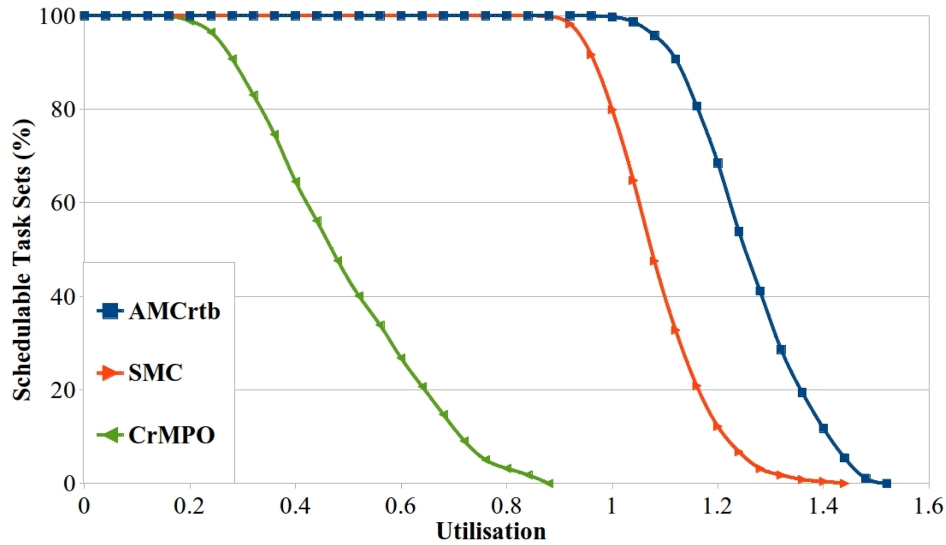

Figure 3.3: AMCrtb, SMC and CrMPO with 5 criticality levels.

Figure 3.2 and Figure 3.3 show the performance of AMCrtb and SMC at 2 and 5 criticality levels respectively. It is clear to see that AMCrtb maintains its dominance over SMC at 2 and 5 criticality levels. This solidifies and extends the conclusion made by Baruah et al. [10] that AMCrtb strictly dominates SMC.

3.5 Summary

The above chapter outlines AMCrtb and how it is extendible to many criticality levels. It also discusses the differences between SMC and AMCrtb and demonstrates that AMCrtb remains the dominant algorithm. This is shown via analytical discussion and experimental results[1].

[1]See Chapter 6 for further experimental work

Chapter 4

The AMCmax Approach

AMCmax is presented as an alternative solution by [10] to assess the schedulability of a task set running under the AMC scheduling policy. As described in the Chapter 3, AMCrtb (AMC Response Time Bound), the other algorithm proposed in [10] is very much an extension of standard response time techniques applied to the mixed criticality problem. AMCmax, however, utilises a novel approach to find the worst case response time of a task. AMCmax exploits the idea that a criticality change may occur only when a task reaches its execution budget for the current criticality level without signalling completion, therefore there are a finite set of points at which the criticality change might take place. By examining the execution of a task set, it is possible to determine which of these points might lead to the worst case response time. The work is organised as follows, Section 4.1 describes AMCmax in further detail and considers the dual criticality analysis presented in [10], Section 4.2 explores the extension of the analysis to 3 and eventually n possible criticality levels, Section 4.3 considers some experimental results and provides an evaluation and Section 4.4 provides a summary.

4.1 Original Analysis

In this section we will consider the original analysis presented by Baruah et al. [10], this analysis is restricted to dual criticality levels, LO and HI. As such the following discussion will consider one criticality change at time s. As we established above, a criticality level change can occur only when a task

executes to its budget without signalling completion. There are a finite number of points in time at which this change might take place. It is possible to bound these points as the criticality change must occur sometime between the start of execution, time 0 and the LO response time ($R_i(LO)$). AMCmax uses these points and seeks to determine the point at which the worst case phasing for a HI criticality task might occur.

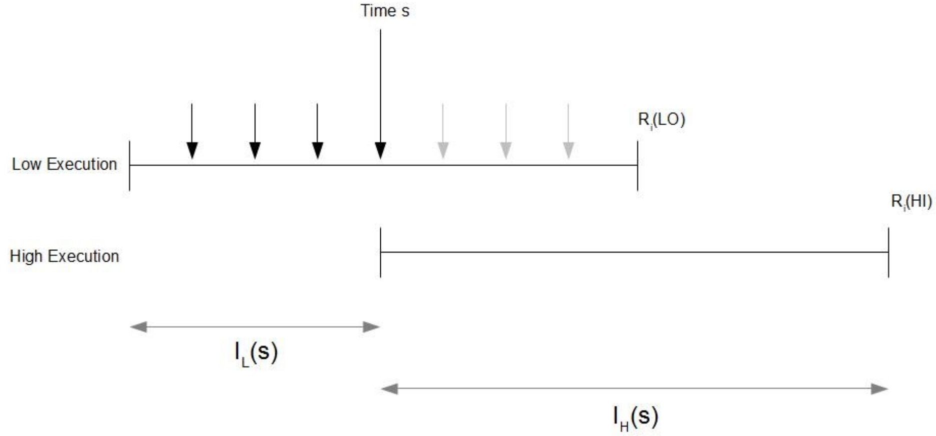

Figure 4.1: Example AMCmax criticality change.

Figure 4.1 shows a criticality change occurring and the system moving into the HI mode. The diagram also shows the Interference suffered in both the LO and HI modes. The possible times of s are shown by the shorter arrows and the time of the criticality change by the long arrow. Baruah et al. [10] illustrate this change with Equation (4.1), showing the calculations required to determine the response time of a high criticality task if the change occurs at time s.

$$R_i^s(HI) = C_i(HI) + I_L(s) + I_H(s) \tag{4.1}$$

From Equation (4.1) it is easy to see the two segments of interference we must assess, I_L and I_H. These sections are also shown in Figure 4.1.

The technique used to assess the response time of low criticality tasks is straightforward, it can be seen in Equation (4.2):

$$I_L(s) = \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) \tag{4.2}$$

Where $hpL(i)$ denotes the sum of high priority, LO criticality tasks. The equation follows standard response time techniques but utilises the floor function rather than ceiling. The floor function is used to ensure that all tasks are accounted for immediately upon release. This algorithm is also used when calculating the LO response times of all tasks, in this case $R_i^s(LO)$ is used rather than s.

Having considered the Low criticality tasks we can return to Equation (4.1). Baruah et al. [10] consider how to calculate the response time of a high criticality task. They consider the high mode as an interval of $t - s$ where $t > s$. t is the response time of the task and is the value that is replaced in each iteration. The number of releases in this interval, $t - s$, can be calculated:

$$\left\lceil \frac{t - s}{T_k} \right\rceil + 1$$

This can be extended for cases where $D_i < T_i$:

$$\left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1$$

The full calculation is shown in Equation (4.3) presented in the form of a function M. With input parameters k, s and t, where k is the task, s is time s and t is time t (or the response time replaced into the equation).

$$M(k, s, t) = min\left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{4.3}$$

The use of *Ceiling* +1, in Equation (4.3), is to account for the completion of all tasks within the interval $s \dots R_i(HI)$. Rare cases are possible where the calculation is overly pessimistic, to deal with this the function ensures that the value returned is no greater than the total number of releases.

The number of releases in the *LO* criticality mode is easily calculable by removing the results of Equation (4.3) from the total number of releases.

$$\left(\left\lceil \frac{t}{T_k} \right\rceil - M(k, s, t) \right) C_k(LO)$$

Therefore $I_H(s)$ is:

$$I_H(s) = \sum_{k \in hpH(i)} \left\{ (M(k,s,t)C_k(HI)) + \left(\left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k(LO) \right) \right\} \tag{4.4}$$

And thus the full equation:

$$\begin{aligned} R_i^s = & \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) + \\ & \sum_{k \in hpH(i)} \left\{ (M(k,s,R_i^s)C_k(HI)) + \right. \\ & \left. \left(\left(\left\lceil \frac{R_i^s}{T_k} \right\rceil - M(k,s,R_i^s) \right) C_k(LO) \right) \right\} \end{aligned} \tag{4.5}$$

And:

$$R_i = max(R_i^s) \forall_s$$

Finally they look at which points of s, within $0 \ldots R_i(LO)$ require consideration. Baruah et al. [10] note that the amount of low criticality interference increases (as a step function), as the value of time s increases, this is due to an increased amount of time spent in the low criticality mode. Similarly the high criticality interference decreases as the low increases. Therefore the response time changes only at the release of a low criticality job, thus we can limit our search to points of s where a LO criticality job is released. It is worth noting that although this behaviour is applicable to the current model, in reality a criticality change could occur whenever any task does not signal completion.

4.2 Extending the Analysis

In the section below we consider the extensions required to allow AMCmax to facilitate greater than 2 levels of criticality. The first Section considers the extension to 3 criticality levels (HI, ME, LO), the second Section considers a similar 3 criticality task system (using levels A, B and C) but explains how the process described during the extension from 2 to 3 criticality levels is applicable for any number of additional levels.

4.2.1 Adding a Medium Level

It is possible to extend the original analysis for AMCmax to include a medium (ME) criticality level.

It is important to reconsider the original premise of the algorithm, locating the time during execution that a criticality change occurs. For two criticality levels this change occurs at time s, therefore when considering 3 criticality levels (LO, ME, HI) there will be two possible criticality changes. For each criticality change from level LO to ME there will be a number of criticality changes from ME to HI.

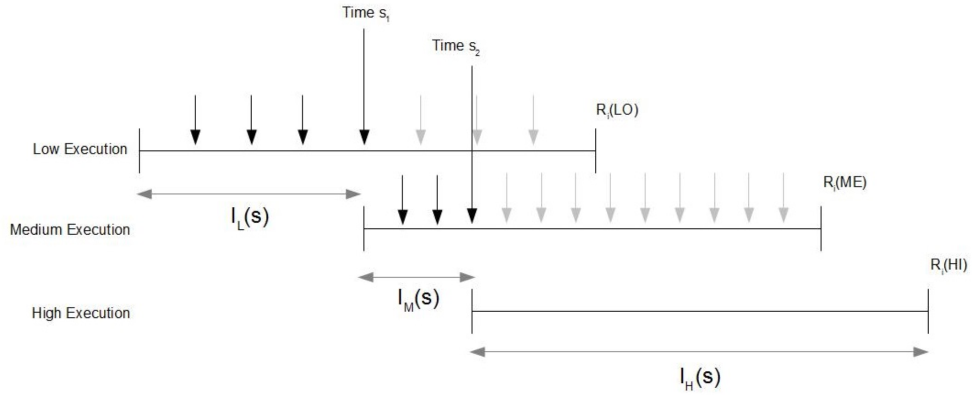

Figure 4.2: AMCmax with three criticality levels.

The diagram shown in Figure 4.2 considers the change from $LO \longrightarrow ME$ and $ME \longrightarrow HI$. Time s_1 is the time at which the $LO \longrightarrow ME$ change occurs and time s_2 is the time at which the $ME \longrightarrow HI$ change occurs. For each point of time s_1, there will be a number of points of time s_2 to check. Where time $s_2 \geq s_1$ and $s_2 \leq R_i(ME)$.

We can produce a formula for high criticality tasks, similar to Equation (4.1), for 3 criticality levels.

$$R_i^s(HI) = C_i(HI) + I_L(s) + I_M(s) + I_H(s) \tag{4.6}$$

And thus the medium:

$$R_i^s(ME) = C_i(ME) + I_L(s) + I_M(s) \tag{4.7}$$

The analysis for the first criticality level (LO) will remain the same using the value of s_1 rather than s (see Equation (4.2)).

Although each algorithm calculates the response time for its criticality level and those below, these might not be the worst case values. As such response times will need to be calculated for a task's criticality level and those below using a separate algorithm for each level. For example, the ME response time produced during the calculation of $R_i^s(HI)$ might lead to the highest HI response time, but not the highest ME value. Each criticality level must be checked in order to ensure that deadlines can be met at that level.

Like AMCrtb, the nature of the algorithm requires that the criticality levels be calculated in order (lowest to highest). However AMCmax does not utilise the response time values produced for the lower levels directly, rather they are used to provide an upper bound to the time in which points of s (the criticality change) can occur. For example the upper bound on time s_1 in a 3 criticality system is $R_i^s(LO)$, therefore for a medium criticality task, the value of $R_i^s(LO)$ must be calculated before $R_i^s(ME)$. This is because the value $R_i^s(LO)$ is used to provide an upper bound on the possible points of s_1 when analysing the response time of the ME criticality level of τ_i. Response time values are required for a task at its criticality level and all those below.

Having established that the low criticality analysis will remain the same we must consider the medium level. As, at the medium level we are only dealing with one criticality change, the analysis is similar to the HI analysis for two criticality levels.

The ME mode as it is the highest mode in this case, is calculated using the method M, defined in Equation (4.3).

$$M(k, s_1, R_i(ME))$$

By removing the above from the total number of releases we can calculate the interference suffered in the LO mode.

$$\left(\left\lceil \frac{R_i(ME)}{T_k} \right\rceil - M(k, s_1, R_i(ME)) \right) C_k(LO)$$

The medium response time can be seen in Equation (4.8):

$$R_i(ME) = \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(LO) +$$
$$\sum_{k \in hpM(i)} \Bigg\{ M(k, s_1, R_i(ME)) C_k(ME) + \tag{4.8}$$
$$\left(\left\lceil \frac{R_i(ME)}{T_k} \right\rceil - M(k, s_1, R_i(ME)) \right) C_k(LO) \Bigg\}$$

Where:

$$R_i(ME) = max(R_i^s(ME)) \forall s_1$$

The algorithm presented above for the medium criticality level is similar in structure to the high criticality calculation for two criticality levels. The high criticality calculation for three criticality levels is more problematic. The calculation for the LO level will remain the same, using *floor* +1 to account for all releases (see Equation (4.2)). The medium level is a little more challenging as it represents the intermediary time between s_1 and s_2. If a medium or high criticality job is mid execution when the criticality change occurs it is given its medium criticality budget to complete, such a task must be considered as a full execution within the medium mode. If the criticality change is triggered from ME to HI whilst a HI criticality task is mid execution, this will be considered as a high criticality execution and thus need not be considered in the medium mode. During this time all task releases must be taken into account, for this we can use the *floor* +1 function.

However a medium or high criticality task cannot trigger the change from LO to ME, we need only consider the points of s_1 on which a low criticality task are released. Therefore we must consider how a medium or high criticality task might execute across two or more criticality levels.

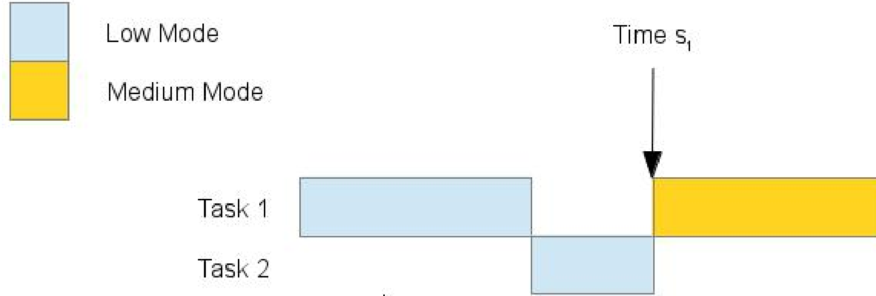

Figure 4.3: A criticality change showing execution of Task 1 in two modes.

Figure 4.3 shows two tasks, Task 1 and Task 2. Task 1 has the highest criticality but the lowest priority, thus Task 2 has the lowest criticality but the highest priority. The diagram shows Task 1 running, it is pre-empted by Task 2 which reaches its budget for the low criticality level without signalling completion. As Task 2 is only a low criticality level task, under AMC it is dropped and Task 1 resumes its execution in the higher mode with its extended budget. From this it is clear to see how, even though the task that caused the criticality change may get dropped, any task of a suitable criticality level with execution time still to complete might execute in the new criticality mode. The diagram also illustrates the need to ensure that a task which executes over two criticality levels is included only in the higher of the two criticality levels, to include it in both would introduce an unnecessary level of pessimism.

In order to include this in our algorithm we must define a new function, similar to M, this function is shown in Equation (4.9).

$$N(k, s_1, s_2) = \left\lfloor \frac{s_2 - s_1 - (T_k - D_k)}{T_k} \right\rfloor + 1 \tag{4.9}$$

Function N makes use of the *floor* +1 calculation to provide the interference suffered between time s_1 and s_2. Therefore the following will provide the ME criticality interference for a high criticality task τ_i:

$$N(k, s_1, s_2) C_i(ME)$$

Following this we can define each of the stages in the calculation shown in Equation (4.6):

$$I_L(s) = \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO)$$

$$I_M(s) = \sum_{k \in hpM(i)} \left\{ N(k, s_1, s_2) C_k(ME) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(LO) \right\}$$

$$I_H(s) = \sum_{k \in hpH(i)} \left\{ M(k, s_2, t) C_k(HI) + N(k, s_1, s_2) C_k(ME) + \left(\left\lceil \frac{t}{T_k} \right\rceil - N(k, s_1, s_2) - M(k, s_2, t) \right) C_k(LO) \right\}$$

It is worth emphasising that the calculation for the medium mode here is different than in Equation (4.8), this is due to the fact that the medium mode here is defined as the time between s_1 and s_2 and is therefore subject to calculation using the N function rather than M. This brings up an important observation, for systems with greater than two criticality levels, there are three distinct stages. Stage one is the low calculation, this calculation always remains the same (see Equation (4.2)). Stage two is the intermediary stage, any criticality level which is bounded by two points of s, or any criticality level which is not the lowest or the highest. Stage three is the highest criticality level, the calculation for this level always utilises the M function.

The full calculation for $R_i^s(HI)$ is shown in Equation (4.10):

$$\begin{aligned} R_i^s(HI) = C_i(HI) + \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(LO) + \\ \sum_{k \in hpM(i)} \left\{ N(k, s_1, s_2) C_k(ME) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(LO) \right\} + \\ \sum_{l \in hpH} \left\{ M(l, s_2, R_i^s(HI)) C_l(HI) + N(l, s_1, s_2) C_l(ME) + \right. \\ \left. \left(\left\lceil \frac{R_i^s(HI)}{T_l} \right\rceil - N(l, s_1, s_2) - M(l, s_2, R_i^s(HI)) \right) C_i(LO) \right\} \end{aligned} \tag{4.10}$$

Where:

$$R_i^s(HI) = max(R_i^s(HI))\forall s$$

The calculations are performed following the structure shown in Figure 4.4.

for all s_1 where $0 \leq s_1 < R_i^s(LO)$ **do**
  for all s_2 where $s_1 \leq s_2 < R_i^s(ME)$ **do**
    AMCmax()
  end for
end for

Figure 4.4: The structure of execution for AMCmax over 3 criticality levels.

As mentioned in section 4.1, in a dual criticality system, Baruah et al. [10] note that the HI criticality response time of a task increases only on the releases of LO criticality tasks. Therefore these releases are the only points of s we need check. The same principle applies when dealing with greater than two criticality levels. During a change from the ME to the HI mode we consider points of s_2, as the value of $R_i(HI)$ can only increase on the release of a ME criticality task, we need only check these points of $s2$ where an ME task is released.

4.2.2 To n Criticality Levels

The process of adding additional criticality levels to a system is best illustrated by re-considering the extension from 2 to 3 criticality levels. The Section below describes this process and explains how it might be repeated to account for n possible criticality levels.

Consider the case of two criticality levels A and B, where A is the lowest criticality level in the system, (B > A). We would use the analysis from Section 2, this is repeated below for convenience.

$$\begin{aligned} R_i(B) = C_i(B) + \sum_{j \in hpA(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(A) + \\ \sum_{k \in hpB(i)} \Big\{ (M(k, s_1, R_i(B))C_k(B) + \\ \left(\left(\left\lceil \frac{R_i(B)}{T_k} \right\rceil - M(k, s_1, R_i(B)) \right) C_k(A) \right) \Big\} \end{aligned} \tag{4.11}$$

Where hpA refers to all higher priority, criticality A tasks and hpB refers to all higher priority, criticality B tasks.

In order to determine the response time of a level B task, AMCmax considers points of s where the criticality change might occur. These points are bounded by the $R_i(A)$ response time of the task in question. If this system were also to include a criticality level C, such that $C > B > A$ then a criticality change might occur at any point (s_2) between the original change from A to B, point s, and the task's response time in criticality mode B, $R_i(B)$. To show this we can look to Equation (4.10) which deals with a three criticality system and makes use of the function N shown in Equation (4.9) to calculate the interference between two points of s_n.

The calculation for the A criticality tasks remains the same, we use s_1 rather than s in order to differentiate between criticality changes.

$$\sum_{j \in hpA(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(A)$$

The calculation for the B criticality tasks changes to make use of the function N (see Equation (4.9)) as it is now used to determine the interference suffered between two points of s. The criticality A interference is calculated by removing the number of releases, as calculated by function N from the total number of releases.

$$\sum_{k \in hpB(i)} \left\{ N(k, s_1, s_2) C_k(B) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(A) \right\}$$

Finally we may consider the calculation for criticality level C. Functions M and N are used in order to calculate the interference a criticality C task might suffer in modes C and B respectively. Both functions M and N are removed from the total number of releases to calculate the criticality A response time.

$$\sum_{l \in hpC} \left\{ M(l, s_2, R_i(C)) C_l(C) + N(l, s_1, s_2) C_l(B) + \left(\left\lceil \frac{R_i(C)}{T_l} \right\rceil - N(l, s_1, s_2) - M(l, s_2, R_i(C)) \right) C_i(A) \right\}$$

Where hpC refers to all higher priority tasks of criticality level C. The final calculation is shown in Equation (4.12).

$$R_i(C) = C_i(C) + \sum_{j \in hpA(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(A) + \\ \sum_{k \in hpB(i)} \left\{ N(k, s_1, s_2) C_k(B) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(A) \right\} + \\ \sum_{l \in hpC} \left\{ M(l, s_2, R_i(C)) C_l(C) + N(l, s_1, s_2) C_l(B) + \\ \left(\left\lceil \frac{R_i(C)}{T_l} \right\rceil - N(l, s_1, s_2) - M(l, s_2, R_i(C)) \right) C_i(A) \right\} \tag{4.12}$$

Where:

$$R_i(C) = max(R_i^s(C)) \forall s_n$$

Equation (4.12) shows how $R_i(C)$ can be calculated by considering points of s_1 where the change from A to B might occur and points of s_2 where the change from B to C might occur.

If we were to extend this system to introduce a 4th criticality level, D, we would follow the same steps as we did for criticality level C. Consider points for the criticality change from C to D at time s_3 bounded by the criticality C response time, $R_i(C)$. It is important to note that the function N is always used to calculate the number of releases between two points of s and the function M is always used to calculate the response time at the highest criticality level in the system.

Figure 4.5 shows the system analysed in Equation 4.11. The system has two criticality levels A and B.

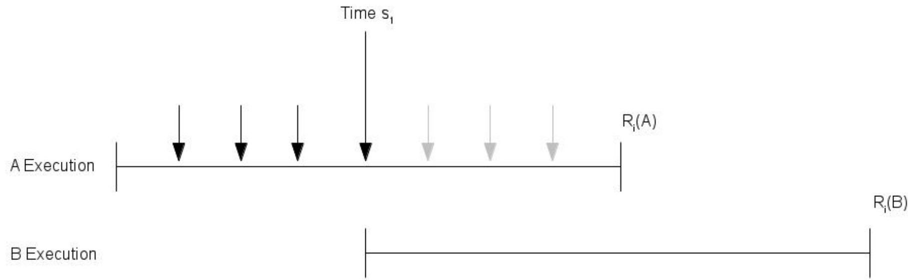

Figure 4.5: A system with modes A and B.

If a 3rd criticality level is introduced, C, points of s_2 are considered between s_1 and $R_i(A)$. The addition of level C is shown in Figure 4.6.

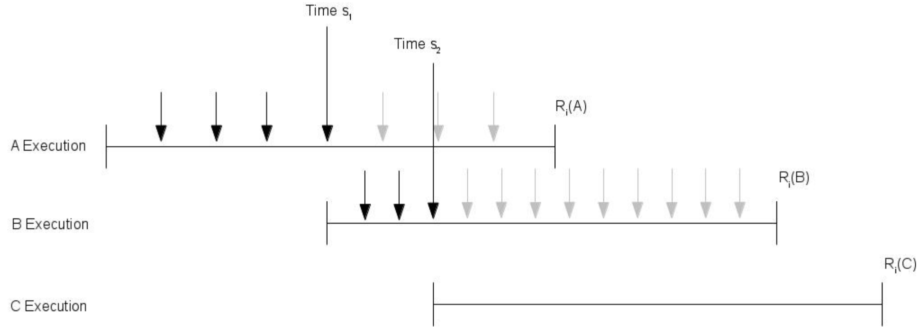

Figure 4.6: The system with modes A and B with an additional level, C added.

This process of adding another set of points to check for each criticality level may be repeated to account for as many criticality levels as desired. However, the computational load increases almost exponentially with the number of criticality levels in the system. Increasing the number of tasks to be analysed or increasing the difference between the WCETs of each criticality level will also have an impact on the level of computation required.

It is worth noting that calculating the criticality level A response time by removing the response times of criticality levels B and C from the total number of releases could cause undue pessimism. The calculations for levels B and C are, by the nature of this analysis, pessimistic. In rare cases, removing these values from the total number of releases could imply a criticality A response time of less than 0. Clearly this is not desirable behaviour. As such checks must be in place to ensure that the criticality level A response time is at least 0 (assuming that a criticality level A task is released first).

4.3 Some Illustrative Results

Although [10] shows that AMCmax dominates AMCrtb, it is also clear that AMCmax is a far more computationally intensive test. In order to assess firstly whether AMCmax maintains its dominance as the number of criticality levels is increased and secondly whether the number of additional schedulable tasks is significant enough to warrant the use of the more expensive AMCmax, we tested the algorithms against sets of artificially generated task sets (The specifics

behind the task generation are covered in Chapter 6).

tasks, L2 with 3 tasks, L3 with 2 tasks and L4 with 2 tasks, where $L4 > L1$.

The following graphs show AMCrtb, AMCmax and SMC. Due to the nature of mixed criticality task sets it is possible for a task set to appear to have a utilisation greater than 1, as we consider a task's utilisation at its own criticality level, not the lowest.

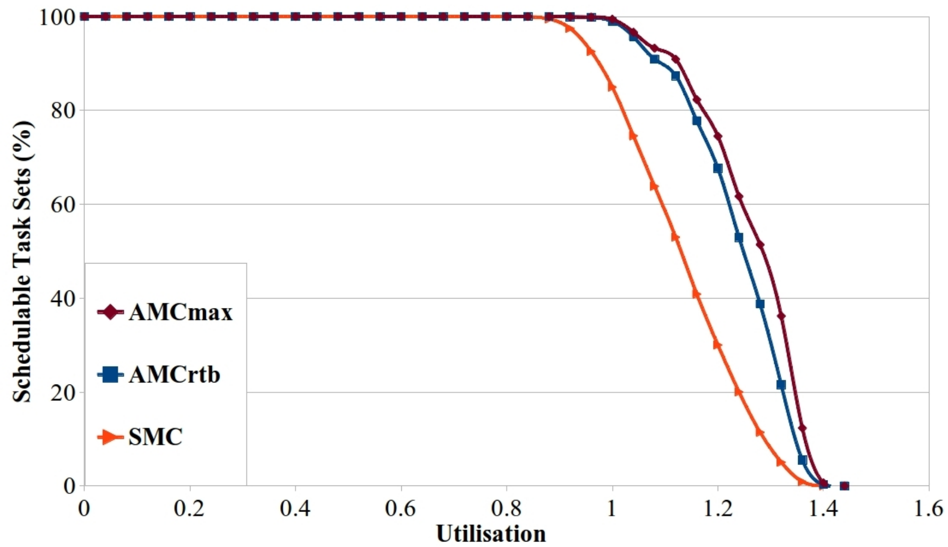

Figure 4.7: AMC, AMCmax and SMC . 2 Criticality Levels

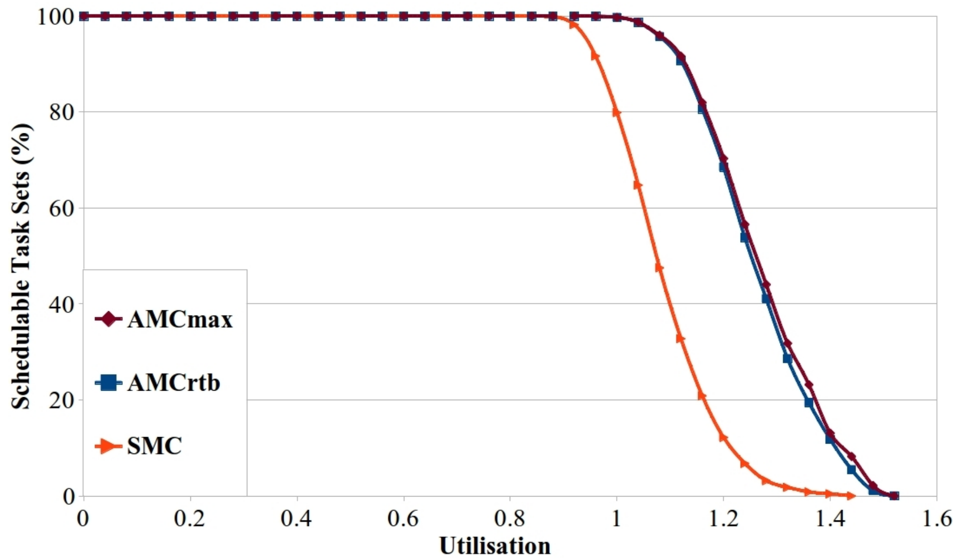

Figure 4.8: AMC, AMCmax and SMC . 5 Criticality Levels

Figures 4.7 & 4.8 show the performance of AMCmax relative to AMC at 2 and 5 criticality levels respectively. It is immediately obvious that the algorithms maintain their relative effectiveness, despite the additional criticality levels. This is perhaps the most significant result of the experimentation as it furthers the claims of [10] that AMC strictly dominates SMC and AMCmax strictly dominates AMCrtb. However the margin between AMCrtb and AMCmax remains small, as such there are cases where it might be justifiable to use AMCrtb rather than AMCmax. Although AMCmax dominates AMCrbt it does so at significant processing cost. As the number of criticality levels is increased the computational time required increases significantly. This is because each level adds additional sets of points of s to search. Coupled with the need to implement Audsley's Optimal Priority Assignment algorithm [2], AMCmax is extremely computationally intensive.

4.4 Summary

To summarise, it is possible to extend AMCmax to facilitate n levels of criticality, as we have shown above. However, the nature of searching many points of s within many points of s is extremely computationally intensive. remains to be seen whether the schedulability gain provided by AMCmax is significant

enough to warrant the higher workload.

Chapter 5

Period Transformation

Period Transformation (PT) is an older scheduling scheme that has been considered for use in Mixed Criticality Systems. The original technique [39] involves transforming the periods, deadlines and worst-case execution time/s of each task in a set in order to create a harmonic task set. A harmonic task set is a set in which all task periods are multiples of each other. The advantage of such a set is that if a transformed task set has a utilisation ≤ 1, it is schedulable under Rate Monotonic priority assignment. It is clear that such a bound is attractive as theoretically, 100% CPU utilisation is possible. However by dividing up tasks in this way will significantly increase the number of context switches and require some form of run-time monitoring to ensure the transformed tasks execute correctly. These additional overheads may be relatively high and are perhaps one of the key reasons for the poor industrial uptake of Period Transformation. Recently PT has been reconsidered for application to MC systems. The following section considers the original PT analysis, extensions for MC systems and its flaws. Section 5.1 describes the original PT analysis, Section 5.2 presents Vestal's approach [41], Section 5.3 considers some improvements to this analysis, Section 5.4 extends the work beyond 2 criticality levels, Section 5.5 presents a small set of experimental results and Section 5.6. provides a summary.

5.1 Standard Period Transformation

The purpose of period transformation was originally to allow any task set to be made harmonic. All tasks within a set are reduced by a factor, n, where τ_j is

the task with the shortest period.

$$n = \frac{T_j}{T_i}$$

And thus the transformed period of τ_j, $T_j^{'}$, would be equal to.

$$T_j^{'} = T_j/n$$

This transformation would standardise each period with a task set, making the set harmonic and thus subject to ≤ 1 utilisation bound for Rate Monotonic schedulability.

Consider the following example show in Table 5.1.

τ	C	T/D
1	4	8
2	10	20

Table 5.1: Period Transformation Example

Using Rate Monotonic assignment, τ_1 would be assigned the highest priority. Using standard Response Time Analysis (RTA) it is clear that this task set is not schedulable.

Period Transformation can be applied to transform τ_2 in order to make the task set harmonic.

$$n = \frac{20}{8}$$

In this case τ_2 will need to be reduced by a factor of 2.5.

$$T_2^{'} = 20/2.5$$

$$C_2^{'} = 10/2.5$$

The transformed task will be identical to τ_1 with a period of 8 and an execution time of 4. Although the utilisation of this new task set is equal to 1[1], the task set is harmonic, therefore its utilisation is ≤ 1 and is schedulable under RM assignment.

[1] $4/8 + 4/8$

5.2 Vestal's Period Transformation

Vestal [41] proposed a Mixed Criticality scheduling approach based upon Period Transformation. He uses PT, not to create a harmonic task set, but to allow for a Criticality and Rate Monotonic based priority ordering which has been shown not to be optimal for untransformed MC systems [41]. The approach, which focuses on just two criticality levels HI and LO, proposes that only those HI criticality tasks with periods greater than or equal to that of the shortest LO criticality period be transformed. This will allow all HI criticality tasks to attain a higher priority than the LO and thus avoid the problem of priority inversion.

This gives us 3 groups of tasks. Those of a LO criticality, these do not need transformation. Those with a HI criticality but a period shorter than the shortest LO criticality task, these do not need transformation. Finally those with a HI criticality with a period greater than that of the shortest LO criticality task, these are the tasks that must be transformed.

The analysis of HI criticality tasks, in the HI mode is done via standard response time techniques on untransformed task sets. The analysis for the LO criticality mode is detailed as follows.

As we are no longer creating harmonic task sets we calculate n slightly differently:

$$n = \left\lceil \frac{T_j}{T_i} \right\rceil$$

We use the ceiling function to ensure that our calculations remain integers.

At runtime, transformed tasks are expected to execute up to their $C_j(HI)/n$ until they reach their untransformed, $C_j(LO)$, only then can we determine if a task will overrun its LO execution bounds and a mode change would need to occur. Transformed tasks, running in the LO mode execute in $C_j(HI)/n$ time slices until $C_j(LO)$.

Consider the example task set in Table 5.2.

τ	$C(LO)$	$C(HI)$	T/D	L
i	2	-	5	LO
j	5	10	25	HI

Table 5.2: Untransformed Mixed Criticality Example

Task 2 is transformed by a factor of 5 to allow it the highest priority. The resulting task set is shown in 5.3.

τ	$C'(LO)$	$C'(HI)$	T'/D'	L
i	2	-	5	LO
j	1	2	5	HI

Table 5.3: Transformed Mixed Criticality Example

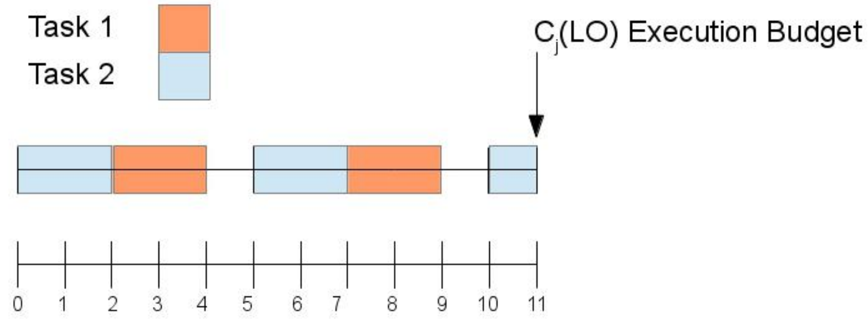

Figure 5.1: $C_j(HI)/n$ slices completing within $C_j(LO)$

Figure 5.1 shows the transformed task set shown in Table 4 executing in its LO mode. As τ_j is the transformed task, each $C_j(HI)/n = 2$, $C_j(LO)/n = 1$ and its original (untransformed) $C_j(LO)$ execution budget is 5. The transformed task executes for its $C_j(HI)/n$ but completes early on its 3rd execution and is therefore still operating in the LO criticality mode.

The number of transformed dispatches could be calculated as follows:

$$\left\lceil \frac{R_i}{T_j/n} \right\rceil$$

The number calculated above will contain several complete executions of $C_j(LO)$ and a remainder, this remainder can execute for no longer than $C_j(LO)$ so vestal assumes this value. He calculates the number of transformed executions which complete to $C_j(LO)$:

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO)$$

So including the added pessimism of those transformed executions that do not complete, the total interference from τ_j can be summed as follows.

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO) + C_j(LO)$$

Figure 5.2 shows complete executions of $C_j(LO)$, transformed executions, $C_j(LO)/n$ and incomplete transformed executions that do not consist of an entire execution of $C_j(LO)$.

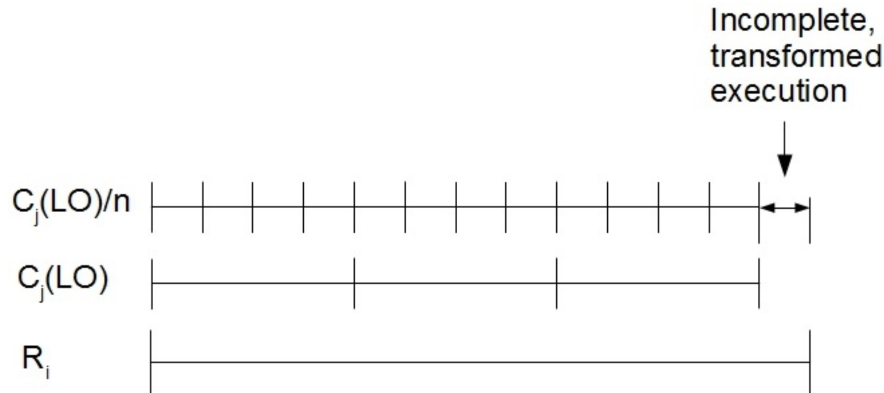

Figure 5.2: An example showing transformed executions constituting a complete $C_j(LO)$

Clearly there are several disadvantages to Vestal's technique. The use of $C_i(LO)$ to account for transformed releases which do not constitute an entire $C_i(LO)$ is clearly overly pessimistic. Vestal's approach also loses one of the key properties of Period Transformation, the ability to create harmonic task sets and, by proxy, the ≤ 1 utilisation bound for RM schedulability. Although not all tasks are transformed it is likely that by transforming the HI tasks the number of context switches will increase significantly. This coupled with the need to monitor transformed executions to ensure the correct sections of code are executing will cause PT overheads to remain, perhaps prohibitively, high.

5.3 Improving the Analysis

In his analysis, Vestal [41] assumes a value of $C_i(LO)$ for the remaining transformed executions, $C_i(HI)/n$ that do not constitute a complete execution of $C_i(LO)$. This value, although an effective upper bound, is undesirably pessimistic. The work below considers a more accurate approach to finding the interference from transformed executions that do not constitute a complete untransformed execution.

We still calculate the number of complete executions of $C_j(LO)$:

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO)$$

The analysis differs here, rather than just assuming the value of $C_j(LO)$ for all remaining transformed executions, we seek to determine the size of the incomplete interval. To find the size of the remaining interval, P, we do the following:

$$P = R_i - \left\lfloor \frac{R_i}{T_j} \right\rfloor T_j$$

And thus we use the value P, to calculate the number of transformed executions within the remaining interval:

$$x = \left\lceil \frac{P}{T_j/n} \right\rceil \frac{C_j(HI)}{n}$$

Therefore the complete calculation will include the transformed tasks within the incomplete interval and the complete executions of $C_j(LO)$. This is shown in Equation [5.1].

$$min\{x, C_j(LO)\} + \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO) \tag{5.1}$$

The interference suffered from the transformed tasks within the incomplete interval will be the minimum of x or $C_j(LO)$. The use of $\min\{\}$ ensure that we are not overly pessimistic as the remaining interval cannot be greater than $C_j(LO)$.

This coupled with the standard techniques used to analyse the HI criticality mode provide the analysis for dual criticality systems with Period Transformation.

5.4 Greater Than Two Criticality Levels

In-keeping with the nature of this work we then considered how the more accurate analysis presented above might be adapted to work in a system with greater than two criticality levels. The analysis itself is applicable with little alteration.

As task sets will be ordered in Criticality Monotonic order any task with a criticality level less than the current, L_i, cannot cause interference. Those higher criticality, transformed tasks will be allowed to execute for their transformed execution time at their criticality level, $C_j(L_j)/n$ until their untransformed

budget at the current criticality level, $C_j(L_i)$. If execution continues then a criticality change will occur. The number of complete executions of $C_j(L_i)$ within the interval can be calculated.

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(L_i)$$

And therefore we calculate the interference in the remaining time period from the transformed executions.

$$P = R_i - \left\lfloor \frac{R_i}{T_j} \right\rfloor T_j$$

$$x = \left\lceil \frac{P}{T_j/n} \right\rceil \frac{C_j(L_j)}{n}$$

The complete calculation for n criticality levels is shown in Equation [5.2].

$$min\{x, C_i(L_i)\} + \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(L_i) \tag{5.2}$$

As can be seen, the analysis is almost directly applicable. The key challenge is the transformation of the tasks in such a way that a criticality monotonic order is created.

The transformation technique proposed by Vestal required those tasks of a higher criticality level, with a period longer than the shortest period of any LO task, to be transformed. Following this, when considering more than two criticality levels it might be tempting to transform all higher criticality tasks with a period greater than the shortest period of any LO task. However this is problematic. Consider the example in Table 5.4.

τ	T	L
1	80	HI
2	110	ME
3	100	LO

Table 5.4: 3 Criticality Level PT Example, Untransformed

Of the three tasks shown in Table 5.4, Task 2 is the only one requiring transformation as it has a period greater than that of Task 3's. We can calculate the transformation factor, n, as follows:

$$n = \left\lceil \frac{110}{100} \right\rceil$$

Thus n = 2, this will give Task 2 a transformed period of 55, less than the period of Task 1. The set is not criticality monotonic and, as such, it is clear that this calculation will not suffice.

Instead the process must be iterative, beginning at the lowest criticality level and moving upwards considering any tasks in the level immediately above. Tasks in the level immediately above are transformed if their period is greater than the shortest period at the current level. To illustrate this, re-consider Table 5.4. We begin the transformation by considering the lowest criticality level, Task 2 is in the ME level, immediately above Task 3 which is LO. Task 2 has a larger period than Task 3, thus as before we transform Task 2 by a factor of 2, giving it a new period of 55. Next we consider any tasks in the level immediately above ME, Task 1 is a HI criticality task, thus is in the level above ME. Task 1 has a period of 80, greater than the newly transformed period of Task 2. As such Task 1 is transformed by a factor of 2 leaving it with a period of 40, less than Task 2's period of 55. The resulting transformed task set is shown in Table 5.5.

τ	T	L
1	40	HI
2	55	ME
3	100	LO

Table 5.5: 3 Criticality Level PT Example, Transformed

The task set may now be scheduled in Rate Monotonic and Criticality Monotonic order.

5.5 Some Illustrative Results

Here we briefly consider the schdulability of Period Transformation against AMCrtb and AMCmax. We do not account for the additional overheads.

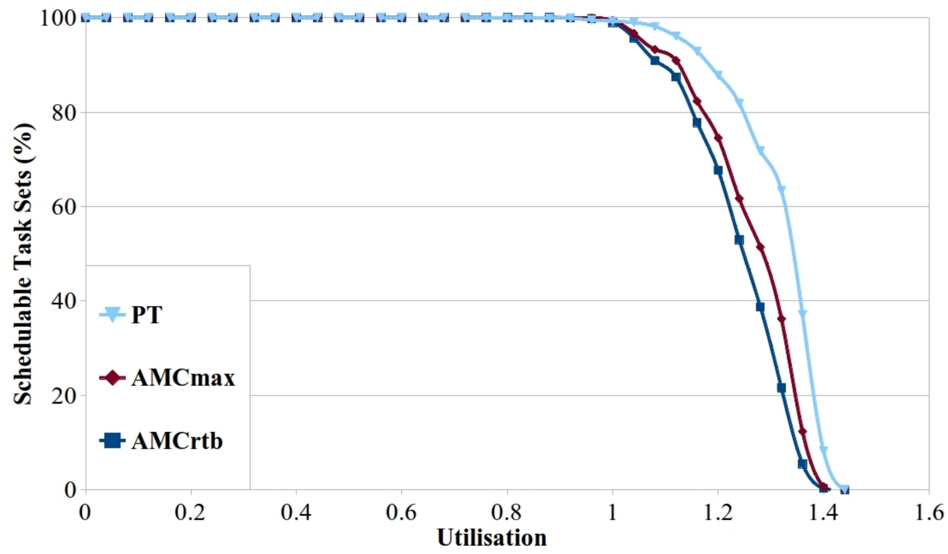

Figure 5.3: Period Transformation, AMCrtb and AMCmax: 2 Criticality Levels

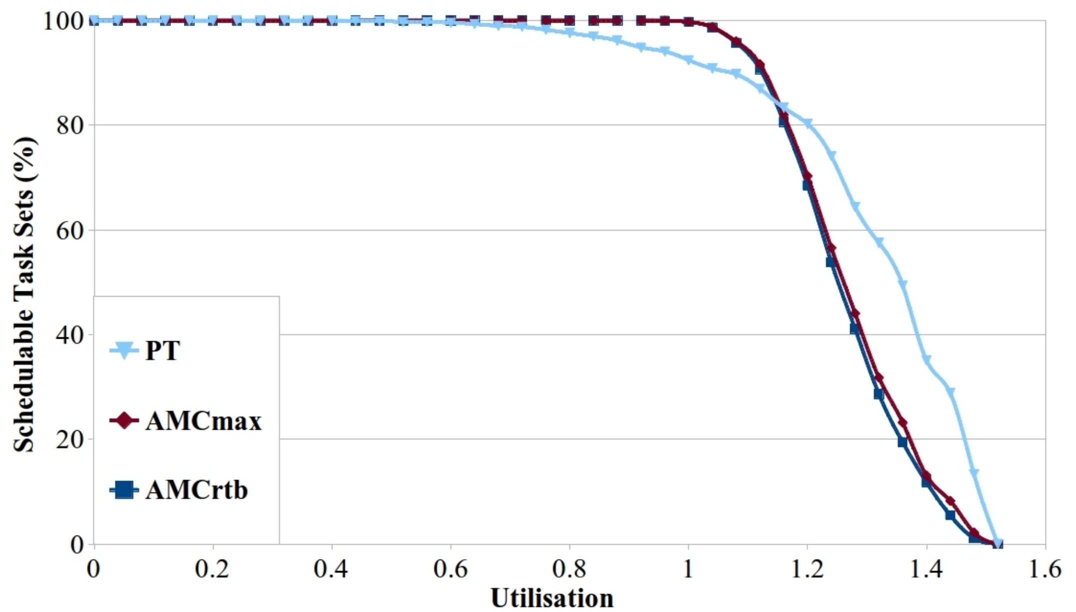

Figure 5.4: Period Transformation, AMCrtb and AMCmax: 5 Criticality Levels

Figures 5.3 & 5.4 show that Period Transformation performs well at lower numbers of criticality levels, however this advantage does not hold up as the

number of criticality levels increases. This is likely due to an increased complexity in the way in which tasks must be transformed when the number of criticality levels is greater. We do not account for any overheads in the graphs above, it is likely that performance would not be as good as shown due to the cost of splitting a task into its transformed version and enforcing this split during runtime.

5.6 Summary

In this chapter we have considered and extended Period Transformation for Mixed Criticality systems. Although its theoretical properties are not as attractive as traditional PT[2] they are effective, particularly with a lower number of criticality levels. The effectiveness of PT decreases as criticality levels are added, this coupled with a significant increase in overheads might make such a scheme impractical.

[2]Harmonic task sets/ Util ≤ 1 schedulability

Chapter 6

Evaluation

To better reinforce and illustrate the analysis presented in the chapters above an evaluation is included in the form of a set of experiments. These experiments set out to assess the performance of AMCrtb, AMCmax and Period Transformation against themselves and other schemes such as SMC. The experiments were performed with the aim of better supporting the properties of AMCrtb and AMCmax reported by Baruah et al. [10], and to determine whether the claims hold in systems with greater than two criticality levels. The Chapter is structured as follows; Section 6.1 discusses the process in which tasks were generated, Section 6.2 explains an adjusted definition of task set utilisation, Section 6.3 and 6.4 present the results, Section 6.5 provides a discussion and 6.6 summarises this Chapter.

6.1 Assumptions

For the experimentation we consider the highest criticality level to have a WCET of double the lowest. If other criticality levels exist between these two then their WCETs are evenly spread between those of the highest and lowest levels. For example, if a task in an MC system had a period of 10 for its LO criticality level and a period of 20 for its HI criticality level then period of its ME criticality level would be 15. The number of tasks within a task set, assigned to each criticality level will be as even as possible. If a completely even assignment is not possible then tasks are placed into the lower criticality levels. For example, a 10 task system with 4 criticality levels would consist of, criticality L4 with 3

tasks, L3 with 3 tasks, L2 with 2 tasks and L1 with 2 tasks, where $L1 > L4$.

6.2 Task Generation

To assess the performance of each algorithm some theoretical task sets were created. The properties of these task sets are randomly generated. To do this we first generate utilisations using the UUniFast algorithm [15] shown in Figure 13.

```
function vectU = UUniFast(n, U)
sumU = U;
for i=1:n-1,
        nextSumU = sumU.*rand^(1/(n-i));
        vectU(i) = (sumU - nextSumU);
        sumU = nextSumU;
end
vectU(n) = sumU;
```

Figure 6.1: UUniFast [15]

The UUniFast algorithm generates a random distribution of utilisations at whatever total task set utilisation is input.

The next stage is to generate Task Periods. These are created with a Log Uniform distribution creating periods between 10 and 1000, similar to those created in [10]. In this case we work with implicit deadlines, as such deadlines are equal to periods. By combining the Utilisations and Task Periods it is possible to generate worst case execution times. However, as we are dealing with mixed criticality systems and we can calculate only one set of WCET times from the Utilisations and Periods, we must decide the criticality level of these values.

Although we have established how the computation times at each criticality level will be determined (see 6.1), it is only possible to generate one initial value. We consider this value to be at the highest criticality level for all tasks, this value is then used to calculate the WCETs for each of the lower criticality levels following the rules set out above. All initial WCETs are at the highest level, regardless of the criticality level of the task itself, a value for each lower criticality level is calculated, and those unnecessary values for tasks with execution times for levels higher than their own are discarded.

6.3 Criticality Dependent Utilisation

In order to better represent tasks with a number of different worst case execution times we consider the notion of criticality dependant utilisation. It is the idea that the utilisation of a task is based on its execution time at its own criticality level. This leads to the unusual prospect of the total (criticality dependant) utilisation of a task set being greater than 1 but still schedulable. As long as no criticality level has a utilisation greater than 1 then the task set might still be schedulable. We define Criticality Dependant Utilisation in Equation [6.1].

$$U_i(L_i) = \frac{C_i(L_i)}{T_i} \tag{6.1}$$

We use this notion of utilisation for our experimental results.

6.4 Results

This subsection presents some of the results of the experimentation in the form of graphs showing the percentage of schedulable tasks for increasing total task set utilisations. These (criticality dependant) utilisations increase in steps of 2%. In this way it is possible to see the percentage of tasks each algorithm can schedule at each utilisation.

6.4.1 AMCrtb

In Section 3.4 we considered, briefly, some results from a comparison between AMCrtb and SMC. The graphs below in Figures 6.2, 6.3, 6.4 and 6.5 present these results for 2, 3,4 and 5 criticality levels.

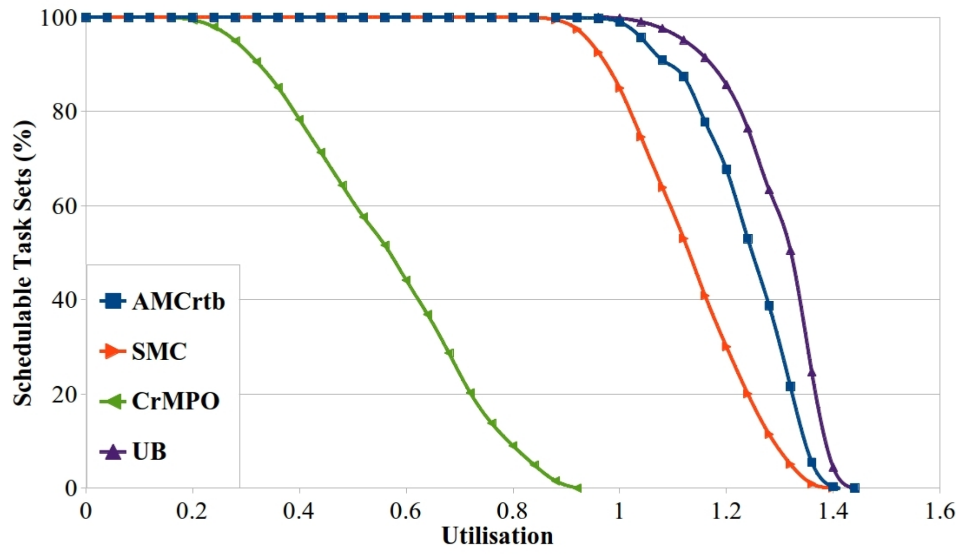

Figure 6.2: AMCrtb, SMC, UB & CrMPO: 2 Criticality Levels

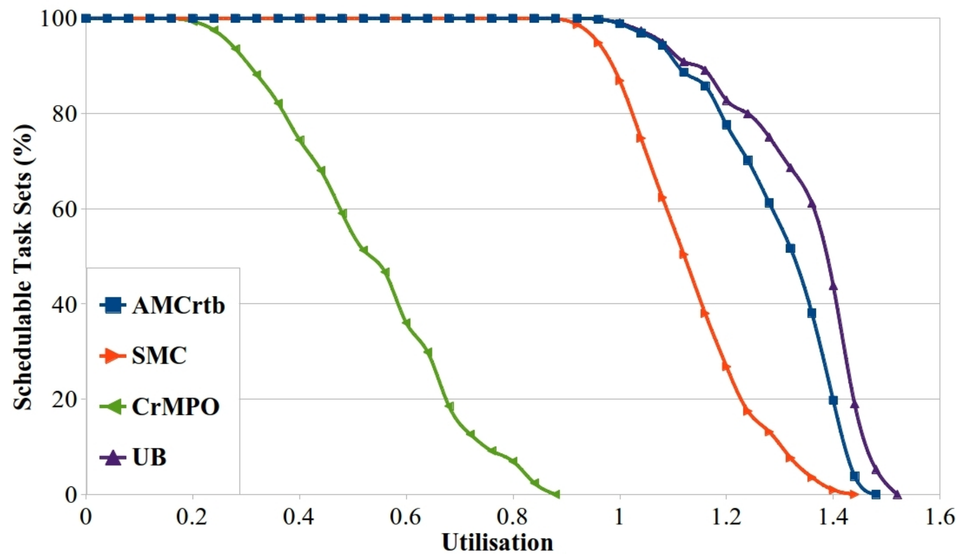

Figure 6.3: AMCrtb, SMC, UB & CrMPO: 3 Criticality Levels

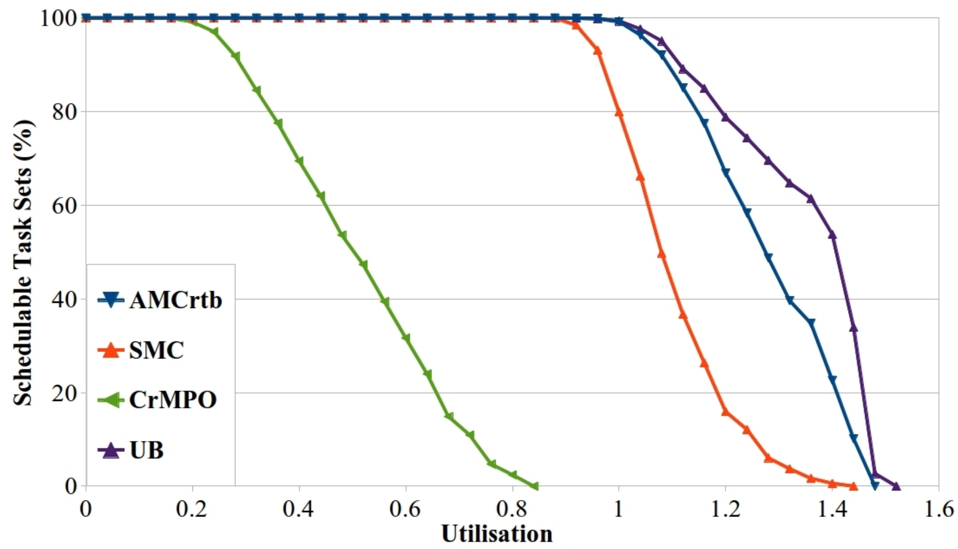

Figure 6.4: AMCrtb, SMC, UB & CrMPO: 4 Criticality Levels

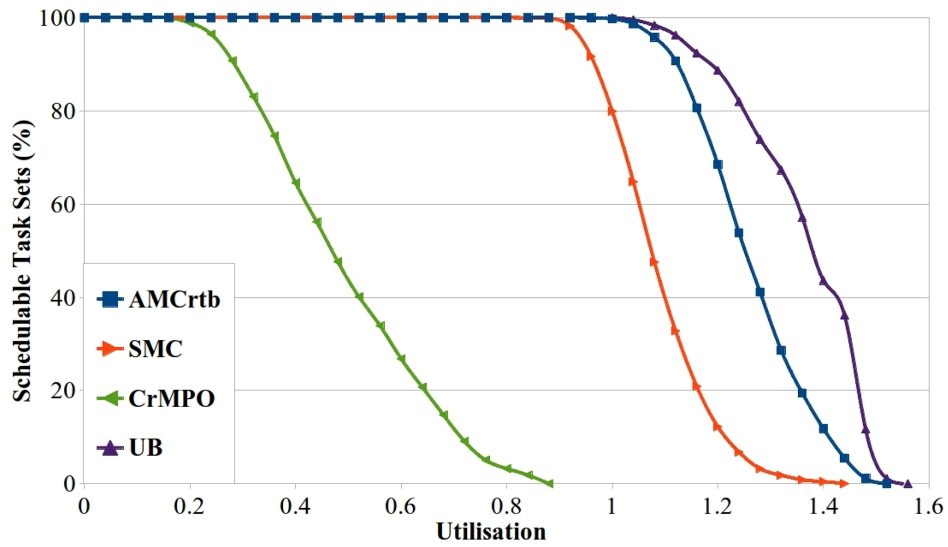

Figure 6.5: AMCrtb, SMC, UB & CrMPO: 5 Criticality Levels

As shown in subsection 3.4 AMCrtb remains the dominant algorithm. Also included in the graphs above are Criticality Monotonic Priority Ordering (CrMPO) and a composite upper bound similar to the UB-H&L bound used by [10]. This composite upper bound considers a task schedulable if each criticality level is

feasible, each level is considered in isolation and is assigned priorities in Deadline Monotonic order (as DM is optimal for a single criticality level). This provides an upper-bound that no FP algorithm can exceed.

AMCrtb's dominance over SMC is due to the way in which AMC handles criticality changes. AMC drops all tasks at its current criticality level and changes to the level above, those tasks dropped are assumed to be suspended indefinitely. SMC prevents a lower criticality task from interfering the the execution of the higher criticality tasks but takes no further action. This difference causes the significant increase in schedulability shown by AMCrtb.

6.4.2 AMCmax

Again we re-consider the short evaluation presented in section 4.3 for AMCmax including AMCrtb, CrMPO and UB. We look at results for 2 to 5 criticality levels and consider the performance of both of the AMC algorithms.

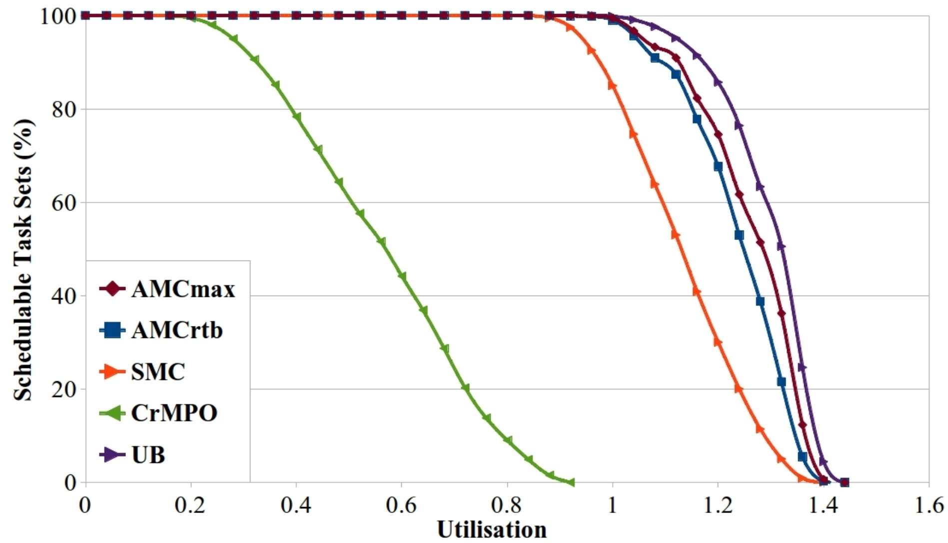

Figure 6.6: AMCmax, AMCrtb, SMC, UB & CrMPO: 2 Criticality Levels

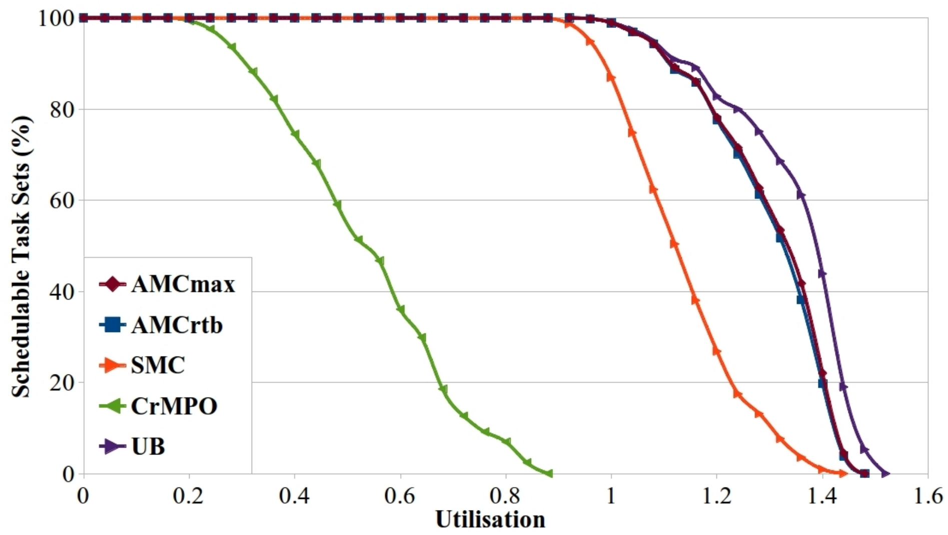

Figure 6.7: AMCmax, AMCrtb, SMC, UB & CrMPO: 3 Criticality Levels

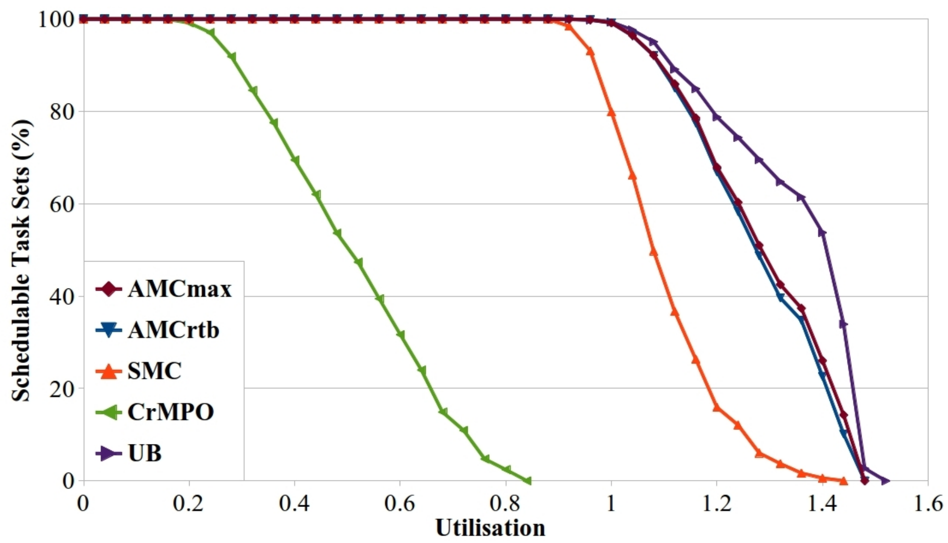

Figure 6.8: AMCmax, AMCrtb, SMC, UB & CrMPO: 4 Criticality Levels

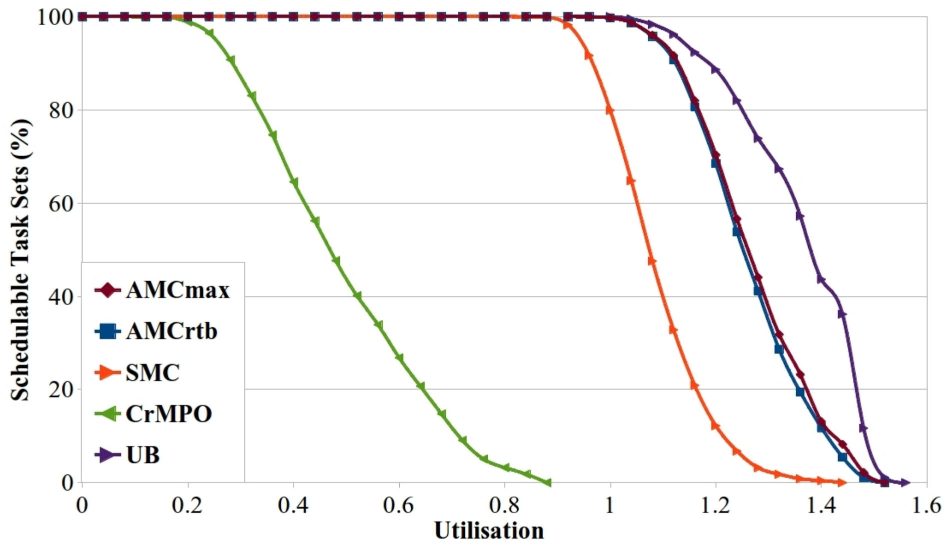

Figure 6.9: AMCmax, AMCrtb, SMC, UB & CrMPO: 5 Criticality Levels

Figures 6.6, 6.7, 6.8 and 6.9 show the performance of AMCrtb, AMCmax, CrMPO and UB from 2 to 5 criticality levels. As mentioned in section 4.3 it is clear that AMCmax remains more effective than AMCrtb. However the difference is slight and the computational intensity of AMCmax is far greater than that of AMCrtb. This is due to the notion of considering each point of s_2 within s_1 etc. This detailed search creates an almost exponential increase in complexity from one criticality level to the next, this quickly becomes prohibitively expensive to execute beyond 5 criticality levels. One might argue that most standards support no more than 5 criticality levels (SIL levels), therefore if additional accuracy is required, and there is the facility to perform intensive computation then AMCmax might be an effective choice. However the schedulability gains over AMCrtb are slight, potentially making AMCrtb a better choice in most cases.

6.4.3 Period Transformation

Here we consider the effectiveness of Period Transformation at 2, 3, 4 and 5 criticality levels. We compare PT with AMCrtb and CrMPO (Criticality Monotonic).

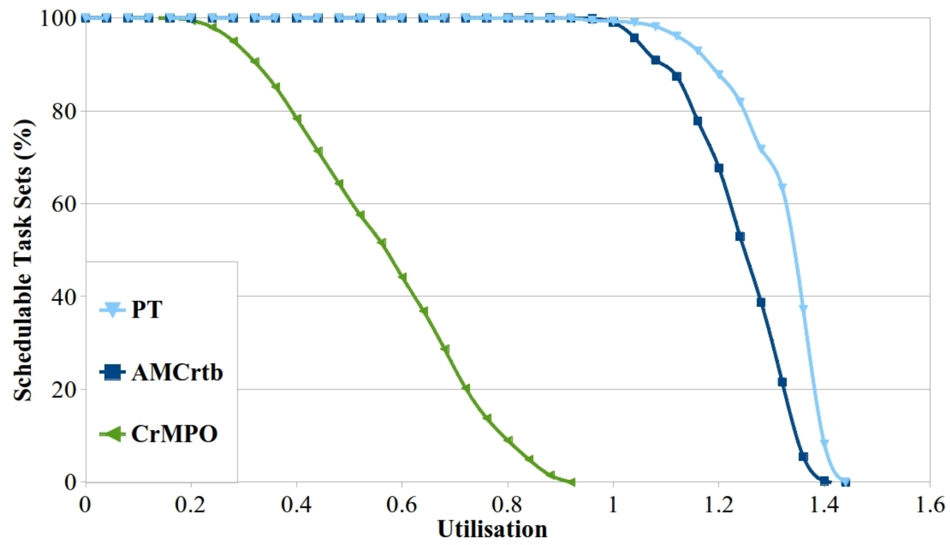

Figure 6.10: Period Transformation, AMCrtb and CrMPO: 2 Criticality Levels

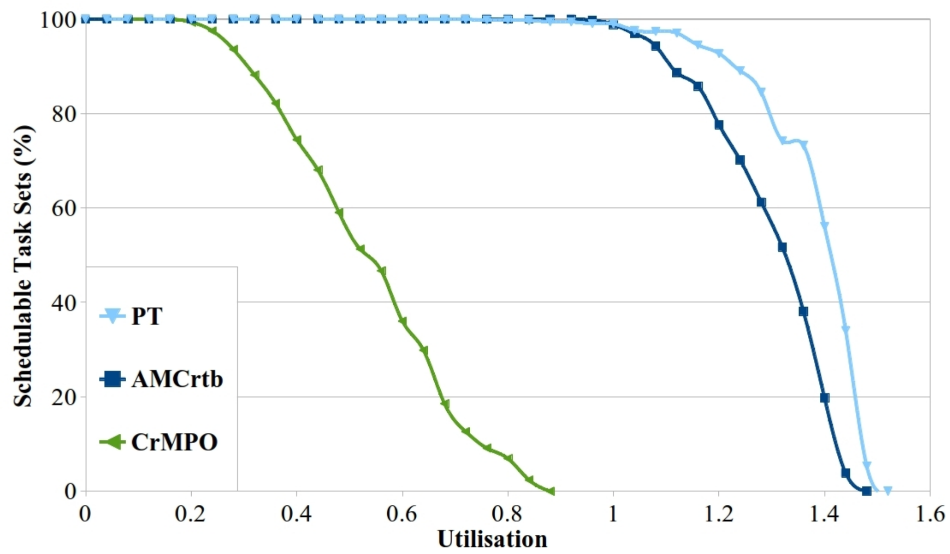

Figure 6.11: Period Transformation, AMCrtb and CrMPO: 3 Criticality Levels

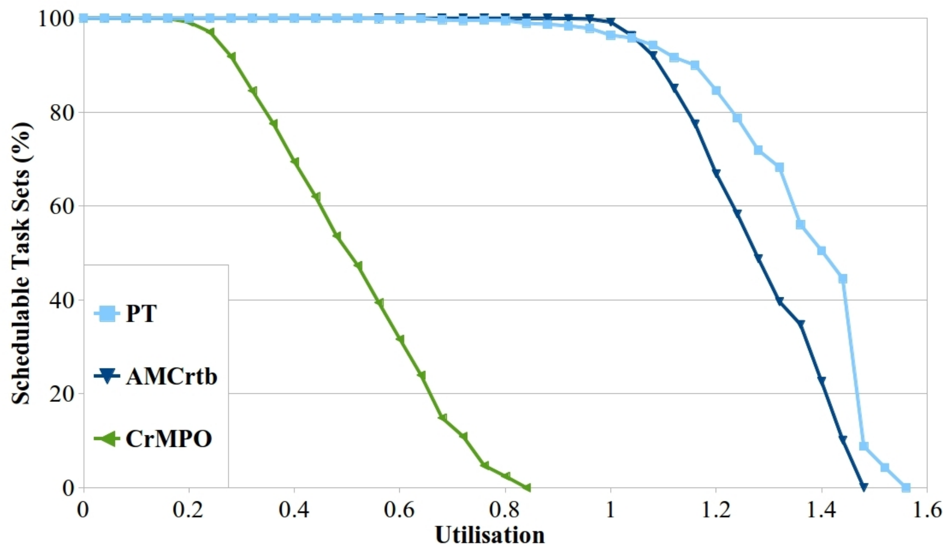

Figure 6.12: Period Transformation, AMCrtb and CrMPO: 4 Criticality Levels

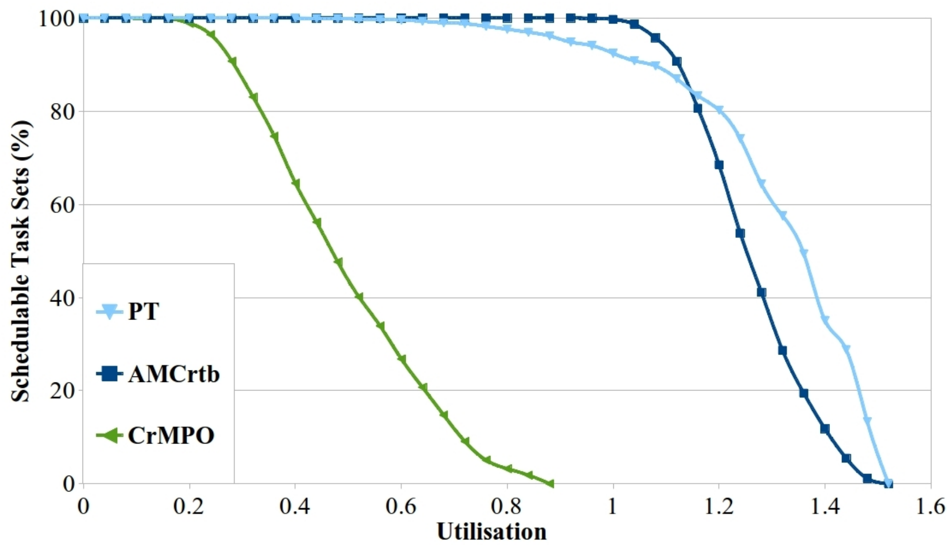

Figure 6.13: Period Transformation, AMCrtb and CrMPO: 5 Criticality Levels

Figures 6.10, 6.11, 6.12 and 6.13 show that PT performs strongly at two criticality levels, however this performance tails off as additional criticality levels are introduced. This is likely to be due to the increased complexity of the

transformation process. Of course, overheads are a key problem for Period Transformation, although performance might appear excellent, in reality it is likely to be far less effective.

6.5 Overall comparison

The graphs in Figures 6.14, 6.15, 6.16, and 6.17 show a comparison of all of the algorithms tested including, AMCrtb, AMCmax, CrMPO (Criticality Monotonic Priority Ordering), SMC-NO (Vestals Algorithm), SMC, Period Transformation and UB (Composite upper bound) across 2 to 5 criticality levels.

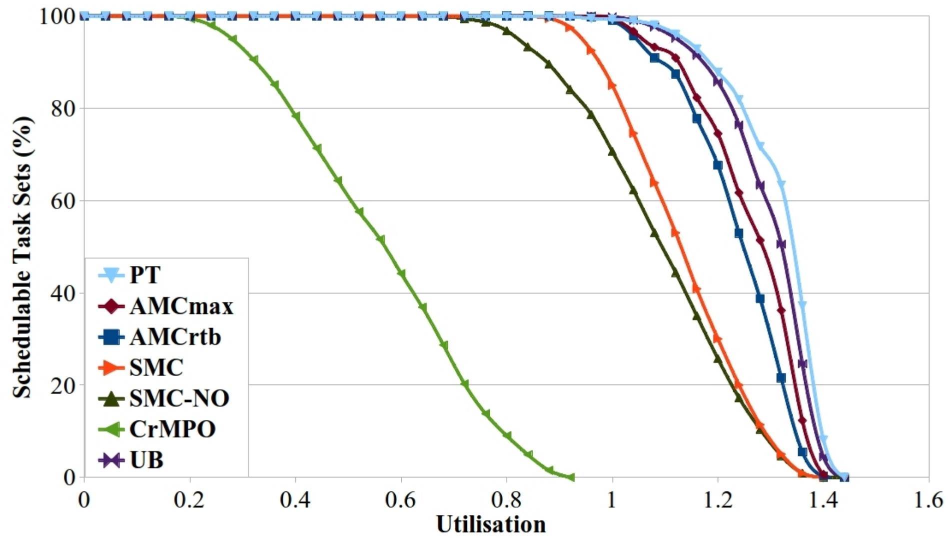

Figure 6.14: AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 2 Criticality Levels

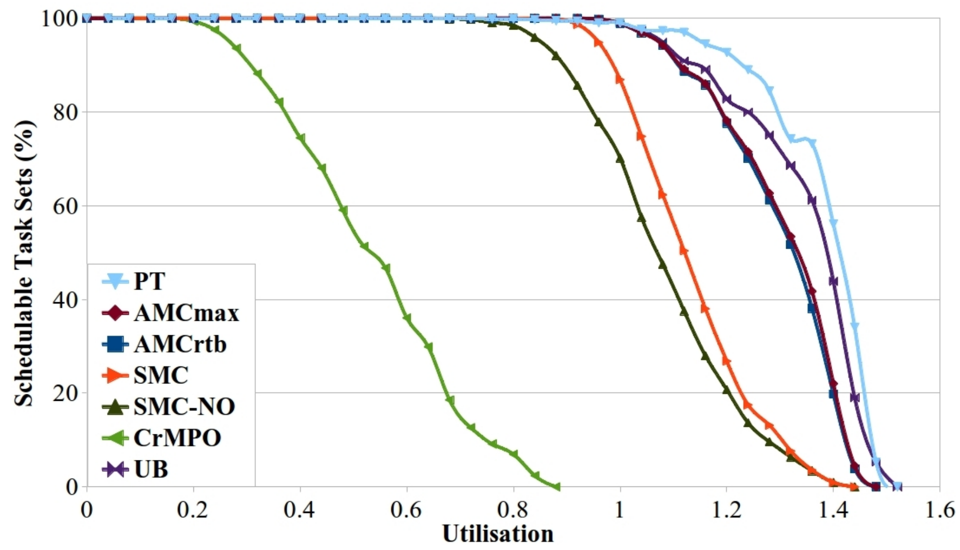

Figure 6.15: AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 3 Criticality Levels

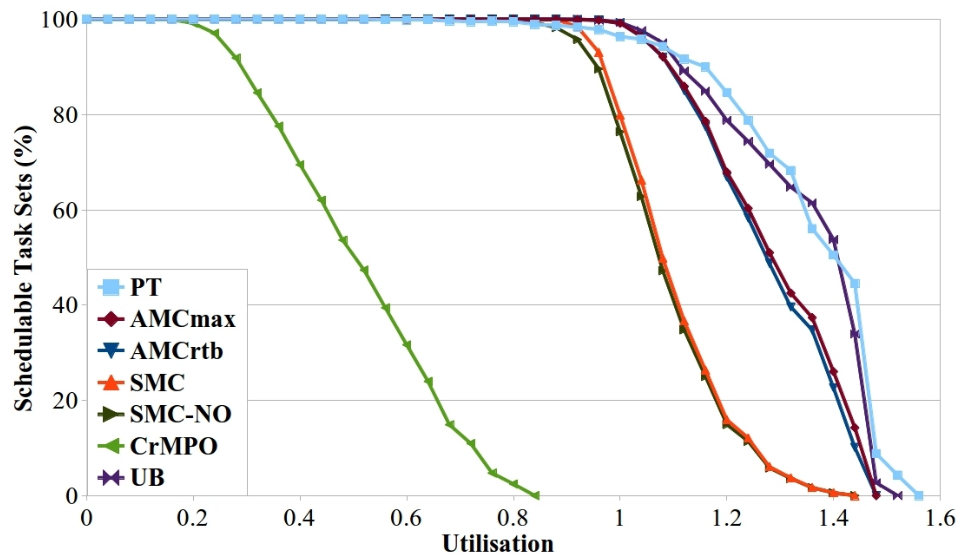

Figure 6.16: AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 4 Criticality Levels

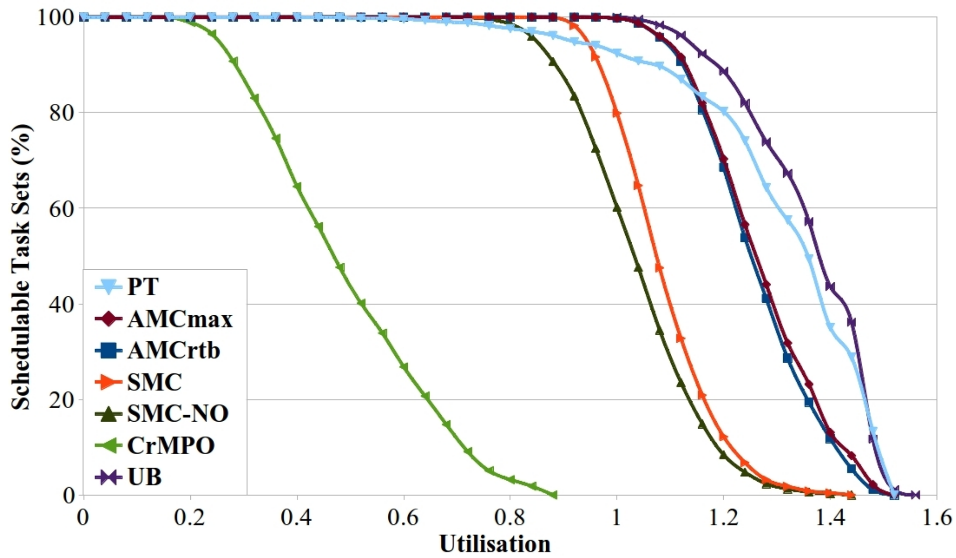

Figure 6.17: AMCrtb, AMCmax, PT, SMC-NO, SMC, UB & CrMPO: 5 Criticality Levels

It is clear to see that, of the standard fixed priority schemes, the AMC algorithms are the most effective and that they remain so as criticality levels are added. Although Period transformation does perform better at lower criticality levels, it suffers from large overheads as discussed in Section 6.3.3 and Chapter 5. Criticality Monotonic Ordering performs in a predictably poor manner. SMC-NO fairs slightly better however it's criticality dependant utilisation is limited to 100 as there is no criticality change behaviour and each task is considered at its own criticality level. Standard SMC is reasonably effective, but is outclassed by the AMC algorithms.

6.6 Discussion

There are a few features of the experimentation that warrant further discussion.

It is clear that varying the number of criticality levels is just one of the possible factors that might affect the efficiency and schedulability of the algorithms. One such factor might be altering the difference in Worst Case Execution Times between criticality levels. Although our experimentation assumes that the WCET at the highest criticality level is double that of the lowest, varying this difference is likely to change the schedulability of the task sets gen-

erated. Similarly the distribution of criticality levels within a system is likely to have an impact upon schedulability, we assume an even distribution (where possible), if this were not the case it is likely that the results might differ.

Another point worth noting is that it is not possible to directly compare the results of the experimentation across any two different criticality levels. This is due to the fact that although the size of the task set remains the same, the number of tasks per criticality level will differ as the number of criticality levels is increased. As such, two factors are varied, the number of criticality levels, and the number of tasks within each criticality level. Therefore the results are not directly comparable, it might even be the case that a higher criticality level appears to out perform a lower, this is as a result of the effect just described. Due to this the important results of this work is the effectiveness of the algorithms relative to their competitors, not their effectiveness across different criticality levels.

The results presented in this Chapter allow for a better gauge of the performance of each of the algorithms over 2 to 5 criticality levels. It backs up the analysis confirming the effectiveness of each algorithms with the addition of extra criticality levels. However the performance gained by the AMC based techniques comes at the cost of suspending lower criticality tasks as criticality changes occur. In practice this behaviour might be undesirable. While AMC appears to be an effective approach it is not without its downside.

6.7 Summary

In this Chapter we presented the results of our experimentation which considered each algorithm running at 2, 3, 4 and 5 criticality levels. We assessed the results and in particular confirmed that the performance of the AMC based approaches remains high.

Chapter 7

Conclusions

Throughout this work we have examined a number of scheduling approaches for Mixed Criticality Systems. Where applicable, these algorithms have been extended to facilitate an unknown number, n levels of criticality, extending the analysis from the Dual criticality focus of current literature. We have considered, in some depth, the implications of extending AMCrtb, AMCmax and Period Transformation to include greater than 2 levels of criticality.

The analysis and experimental results described in Chapters 3, 4 and 6 consider AMCrtb and AMCmax. They show that both AMCrtb and AMCmax continue to significantly dominate SMC. The results of the experimentation considering up to 5 criticality levels suggests that this dominance would be maintained up to n criticality levels. This strengthens the dominance of AMCrtb and AMCmax over SMC shown by Baruah et al. [10] for Dual criticality systems and gives further evidence of the extendibility of AMC. Both AMC approaches maintain the same levels of performance for many criticality systems as they do for dual criticality, in comparison to other approaches such as SMC.

In Chapter 4 we considered the implications and provide the analysis to extend AMCmax to greater than 2 criticality levels. It quickly became clear that whilst an extension to n possible levels of criticality is feasible, it is done at significant processing cost. The results of the experimentation shown in Sections 4.3 and 6.3 show the performance of the algorithm up to 5 levels of criticality, running these tests was intensive, running 6 or greater levels of criticality would not have been possible within a reasonable time frame. Although AMCmax does out perform and dominate AMCrtb at each criticality level, AMCrtb remains a very good approximation of AMCmax. As such AMCrtb is the most

practical form of analysis with room to increase the size of the task set or the number of criticality levels without seriously hampering the time it would take to generate the results. This extendability will become more valuable as Mixed Criticality systems develop in complexity. Current standards might only specify 4 or 5 criticality (SIL) levels, however future development might require a greater number of levels. In this case the ability to extend and run the analysis on greater than 5 levels, within a reasonable time frame, is important. AM-Crtb provides en effective approximation of AMCmax whilst maintaining a far greater level of efficiency, as such it is well suited to the rapidly changing Mixed Criticality domain.

We also examined Period Transformation in Chapter 5. In this chapter Vestal's Mixed Criticality Period Transformation is considered, some updates to his analysis are proposed in order to better consider the MC case, and to reduce the level of pessimism. The experimental data in Sections 5.5 and 6.3 show that Period Transformation performs well at 2 criticality levels, however performance tails off as criticality levels are added. It is well documented that Period Transformation, although being theoretically effective, is not practical. This is due largely to an excessive number of context switches and the need to closely manage transformed task executions. Not only is this impractical, but the overheads incurred could reduce the schedulability shown significantly. However, we do show that it is possible to adapt the Period Transformation approach to facilitate Mixed Criticality systems.

There can be no doubt that Mixed Criticality systems pose a challenging set of problems. In this work we have developed and provided evidence of the extendability and effectiveness of both AMC based analytical approaches. We have considered the use of Period Transformation and its Mixed Criticality extensions. Finally we presented an evaluation further detailing the performance of each scheme.

Bibliography

[1] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg. Multicore operating-system support for mixed criticality, 2009.

[2] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, 1991.

[3] N. 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(5):284–292, 1993.

[4] P. Axer, M. Sebastian, and R. Ernst. Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '11, pages 149–158, New York, NY, USA, 2011. ACM.

[5] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *Computers, IEEE Transactions on*, 53(6):781–784, 2004.

[6] S. Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In test, editor, *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 31 –38, june 2012.

[7] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 145 –154, july 2012.

[8] S. Baruah, V. Bonifaci, G. DAngelo, A. Marchetti-Spaccamela, S. Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In C. Demetrescu and M. Halldrsson, editors, *Algorithms ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 555–566. Springer Berlin Heidelberg, 2011.

[9] S. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2011*, volume 6652 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2011.

[10] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34 –43, 29 2011-dec. 2 2011.

[11] S. Baruah and B. Chattopadhyay. Response-time analysis of mixed criticality systems with pessimistic frequency specification. Technical report, University of North Carolina at Chapel Hill, 2013.

[12] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 3 –12, 29 2011-dec. 2 2011.

[13] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 13 –22, april 2010.

[14] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 147 –155, july 2008.

[15] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[16] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In C. Jones and J. Lloyd, editors, *Dependable and Historic Computing*, volume 6875 of *Lecture Notes in Computer Science*, pages 147–166. Springer Berlin Heidelberg, 2011.

[17] A. Burns and R. Davis. Mixed criticality on controller area network. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 125–134, 2013.

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

[19] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 135 –144, july 2012.

[20] P. Feiler, B. A. Lewis, and S. Vestal. The SAE architecture analysis & design language (AADL) a standard for engineering performance critical systems. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1206–1211, 2006.

[21] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 13 –23, 29 2011-dec. 2 2011.

[22] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson. RTOS support for multicore mixed-criticality systems. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, RTAS '12, pages 197–208, Washington, DC, USA, 2012. IEEE Computer Society.

[23] H.-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, RTAS '12, pages 23–32, Washington, DC, USA, 2012. IEEE Computer Society.

[24] B. Huber, C. El Salloum, and R. Obermaisser. A resource management framework for mixed-criticality embedded systems. In *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pages 2425 –2431, nov. 2008.

[25] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance [scheduling problems]. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 214–221, 1995.

[26] O. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *Trust, Security and Privacy in Computing*

and Communications (TrustCom), 2011 IEEE 10th International Conference on, pages 1051 –1059, nov. 2011.

[27] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig. Flattening hierarchical scheduling. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '12, pages 93–102, New York, NY, USA, 2012. ACM.

[28] K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 47 –56, april 2011.

[29] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, 1989.

[30] H. Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 166 –175, july 2012.

[31] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, J. A. Scoredos, and N. G. Corporation. Mixed-criticality real-time scheduling for multicore systems, 2010.

[32] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic. IDAMC: A many-core platform with run-time monitoring for mixed-criticality. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 24 –31, oct. 2012.

[33] M. Neukirchner, S. Stein, H. Schrom, J. Schlatow, and R. Ernst. Contract-based dynamic task management for mixed-criticality systems. In *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on*, pages 18 –27, june 2011.

[34] D. Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 291 –300, dec. 2009.

[35] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Embedded Software*

(EMSOFT), 2011 Proceedings of the International Conference on, pages 253 –262, oct. 2011.

[36] R. Pathan. Schedulability analysis of mixed-criticality systems on multiprocessors. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 309 –320, july 2012.

[37] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed-criticality in SoC-based real-time embedded systems. In *Proceedings of the seventh ACM international conference on Embedded software*, EMSOFT '09, pages 235–244, New York, NY, USA, 2009. ACM.

[38] A. Sangiovanni-Vincentelli. Quo vadis, SLD? Reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[39] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *RTSS*, pages 181–191, 1986.

[40] W. Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 11 –18, march 2011.

[41] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239 –243, dec. 2007.

[42] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 299 –308, july 2012.

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